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 9, 2021

Contents

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.

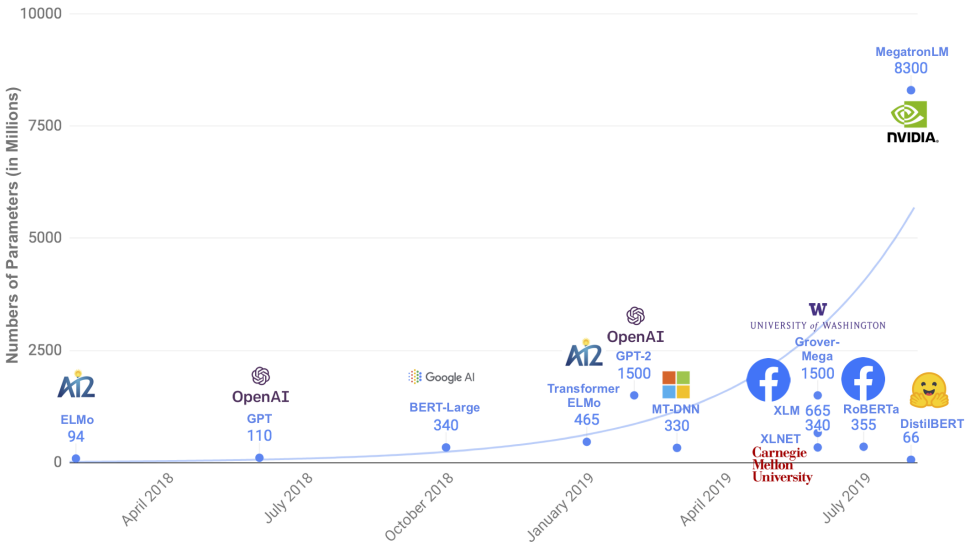

FIGURE 1: Parameter counts of recently released pre-trained language models which showed competitive or SOTA performance when fine-tuned over a range of NLP tasks; figure taken from Sanh et al. (2019)

1.2 Research questions

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

1. To what extent 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

With the aforementioned research questions, we summarize the structure and contents 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 and discuss the implications of the aforementioned results.

Chapter 6: Conclude this thesis by answering the research questions.

Chapter 7: Document 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."*

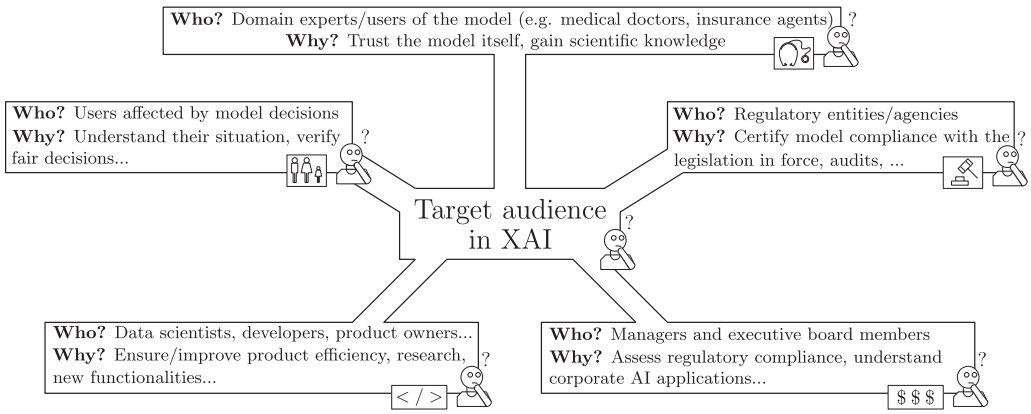

FIGURE 2: Examples of various target audiences in XAI; figure taken from Arrieta et al. (2020)

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.

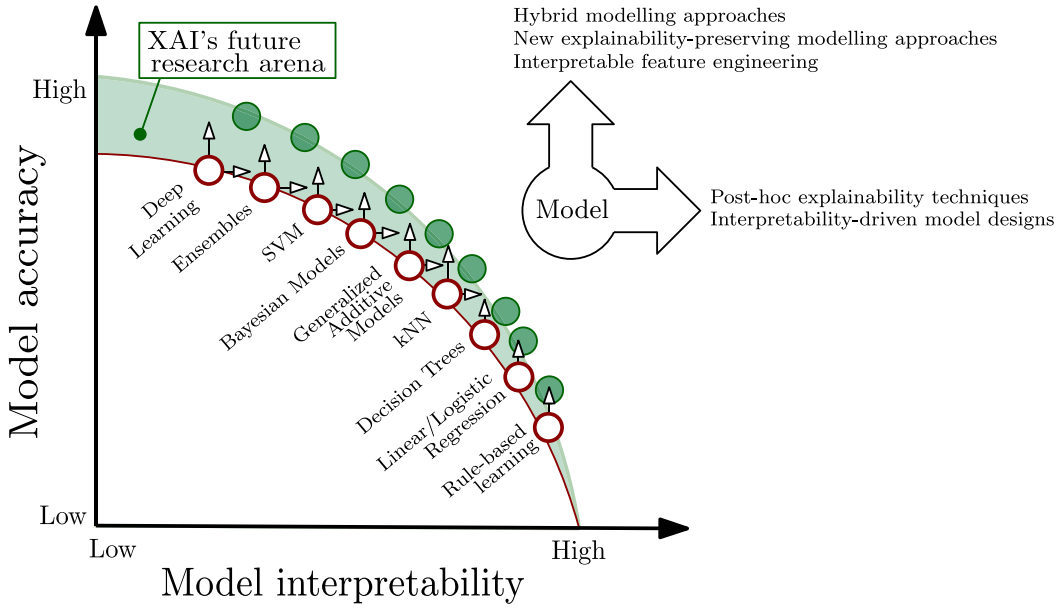

FIGURE 3: Qualitative visualization of the performance-interpretability tradeoff; figure taken from Arrieta et al. (2020)

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 method 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.

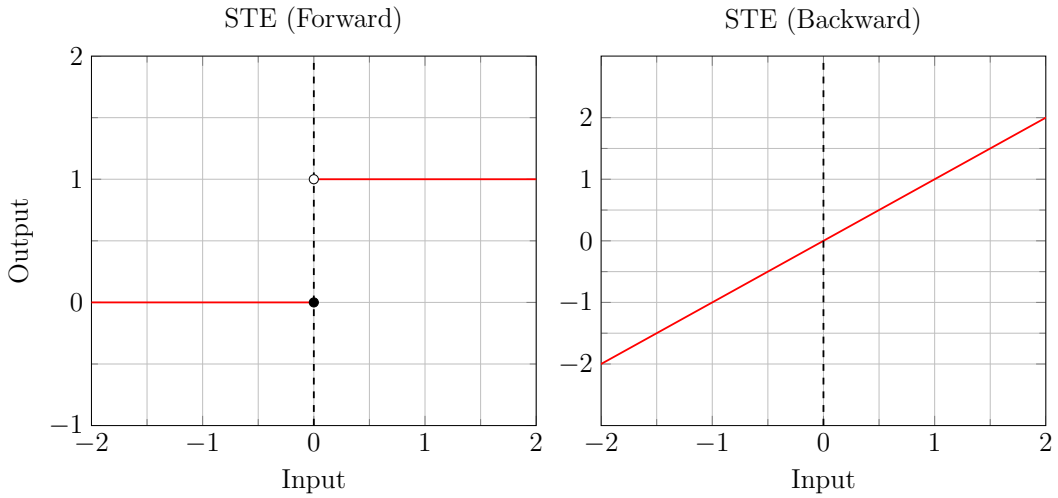

FIGURE 4: Visualization of the vanilla STE's forward and backward passes

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 Σ;
- 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 Σ;
- 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 symbol. 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.

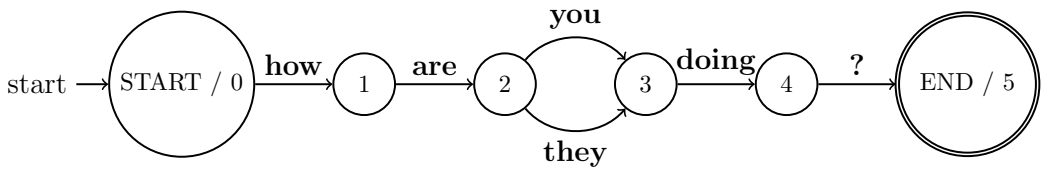

FIGURE 5: Regular expression ‘‘`how are (you|they) doing ?`’’ converted into a strict linear-chain NFA; double circles on state 5 indicate an accepting state

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 symbol.

