ENABLING CODE-LESS MAINTENANCE AND MACHINE LEARNING MODEL PERFORMANCE EVALUATION FOR NON-MACHINE LEARNING EXPERTS

Hameed M.S.

(IT19064932)

B.Sc. (Hons) Degree in Information Technology specializing in Software Engineering

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology
Sri Lanka

September 2022

ENABLING CODE-LESS MAINTENANCE AND MACHINE LEARNING MODEL PERFORMANCE EVALUATION FOR NON-MACHINE LEARNING EXPERTS

Hameed M.S.

(IT19064932)

The dissertation was submitted in partial fulfillment of the requirements for the B.Sc. Special Honors degree in Information Technology

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology
Sri Lanka

September 2022

DECLARATION, COPYRIGHT STATEMENT AND THE STATEMENT OF THE SUPERVISORS

I declare that this is my own work, and this dissertation does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text. Also, I hereby grant to Sri Lanka Institute of Information Technology the non-exclusive right to reproduce and redistribute my dissertation in whole or part in future works (such as articles or books).

I declare that this is my own work, and this dissertation does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text. Also, I hereby grant to Sri Lanka Institute of Information Technology the non-exclusive right to reproduce and redistribute my dissertation in whole or part in future works (such as articles or books).

Name	Student ID	Signature
Hameed M.S.	IT19064932	

The above candidate is carrying out research for the undergraduate dissertation under my supervision.

Name of the supervisor: Dr. Lakmini Abeywardhana
Signature of the supervisor: Date: 09/09/2022

Name of the co-supervisor: Ms. Dinuka Wijendra
Signature of the co-supervisor: Date: 09/09/2022

ABSTRACT

Configuring and training conversational AI machine learning models is no easy task, since it requires having to be familiarized with the framework being used to develop and Conversational AI, and also having to configure the backend files and execute Command Line Interface commands. Also analyzing and improving conversational AI dialogues require retraining the models for successful implementation and maintenance of the conversational AI for better customer satisfaction, this is true even more so for low resource languages. Yet, analyzing a model to identify overfitting or underfitting requires expertise, without this, it is not possible to identify the most suitable epoch to stop training at, to get the best model. This would require having to contact the conversational AI solution provider each time there is a need to improve the model, which can be a tedious task. The solution is an efficient and code-less approach to improving training data of the conversational AI assistant by eliminating the need to interact with the backend and efficient model testing and evaluation techniques implemented with the zero-coding approach to allow non-machine-learning experts to easily improve the conversational AI assistant models by interacting with the User Interface which helps identify how the model is performing, thus helping the user choose the model that performs the best.

Keywords: Model Evaluation, Conversational AI, Codeless

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to our research team supervisor, Dr. Lakmini Abeywardhana, and co-supervisor, Ms. Dinuka Wijendra, for their support, guidance, and feedback. This endeavour would not have been possible without their knowledge and expertise. I also appreciate the Sri Lanka Institute of Information Technology for hosting an extensive collection of publicly accessible documents and student and course guides on their official websites, which we used to design the conversational AI dataset utilized during this research.

I am also grateful to the SLIIT research project team (CDAP) and lecturers who conducted invaluable sessions for the research project (RP) module to assist us in devising a well-prepared research timeline, document templates, and guidelines. In addition, I would also like to thank the research progress evaluation panel for their invaluable feedback given throughout the research duration.

Finally, I am grateful to everyone whom I might not have mentioned by name who assisted me and my research team in conducting the research at any stage.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Description
AI	Artificial Intelligence
API	Application Programming Interface
CLI	Command Line Interface
FAQ	Frequently Asked Questions
GUI	Graphical User Interface
IDE	Integrated Development Environments
I/O	Input/ Output
LCE	Learning Curve Explainer
ML	Machine Learning
NLG	Natural Language Generation
NLU	Natural Language Understanding
PDF	Portable Document Format
PyPI	Python Package Index
RASAC	Rasa-Codeless
SLIIT	Sri Lanka Institute of Information Technology
UI	User Interface
UUID	Universally Unique Identifier
WSGI	Web Server Gateway Interface
YAML	Yet Another Markup Language

1. INTRODUCTION

1.1 Background & Literature survey

The idea of chatbots was first popularized in the 1950s when the Turing test was proposed by Alan Turing. The Turing test was also called the imitation game. The idea behind the Turing test was to test a machine's ability to exhibit intelligent behaviour indistinguishable or equivalent to that of a human [1]. Eliza was the first known chatbot to be developed, and it was developed completely from scratch in 1966. Eliza used a template-based response mechanism along with simple pattern matching [2]. to return the user utterances in a question form, acting as a psychotherapist [3].

Over the years, chatbots have evolved and become Conversational Artificial Intelligence (AI), more commonly referred to as Conversational AI. This is due to chatbots adopting AI to perform Natural Language Understanding (NLU) and Natural Language Generation (NLG) instead of following pattern-matching or rule-based techniques [4]. Also, conversational AIs have recently gained traction, which is proven by Figure 1.1.1 taken from Scopus [5]. , according to which there is a major increase in publications related to conversational AIs.