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 $\mathcal{A} = \langle \Sigma, \mathcal{Q}, \mathbf{\Gamma}, \boldsymbol{\lambda}, \boldsymbol{\rho} \rangle$, with:

- a finite input alphabet Σ;
- 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. Σ^* refers to the (possibly infinite) set of all strings over the alphabet Σ, 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 ϵ-free string $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_m \rangle \in \Sigma^*$; which is a substring of the ϵ-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 ϵ-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 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 $\boldsymbol{x}$.

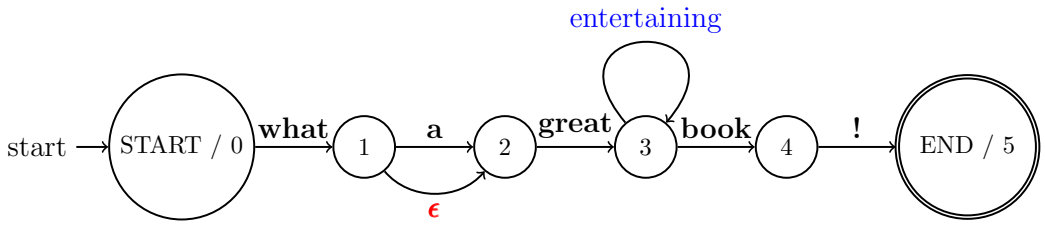

FIGURE 6: Visualization of a (non-strict) linear-chain NFA with self-loop (blue), ϵ (red) and main-path (black) transitions; figure adapted from Schwartz, Thomson, and Smith (2018)

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 **ϵ-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, ϵ-transitions are parameterized with the following representation in $\mathbf{\Gamma}$:

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

Here, c_i represents a learnable scalar bias for ϵ-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 ϵ-transitions and

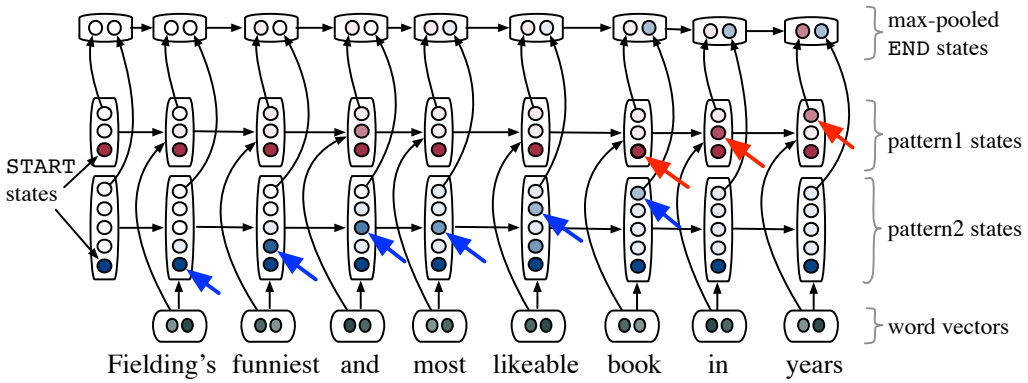

FIGURE 7: Visualization of the SoPa model framework with two constituent WFAs highlighted in blue and red; figure taken from Schwartz, Thomson, and Smith (2018)

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 ϵ-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 Pattern hyperparameter

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 pattern 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 methods 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 methods presented in Schwartz, Thomson, and Smith (2018) using terminology consistent with XAI research. The first explainability method 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 method. 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 method as a feature relevance explainability method. 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 methods 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 methods 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 methods 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 methods 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

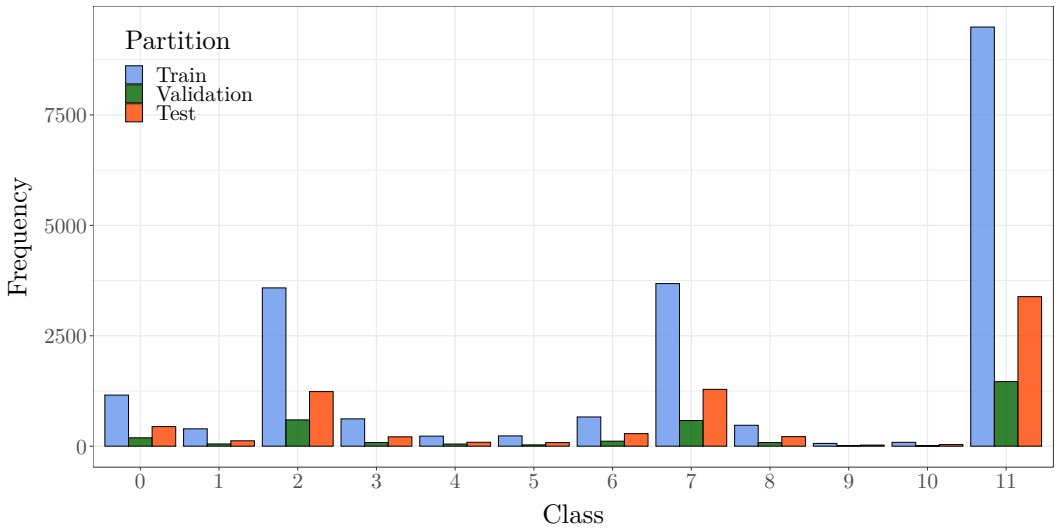

FIGURE 8: Frequency distribution of the preprocessed FMTOD data set by classes and partitions

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 ± 1.9	cancel weekly alarm
1: `alarm/modify_alarm`	7.1 ± 2.5	change alarm time
2: `alarm/set_alarm`	7.5 ± 2.5	please set the new alarm
3: `alarm/show_alarms`	6.9 ± 2.2	check my alarms.
4: `alarm/snooze_alarm`	6.1 ± 2.1	pause alarm please
5: `alarm/time_left_on_alarm`	8.6 ± 2.1	minutes left on my alarm
6: `reminder/cancel_reminder`	6.6 ± 2.2	clear all reminders.
7: `reminder/set_reminder`	8.9 ± 2.5	birthday reminders
8: `reminder/show_reminders`	6.8 ± 2.2	list all reminders
9: `weather/check_sunrise`	6.7 ± 1.7	when is sunrise
10: `weather/check_sunset`	6.7 ± 1.7	when is dusk
11: `weather/find`	7.8 ± 2.3	jacket needed?
μ	7.7 ± 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; μ 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-ω'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-ω'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 ϵ and self-loop transitions. As noted in Section 2.4.3, ϵ 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 ϵ-transitions and self-loops. We believe that this reduces the impact of SoPa's explainability methods 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 ϵ 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 ω-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-ω.

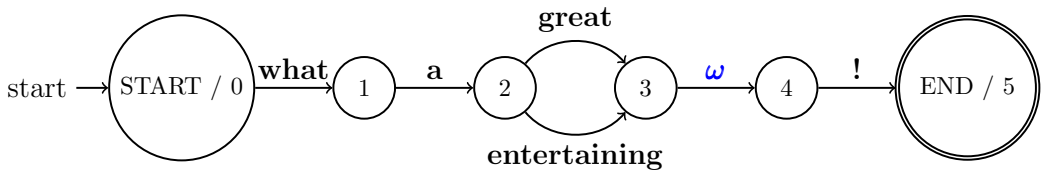

FIGURE 9: Visualization of a strict linear-chain NFA with ω (blue) and main-path (black) transitions

Definition 14 (Weighted finite-state automaton-ω)**.** A weighted finite-state automaton-ω 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 Σ;
- 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 ω 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 ω-transition and removal of the ϵ-transition, a WFA-ω 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-ω is similarly alloted a sequence of $|\mathcal{Q}|$ states. However, each state i in the linear-chain WFA-ω only has two possible outgoing transitions; namely a **ω-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-ω as strict linear-chain WFA-ω 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-ω's.

Next, we provide a mathematical formulation of the modified transition matrix $\boldsymbol{\Gamma}$ in our strict linear-chain WFA-ω. 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, ω-transitions are parameterized with the following representation in $\boldsymbol{\Gamma}$:

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

Here, c_i represents a learnable scalar bias for ω-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-ω document score*

Input(s): Strict linear-chain WFA-ω (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 ω 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-ω'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-ω 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-ω 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 ω-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-ω. 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

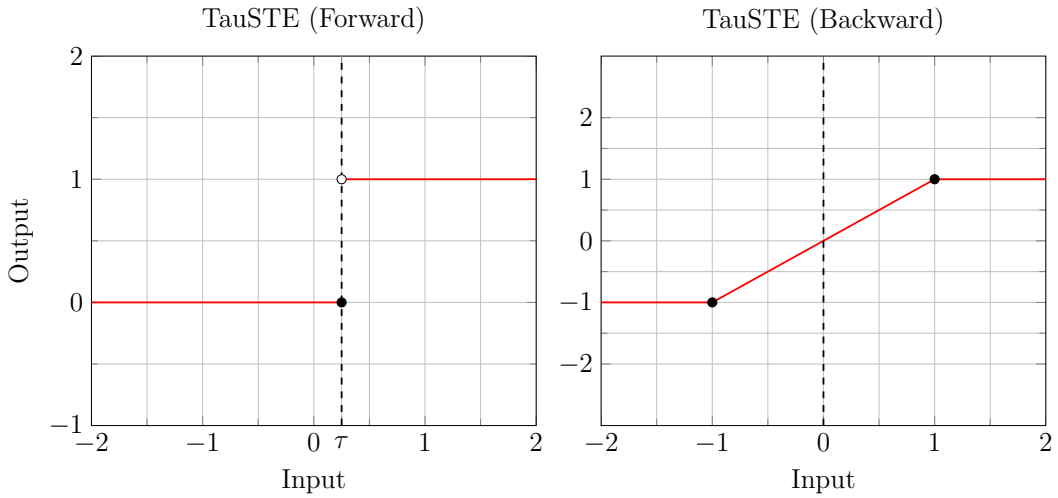

FIGURE 10: Visualization of the TauSTE's forward and backward passes

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 τ-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

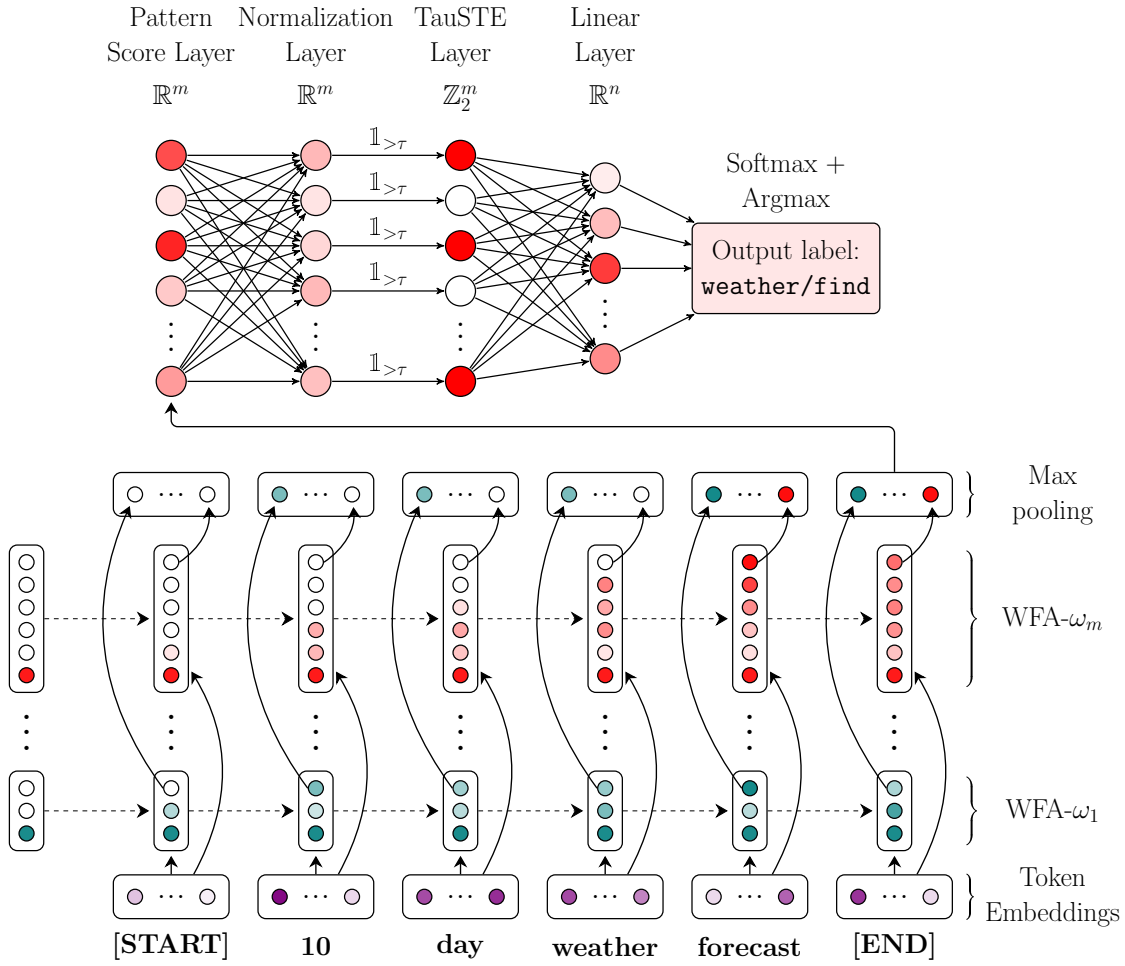

FIGURE 11: Visualization of the SoPa++ computational graph

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-ω'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 end states of each of the m strict linear-chain WFA-ω's and max-pool the score present in this state. In the edge case that an input string was too short for the end state of a WFA-ω 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-ω'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-ω and TauSTE neurons. For example, if we allow for $m = 40$ and therefore provide 40 WFA-ω 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-ω.

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 pre-processed 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-ω path-augmented document score*

Input(s): Strict linear-chain WFA-ω (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 ω 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 π_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-ω 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-ω.

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-ω's in $\mathcal{S}$
      $\mathcal{A}_j \leftarrow$ get_WFA($\mathcal{S}, j$) ▷ Get WFA-ω by index
      $s^j_{\text{doc}}(\boldsymbol{y}_i), \pi^j_{\text{doc}}(\boldsymbol{y}_i) \leftarrow$ DOCSCORE_PATH($\mathcal{A}_j, \boldsymbol{y}_i$)
      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 τ-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 $\boldsymbol{N_i}$ 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 a weighted decision. 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-ω when computing the path-augmented document score. Next, we can also observe the effect of the τ-threshold on the RE lookup layer. If the τ-threshold is low, we effectively allow more TauSTE neurons to be activated and therefore allow more regular expressions from the strict linear-chain WFA-ω to be saved in the RE lookup layer. Contrastingly, if the τ-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 τ-thresholds are better for performance; but a larger τ-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.

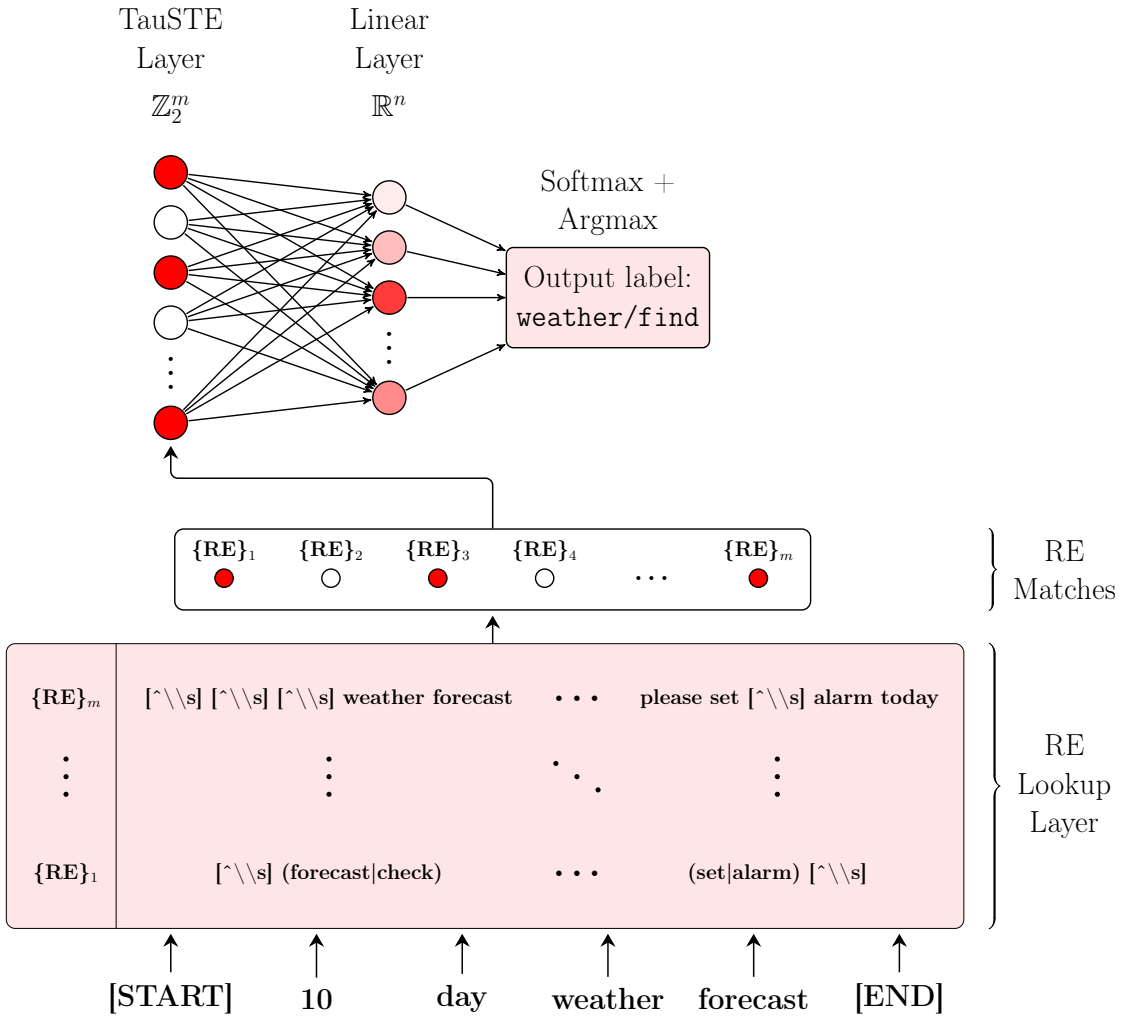

FIGURE 12: Visualization of the RE proxy computational graph

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 an empirical matter. 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 similar the RE proxy model is to its antecedent SoPa++ model.

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 regression 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 due to our unoptimized single-threaded RE lookup process.

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.

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

Characteristic	SoPa	SoPa++
Text casing	True-cased	Lower-cased
Token embeddings	GloVe 840B 300-dimensions	GloVe 6B 300-dimensions
WFAs	Linear-chain WFAs with ϵ, self-loop and main-path transitions	Strict linear-chain WFA-ω's with ω 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++

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 easily causally attributed to 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 modifications to SoPa++ include the modification of linear-chain WFAs to strict linear-chain WFA-ω's, the replacement of the MLP with layer normalization, TauSTE and linear layers and the significantly different 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	Pattern 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 pattern 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-ω's in SoPa++. We furthermore apply batch sorting based on input utterance lengths to ensure maximum efficiency when computing pattern/document 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-ω's.

To obtain some variation in our SoPa++ models, we decide to use a grid-search technique while varying the patterns P and τ-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 τ-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

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 important criterion that the proxy model should be less complex. We discuss more regarding the actual vs. theoretical transparency of the RE proxy models in the Discussions chapter.

Another key factor related to explanations by simplification is that the proxy model should be more explainable compared to its antecedent model. We explore this improvement of explainability using the three criteria for "good" explainability techniques as mentioned in Section 2.1.6. An evaluation based on target audiences is also important for explainability; with the target audience of SoPa++ and its RE proxy model largely being experienced ML practitioners. Due to limitations of this study, we are unable to address a detailed survey of explainability on our target audience and instead leave this aspect to future work.

We now describe techniques of evaluating explanations by simplification which address the performance scores and resemblance portions of Definition 6. We use the FMTOD test set to do this since the test set represents unseen data for both SoPa++ and the RE proxy, and therefore could provide a more unbiased representation of these models' performance. Furthermore, we conduct this analysis on our light, medium and heavy models with all available τ-thresholds to obtain a variety of results for comparison.

3.6.1 Performance score

As previously mentioned, an important aspect of evaluating explanations by simplification is to ensure that the proxy and antecedent models both have similar performance scores on a given task. Given this, we evaluate the accuracy of both SoPa++ models and their RE proxy counterparts on the FMTOD test set and compare their accuracies.

3.6.2 Distance metrics

As previously mentioned, another criteria for effective explanations by simplification is to maximum resemblance between the proxy and antecedent models. While comparing the performance scores is one way of doing this, this can also be seen as an aggregrate output-oriented comparison. We make an argument to test other distance metrics between SoPa++ and corresponding RE proxy model pairs. We describe these distance metrics and their mathematical formulations in this section.

Softmax distance norm: One method of quantifying a distance between the SoPa++ and RE proxy model pairs is to run both of them on the FMTOD test data set and obtain the softmax scores on each sample. Next, we can find the Euclidean norm of the difference of these two softmax vectors. A low softmax Euclidean distance norm would imply that the softmax distributions are 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 in that it allows us to analyze continuous vector differences. We provide a mathematical formulation of the softmax distance norm $\delta_\sigma(\boldsymbol{y})$ below for a single document $\boldsymbol{y}$ where $\sigma_\mathcal{S}$ and $\sigma_\mathcal{R}$ represent the softmax vectors for the SoPa++ and RE proxy model on the given sample respectively and n represents the number of classes in the FMTOD data set:

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

Binary misalignment rate: Another technique of measuring the distance between the SoPa++ and RE proxy model would be to compare their internal representations activated in the TauSTE binary layer. This can be computed by calculating the 1-norm between the binary vectors and then normalizing this norm by the dimensionality of this vector. The step of normalization is necessary since we test models with different sizes and these could have different binary vector lengths. This would provide a mean binary misalignment rate $\delta_b(\boldsymbol{y})$ for a single document $\boldsymbol{y}$, where $b_\mathcal{S}$ and $b_\mathcal{R}$ represent the TauSTE binary vectors for the SoPa++ and RE proxy model respectively and *dim* represents an operator that retrieves the dimensionality of a vector space:

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

3.7 RQ3: Interesting and relevant explanations

In this section, we describe the methodologies utilized to answer our third research question; particularly to show what interesting and relevant explanations the SoPa++ model can provide on the FMTOD data set.

3.7.1 Output neuron weights

Since SoPa++ uses a linear regression layer to provide weights and biases towards TauSTE outputs, we can analyze the weights of the linear regression layer to probe into how the SoPa++ and its RE proxies distribute importances to various binary features present in the TauSTE layer. This is a major advantage of using a single additive layer such as the linear regression layer, compared to using a MLP. To go about this, we simply perform a softmax operation over all weights applied for each class in each TauSTE neuron; and then visualize these to observe how SoPa++ and its RE proxies distribute feature importance.

3.7.2 Regular expression lookup layer

One major advantage of producing simpler RE proxy models is the presence of the RE lookup layer. This RE lookup layer can be seen as an ontology or database of

REs that lead to activations in the TauSTE layer. As a result, the RE lookup layer can be seen as an ontology of important regular expressions that aid SoPa++ and its RE proxies in making decisions based on input utterances. We analyze some of these REs juxtaposed next to their representative TauSTE neurons to visualize which REs activate which neurons and what kinds of importances are placed in said REs.

Chapter 4

Results

In this chapter, we focus on summarizing the results of our aforementioned methodologies. As a note, we do not dive deep into answering our research questions here. Instead, we simply use this chapter to report on our results and we answer our research questions in the next chapter.

4.1 RQ1: Evaluating performance of SoPa++

Following our methodologies, we conducted a grid-search training paradigm where we varied our patterns and τ-threshold parameters to obtain a total of 15 different modelling runs. Given that we repeated unique model run 10 times, we ultimately ran our grid-search for 150 model runs. This process took roughly 24 hours on a single NVIDIA GeForce GTX 1080 Ti GPU running light, medium and heavy model runs concurrently.

4.1.1 Training

We first describe the results of our training. Figure 13 shows the progress of the validation accuracy against training updates. The different coloured lines indicate the various initial random seeds assigned to the particulat model run. We can observe that increasing the τ value from 0 to 1 tends to decrease the overall validation accuracy profile. Correspondingly, we can also observe that the larger model tends to have a higher validation accuracy profile compared to the smaller models. We refer to the pattern hyperparameter as P in Figure 13. Finally, we can observe that the larger models tend to have an earlier convergence or early-stopping window compared to the smaller models.

4.1.2 Evaluation

In regards to the evaluation of SoPa++ performance, we can refer to Table 6 for a tabular summary of test accuracies across the light, medium and heavy model variants grouped by the various τ-thresholds. The exact specification of the light, medium and heavy model variants were described in Section 3.5.1. Here, we can observe that the best performing models were the heavy models with τ=0.0 and τ=0.25 and the medium model with τ=0.0. We can observe that test accuracies generally show a decreasing trend as we increase the τ-threshold. Correspondingly, we can observe a decreasing performance trend as we decrease the size of the model from heavy to light models. We can also observe that the standard deviations in performance are generally similar.

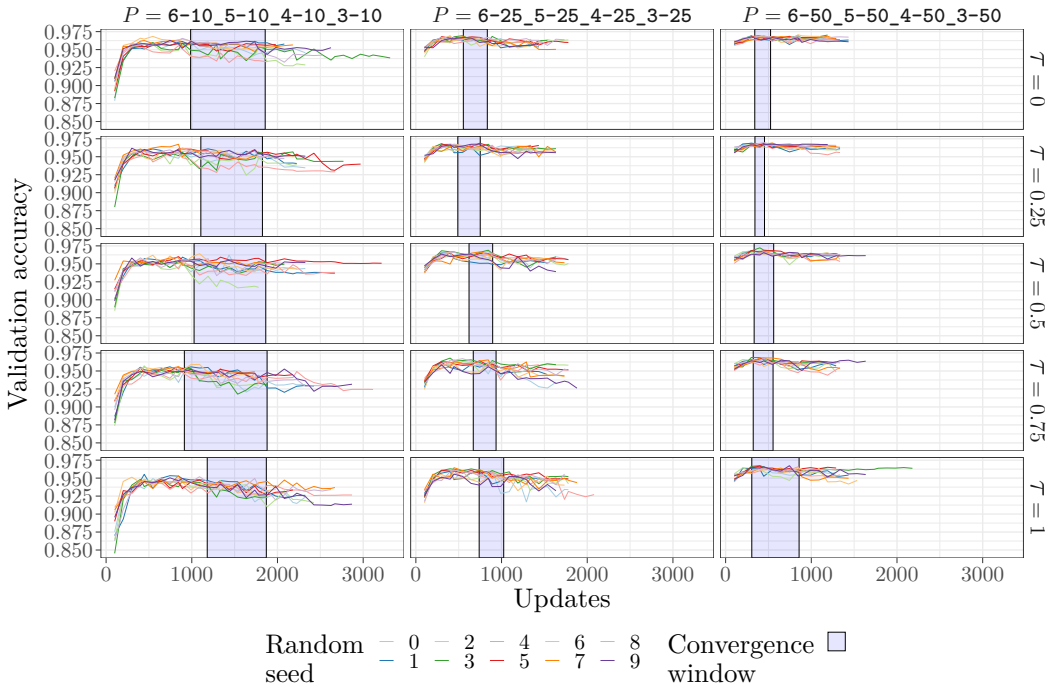

FIGURE 13: Visualization of validation accuracy against number of training updates for grid-search grouped by pattern hyperparameters and τ-thresholds

		Accuracy in % with mean ± standard-deviation				
Model	Parameters	τ=0.0	τ=0.25	τ=0.5	τ=0.75	τ=1.0
Light	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
Heavy	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 τ-thresholds; accuracies and standard deviations were calculated across random seed iterations; bold scores show best performing models

4.2 RQ2: Evaluating explanations by simplification

For evaluating explanations by simplification, we also follow our methodologies. Firstly, we refer to Figure 14 for a visualization of all results. We can observe a general trend that SoPa++ and RE proxy accuracies become more similar to one another as we increase the value of the τ-threshold. Similarly, we can observe a general trend that the mean model-pair distances $\overline{\delta_\sigma}$ and $\overline{\delta_b}$ both become smaller as we increase the value of the τ-threshold. In general, we can observe that the models with more parameters tend to have better test set accuracies for both SoPa++ and RE proxy models. Similarly, we can observe that the overall test set performance tends to decrease for SoPa++ but increase for the RE proxy models as the τ-threshold value is increased. Table 6 shows a tabular summary of test set accuracies and model-pair distance metrics for both SoPa++ and RE proxy models.

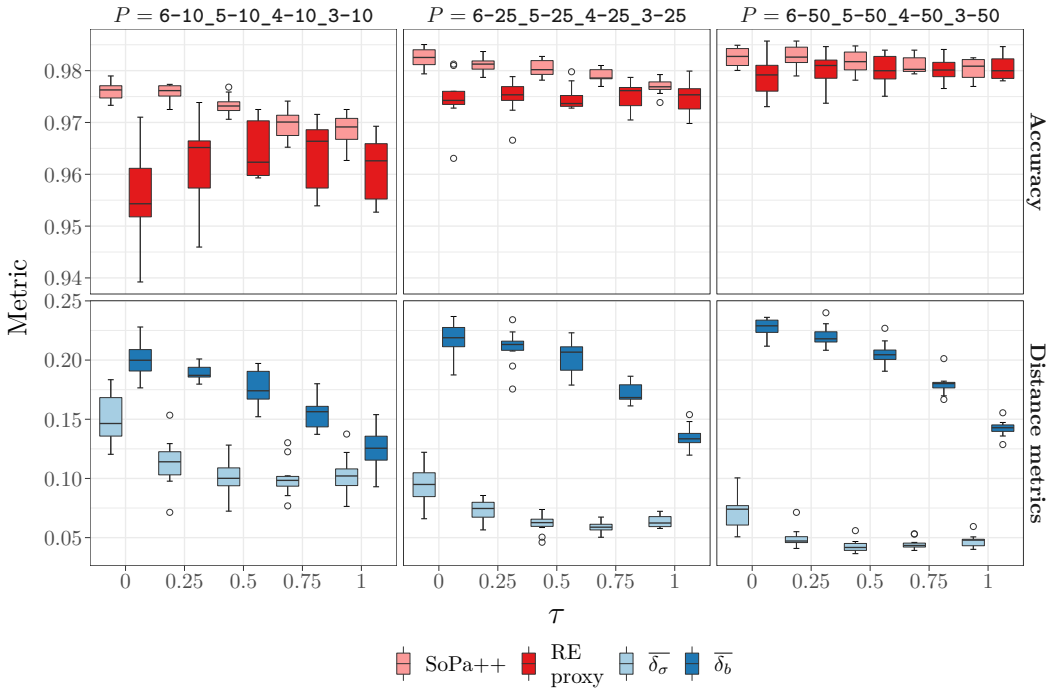

FIGURE 14: Visualization of accuracy and model-pair distance metrics against pattern hyperparameters and τ-thresholds

		Accuracy in % with mean ± standard-deviation				
Model	Variant	τ=0.0	τ=0.25	τ=0.5	τ=0.75	τ=1.0
Light	SoPa++	97.6 ± 0.2	97.6 ± 0.2	97.3 ± 0.2	$\mathbf{97.0 \pm 0.3}$	96.9 ± 0.3
	RE proxy	95.5 ± 1.0	96.2 ± 0.8	96.5 ± 0.6	$\mathbf{96.3 \pm 0.7}$	96.1 ± 0.6
Medium	SoPa++	98.3 ± 0.2	98.1 ± 0.1	98.0 ± 0.2	97.9 ± 0.1	$\mathbf{97.7 \pm 0.1}$
	RE proxy	97.4 ± 0.5	97.5 ± 0.3	97.5 ± 0.2	97.5 ± 0.3	$\mathbf{97.5 \pm 0.3}$
Heavy	SoPa++	98.3 ± 0.2	98.3 ± 0.2	98.2 ± 0.2	$\mathbf{98.1 \pm 0.2}$	$\mathbf{98.0 \pm 0.2}$
	RE proxy	97.9 ± 0.4	98.0 ± 0.3	98.0 ± 0.3	$\mathbf{98.0 \pm 0.2}$	$\mathbf{98.1 \pm 0.2}$
		Metric in % with mean ± standard-deviation				
Model	Metric	τ=0.0	τ=0.25	τ=0.5	τ=0.75	τ=1.0
Light	$\overline{\delta_\sigma}$	15.0 ± 2.3	11.3 ± 2.2	$\mathbf{10.0 \pm 1.6}$	$\mathbf{10.0 \pm 1.6}$	10.3 ± 1.8
	$\overline{\delta_b}$	20.1 ± 1.5	18.9 ± 0.7	$\mathbf{17.8 \pm 1.5}$	$\mathbf{15.5 \pm 1.3}$	12.4 ± 1.8
Medium	$\overline{\delta_\sigma}$	9.5 ± 1.7	7.3 ± 0.9	6.1 ± 0.8	$\mathbf{5.8 \pm 0.5}$	6.3 ± 0.5
	$\overline{\delta_b}$	21.7 ± 1.5	21.0 ± 1.6	20.3 ± 1.5	$\mathbf{17.2 \pm 0.9}$	13.5 ± 1.0
Heavy	$\overline{\delta_\sigma}$	7.1 ± 1.5	5.0 ± 0.8	$\mathbf{4.3 \pm 0.6}$	4.5 ± 0.5	4.7 ± 0.5
	$\overline{\delta_b}$	22.7 ± 0.9	22.0 ± 1.0	$\mathbf{20.6 \pm 1.0}$	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 τ-thresholds; accuracies and standard deviations were calculated across random seed iterations; bold test accuracies show model pairs whose accuracies are closest to one another; bold distance metrics show models whose $\overline{\delta_\sigma}$ metrics are closest to zero

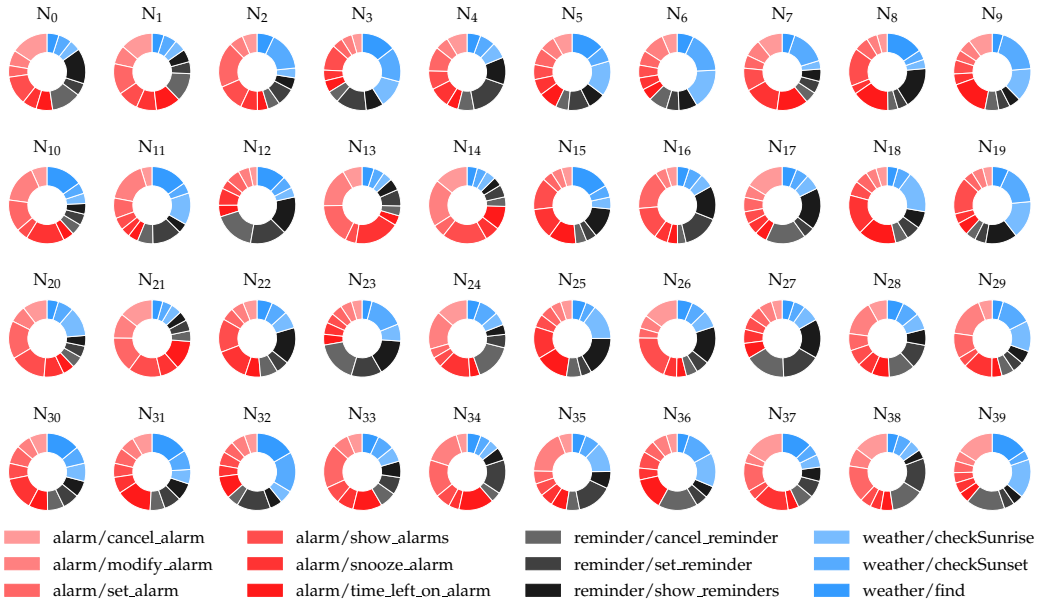

FIGURE 15: STE neuron relative-weights by output class

4.3 RQ3: Interesting and relevant explanations

Based on our methodologies for RQ3, we proceed to firstly analyze the weights of our linear regression layer and plot the weights provided to each neuron. This is shown in Figure 15 for a light SoPa++ or RE proxy model with 40 TauSTE neurons. We can observe that each of the 40 neurons distribute the weights provided for each class continuously; with certain neurons having more weights placed on certain classes than others. But in general, weights are quite continuously distributed. For ease of visualization, we grouped all of the 12 individual classes into three subclasses and coloured these as well. This makes the visualization slightly easier to understand.

Next, we single out neuron 21, 27 and 32 as they appear to place proportionately high importance weights on the alarm, reminder and weight sub-classes respectively. For the sake of further analysis, we probe into the RE lookup layer for REs that tend to activate the aforementioned three neurons. We visualize these regular expressions as strict linear-chain NFA with ω-transitions as shown in Figures 16, 17 and 18 respectively. It should be noted that we only sampled three regular expressions per neuron for brevity. There are however many more regular expressions that could be explored and analyzed. Superficially, we can observe that the regular expressions corresponding to these neurons tend to show contextual relevance to each of the aforementioned sub-classes.

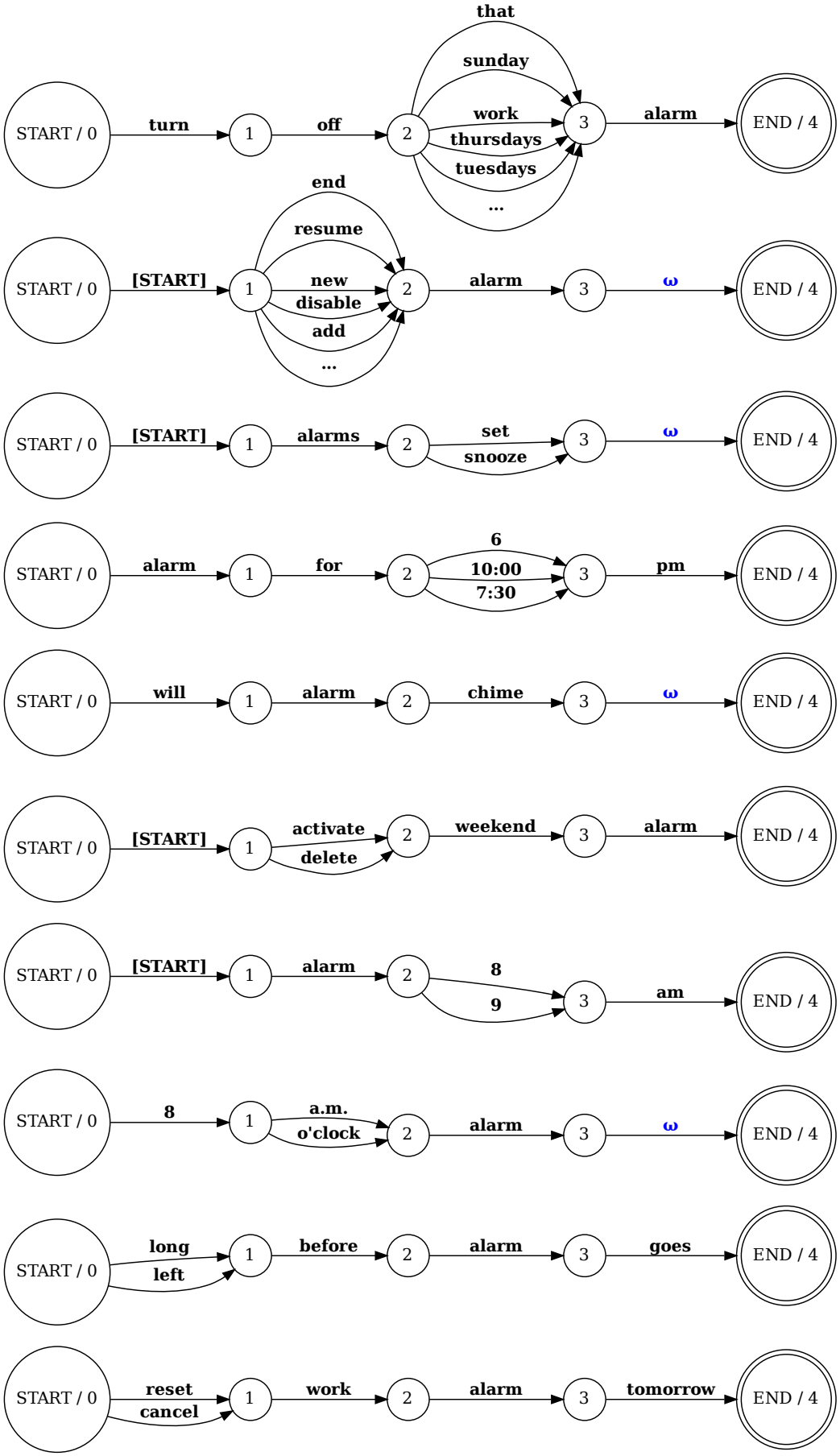

FIGURE 16: Sampled regular expressions corresponding to Neuron 21

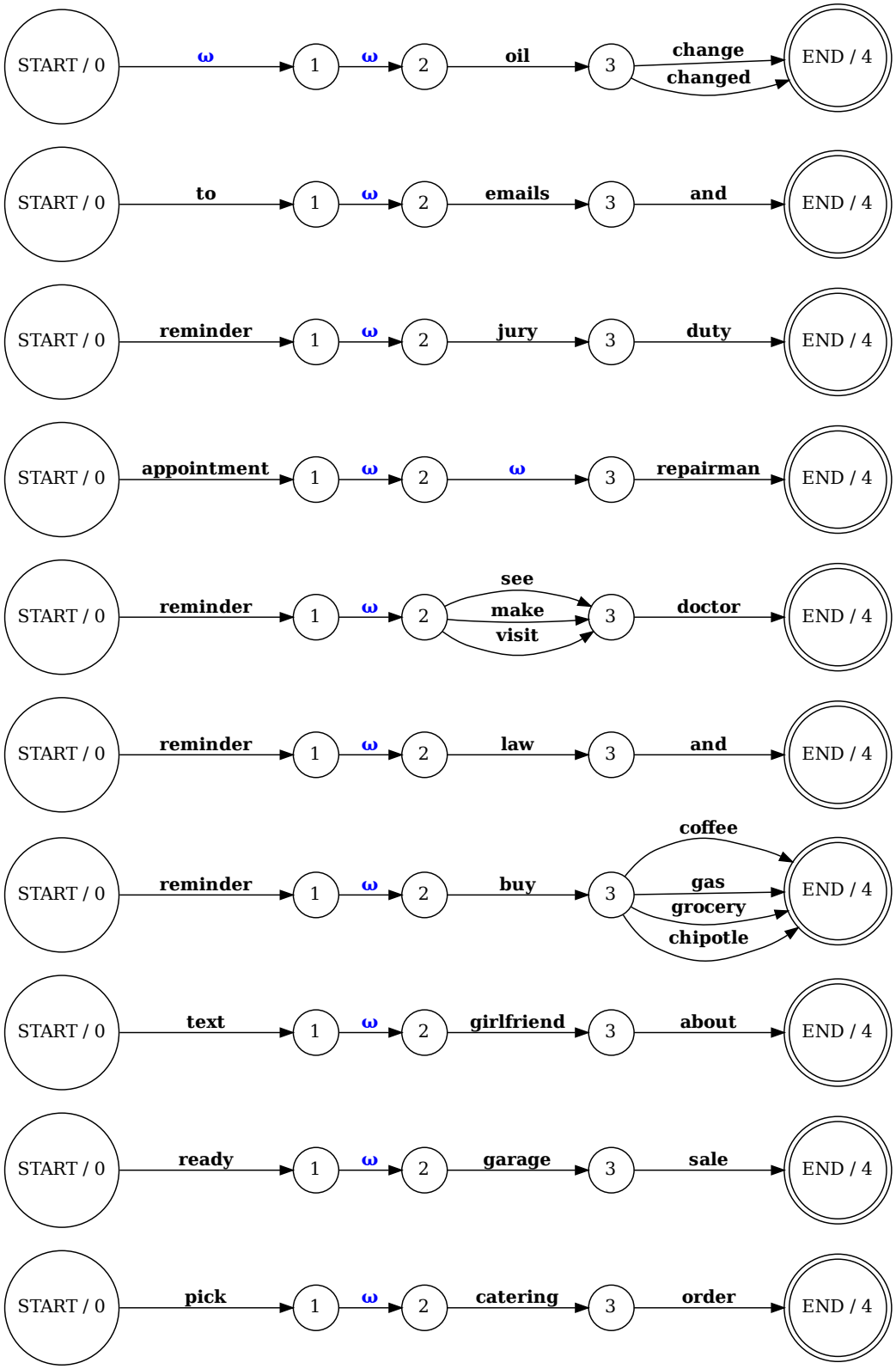

FIGURE 17: Sampled regular expressions corresponding to Neuron 27

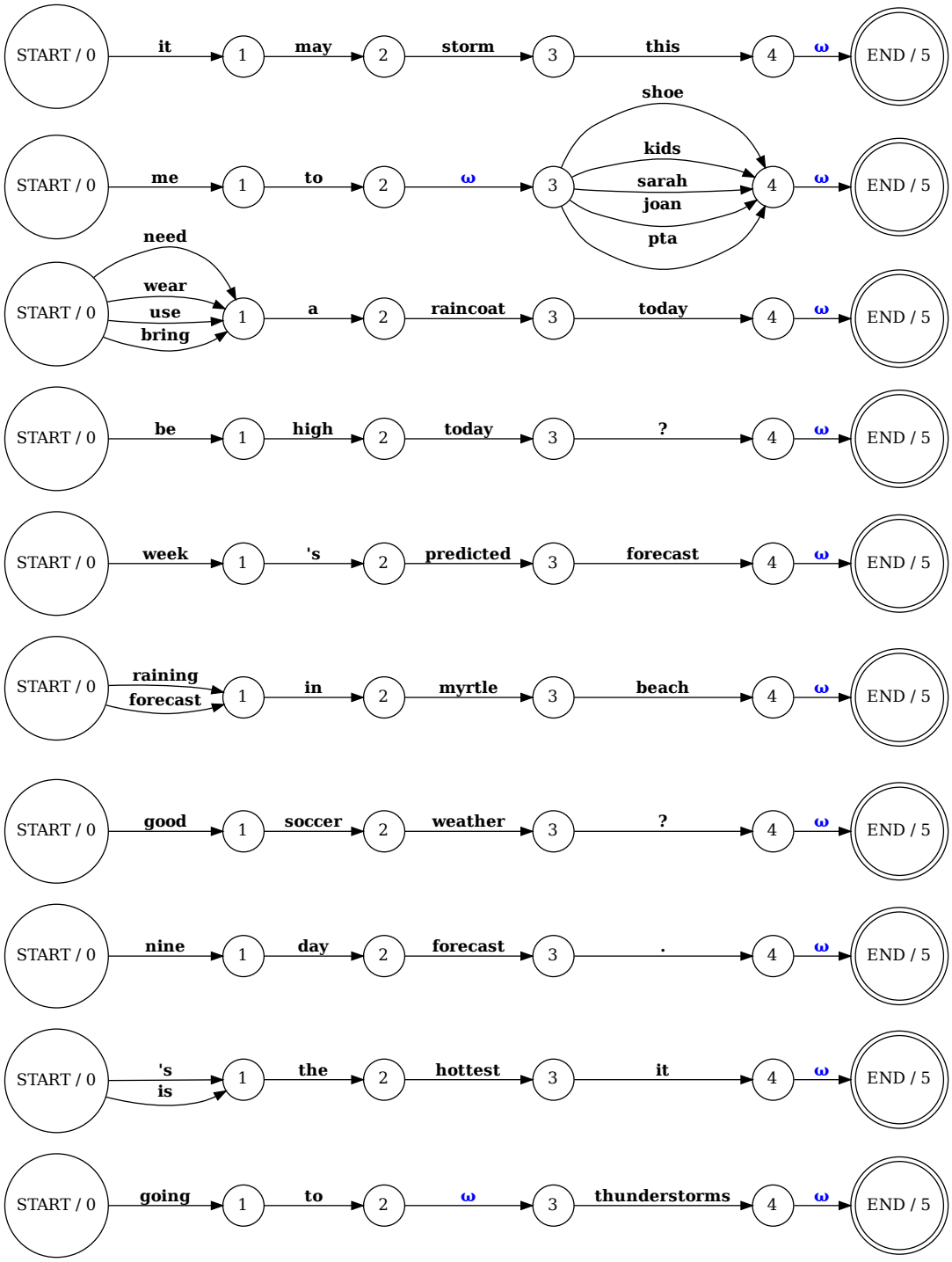

FIGURE 18: Sampled regular expressions corresponding to Neuron 32

Chapter 5

Discussion

In this chapter, we investigate the implications of the aforementioned results based on our methodologies and attempt to interpret these results in order to answer our research questions. Furthermore, we evaluate to what extent these results can answer our research questions and what limitations our methodologies pose in this regard.

5.1 RQ1: Evaluating performance of SoPa++

In order to answer our first research question on whether SoPa++ can deliver competitive performance for the English language intent detection task on the FMTOD data set, we must compare the general performance range of our SoPa++ model(s) against the results from other recent studies as mentioned in Section 3.1.3. Referring to our accuracy ranges from Table 6, we can observe the SoPa++ model shows a general accuracy range from 96.9-98.3% for the aforementioned task. This falls into the general performance range of 96.6-99.5% observed in other studies; albeit in the lower end of the performance spectrum. We can therefore conclude that SoPa++ offers competitive performance on the FMTOD's English language intent detection task compared to other recent studies.

While SoPa++'s performance range being on the lower spectrum can be seen as disadvantageous, it is worth noting that the models it is being compared against are vastly different. For one, the BERT models shown in 3 had parameter counts ranging from ∼110-340 million parameters (Devlin et al., 2019); which are ∼100-300 times larger than our SoPa++ model. It is therefore also useful to take these differences into account when evaluating and comparing model performances. Next, models from Zhang, Lyu, and Callison-Burch (2020) showed an exceptionally high accuracy of 99.5% because of pre-training on the external WikiHow data set for general intent detection tasks. As a result, it might be difficult to compare our results with that of Zhang, Lyu, and Callison-Burch (2020) since they utilized external specialized data to pre-train their models while we did not.

5.2 RQ2: Evaluating explanations by simplification

In order to answer our second research question on whether SoPa++ provides effective explanations by simplification, we need to first interpret some of the results on comparing SoPa++ and RE proxy model pairs. Firstly, we can compare the accuracy scores of SoPa++ and RE proxy model pairs as shown in the top portion of Table 7. Here, we can observe that the medium and heavy models with τ-threshold values of 0.75 and 1.00 result in SoPa++ and RE proxy model pairs with performance differences as low as ∼1-2%. Similarly, medium and heavy models with τ-thresholds of

0.50 and 0.75 show δ_σ metrics in the range of 4.3-5.8%. We interpret these results to imply that SoPa++ can provide effective explanations by simplification for models with more WFA-ω's and with τ-thresholds in range of 0.50-1.00. This is however not necessarily the case for smaller models or models with low τ-thresholds.

While our results show that conversion of SoPa++ to RE proxy models are generally effective with a minimal loss in performance and high resemblance given large models and high τ-thresholds; there are still certain limitations to our RE proxy models. While we did mention that the RE proxy models should theoretically be transparent models in Section 3.3.5, this could very easily no longer be the case given large RE proxy models with too many internal regular expressions. As an example, some of the RE lookup layers in heavy RE proxy models contain tens to hundreds of thousands of REs. Reading and understanding all of these internal REs might not be a practical task for a human, which may essentially render the theoretically transparent RE proxy model as a black-box model in a practical sense. This is a natural trade-off that we observe here.

5.3 RQ3: Interesting and relevant explanations

In this section, we interpret the results of probing into our best performing light RE proxy model in order to gain interesting and relevant explanations regarding the FMTOD English language intent detection data task. For one, we can analyze the TauSTE neuron-based weight distributions as shown in Figure 15 and observe that weights are generally continously distributed across all neurons; with some exceptions such as neurons 21, 27 and 32 where the weights are more skewed towards the alarm, reminder and weather sub-classes respectively. This implies that despite the quantization applied in the TauSTE layer, the SoPa++ and RE proxy models still distribute feature importances across neurons in a connective sense; which also implies that the purpose of each neuron in making any decision would be difficult to understand since each neuron would have a mixed impact on different classes. This could possibly be an impediment to explainability since clear causes and effects between neurons and output classes would be hard to identify.

Next, we can observe the RE samples from the RE lookup layer pertaining to neurons 21, 27 and 32 in Figures 16, 17 and 18 respectively. Firstly, we can observe a clear stratification of the types of REs captured between neurons 21, 27 and 32. As mentioned earlier, neurons 21, 27 and 32 specialize in alarm, reminder and weather sub-classes respectively. Similary, the REs captured for each of these neurons tend to show high similarity for each of these sub-classes. This could imply that certain neurons which place high weights on a particular sub-class set tend to gather similar REs as well.

Next, we can observe clustering or branching of REs in the RE lookup layer. For example, the top-most RE in Figure 16 shows that the third transition capturing many different words. This branching can also be observed in many other RE examples. Finally, another interesting phenomenon is that this branching of RE transitions tend to have semantic similarities as well. For example using the aforementioned RE example, we can observe words such as "sunday" and "thursdays" occur in the same transition and could therefore imply that days of the week trigger a high score in this transition.

Lastly, we can also observe certain inductive biases that the RE proxy model (and correspondingly the SoPa++ model) has acquired from the training data set. For example, the fourth and eighth REs from the top in Figure 17 show transitions with

the word "repairman" and "girlfriend" which led to high path scores. While these words were indeed relevant in both the training and test data set, the utility of a different gender such as "repairwoman" or "boyfriend" might not lead to the same model decisions since these words could be either outside of the model's vocabulary or unseen and therefore poorly weighted. A major advantage of the SoPa++ and its RE proxy model is that the user can have direct access to these representative REs in the RE lookup layer and can theoretically parse through these and adjust these "problematic" REs to adjust the inductive bias. One way to adjust the inductive biases in this case could be to add branching transitions with the different gender roles in the above single-gender cases. This is an advantageous component of the SoPa++ model framework compared to other deep learning models without explanations by simplification capabilities, since it would be difficult if not impossible to easily discover such inductive biases within these black-box models.

Chapter 6

Conclusions

In this thesis, we emphasized the importance of XAI research and laid out clear definitions of XAI-related concepts. We drew inspiration from the SoPa model in Schwartz, Thomson, and Smith (2018) and worked on improving the SoPa model to create our SoPa++ model. We focused on targetted changed to the SoPa model in order to create a SoPa++ model that allows for the ability to conduct the post-hoc explanations by simplification explainability method; where we ultimately simplify a SoPa++ model into a RE proxy model. This ability can be largely attributed to the special features of WFAs as well as the flexibility of the TauSTE layer. With all of these features, we were able to answer our research questions.

Firstly, we were able to show that SoPa++'s performance range of 96.9-98.3% falls into the competitive accuracy range of 96.6-99-5% on the FMTOD English language intent detection task while comparing with other recent studies. In this respect, we conclude that SoPa++ offers competitive performance on the aforementioned data set and task.

Next, we address to what extent SoPa++ allows for effective explanations by simplification. In order to address this question, we analyzed a variety of performance scores and distance metrics for SoPa++ and RE proxy models. We were able to show that larger models with higher τ-thresholds tend to have highly similar performance scores within $\sim$1-2% accuracy and mean softmax distance norms as low as $\sim$4-5%. As a result, we show that explanations by simplification are effective especially under the aforementioned special conditions where SoPa++ models are large with high τ-thresholds.

Finally, we address our third research question by probing for interesting and relevant explanations for classification on the FMTOD English language intent detection data set. We analyze light variants of SoPa++ and RE proxy model pairs by looking at their linear weight distributions in TauSTE neurons. Based on this distribution, we analyze captured activating REs in the RE lookup layer corresponding to these neurons and found interesting captured REs. These REs capture semantic and syntactic information which reflect important phrases for reaching classification decisions. Furthermore, we were also able to identify some inductive biases in the REs captured, such as those related to gender.

Chapter 7

Further work

In this final chapter, we address the limitations of our thesis and provide ideas for future research which could address these limitations.

7.1 Efficiency

In this thesis, we showed the effectiveness of simplifying SoPa++ models into RE proxy models. In the course of our experiments, we realized that while this process was effective; the simplification and evaluation processes using RE proxy models tended to be slow. This is largely because we used single-threaded processes for both simplification and evaluation of RE proxy models; in contrast to PyTorch's parallelized functionalities on a GPU which made SoPa++ very fast. To overcome the low efficiency of simplifcation and evaluation using RE proxy models, we could recommend two 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 utilize GPU-accelerated regular expression matching algorithms to parallelize the overall RE lookup layer and its searching functionalities (Wang et al., 2011; Zu et al., 2012; Yu and Becchi, 2013).

7.2 Explainability

As mentioned in Section 3.6, we were only able to address the technical requirements of explanations by simplification without necessarily delving into how good the explanations provided are for given target audiences. This process is subjective and would require a thorough survey with a given target audience. Conducting such a survey of quality of explanations could be useful to further evaluate the explainability of SoPa++ and its RE proxy counterparts.

7.3 Discovery and correction of inductive biases

As shown in Section 5.3, we provided examples of how a human could manually find and correct inductive biases in the RE lookup layer. It would be interesting to explore how extensively this process could be conducted to find and correct inductive biases in the REs inside the RE lookup layer. Furthermore, it would be interesting if we could also find adversarial samples in this layer. While correcting the RE lookup layer "fixes" problems on the RE proxy model's side, it does not modify the functionality of the antecedent SoPa++ model. It would therefore also be interesting to explore how we could propagate the corrections in the RE lookup layer back into the SoPa++ model's neural network components.

7.4 Generalization

Based on sampled regular expressions shown in Figures 16, 17 and 18, we can observe certain transitions which contain many possible words. We can conduct semantic analyses on these highly utilized transitions and perhaps generalize these to include more tokens which may even be outside of the initial model vocabulary. For example, in the fourth RE from the top in Figure 16, we can observe that the third transition generally captures numbers formatted as time. We could generalize this transition to capture any tokens that express the time; which could lead to generalization on previously unseen data instances. This could also address issues of unknown tokens altogether.

7.5 Modeling extensions

Since our current SoPa++ passes binary outputs from the TauSTE layer to the linear regression layer, we can infer that the classification outputs will be discretized as are the TauSTE outputs. As a result, we could transform the linear regression layer during the SoPa++ to RE proxy simplification phase into a decision tree; which could make the RE proxy model even more transparent given a small enough decision tree. This process could however be difficult for larger models with more TauSTE neurons; therefore this process could be more viable for smaller models.

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 only path scores. In this way, SoPa++ with finite-state transducers could even be used for sequence-to-sequence tasks; making it viable to other applications in NLP such as Neural Machine Translation (NMT). Finally, it would be interesting to see how SoPa++ performs in tasks given longer sequence lengths since the FMTOD data set generally contains short input utterances.

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.