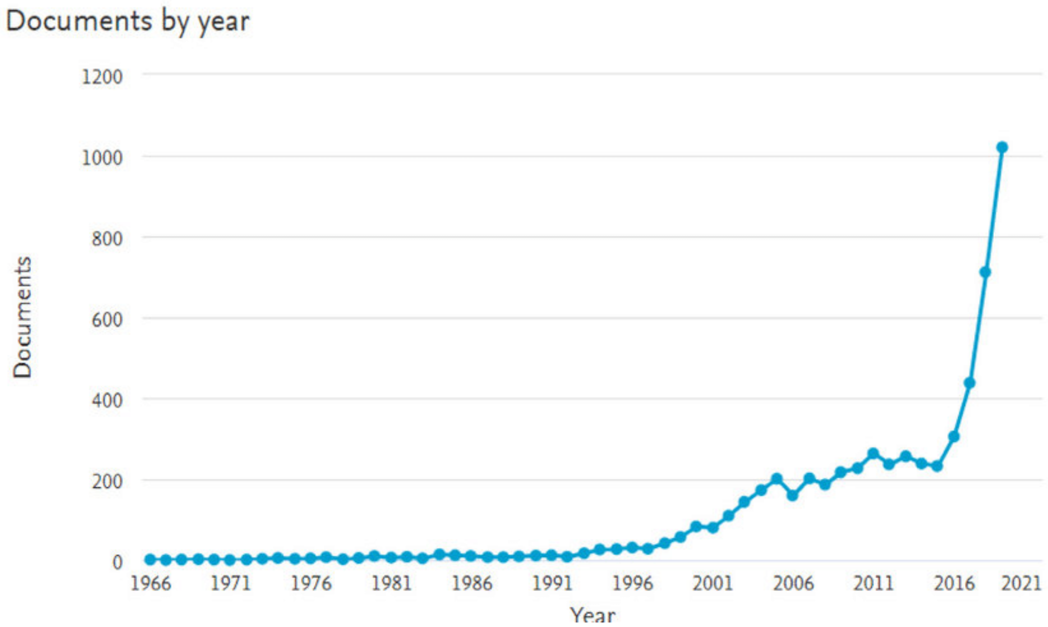


Figure 1.1.1. Search results for *conversation agent* or *chatbot* as a keyword from the year 1966 to 2019 in Scopus

Initially, since conversational AIs were developed from the ground up, it required developers with years of experience and expertise. Later on, local server-based frameworks like Rasa open-source [6]. , with a CLI were created, which provided developers the ability to create conversational AIs without having to build them from scratch. Using a framework like Rasa, they can create conversational AIs by configuring the required Yet Another Markup Language (YAML) files and then executing commands in the Command Line Interface (CLI) to train the Machine Learning (ML) model required for the Conversational AI. Even the approach of using a framework like Rasa still requires some level of expertise since it requires configuring the YAML files, which, if not configured correctly, could cause runtime errors when trying to train the model using the CLI commands. The user should also know the specific CLI commands that are required. Also, cloud-based frameworks such as Google Dialogflow [7]. , Amazon Lex [8]. , and Microsoft LUIS [9]. , with a Graphical User Interface (GUI) as the development console, were introduced. Even though the GUI-based frameworks simplified the process of creating a conversational AI, its functionalities are still limited since they do not provide the ability to fully configure the ML pipeline and policy components required to train a ML model.

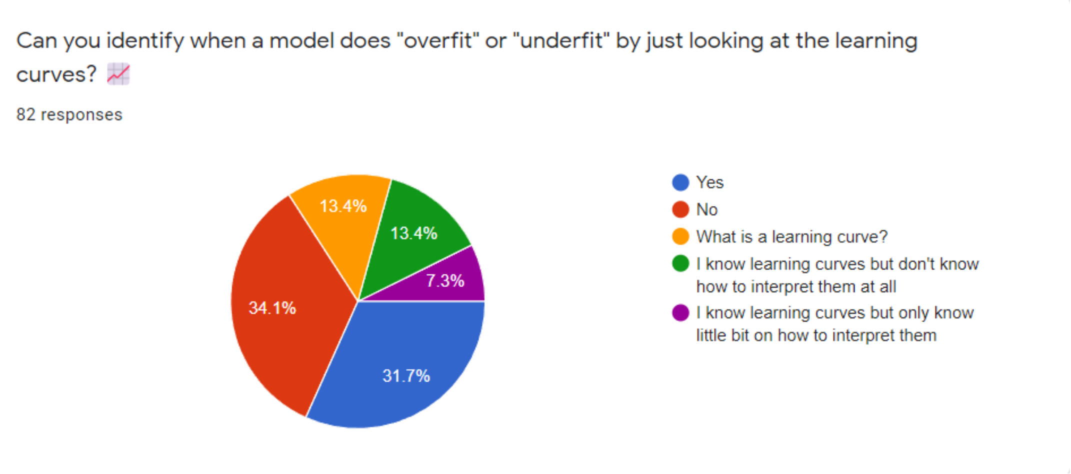
Conversational AIs use ML models that are trained on human conversation data which provides them with the capability to handle complex queries that are handed to them and has the ability to have conversations that are much more natural [10]. Training these models requires expertise. Generally, it requires having to work with the backend, knowing the CLI commands required to split the data into training and testing and then carrying out the model training. Just training alone does not suffice, it is also vital that the right model that performs the best is chosen, and this requires experimenting with the hyperparameters and then looking for any overfittings or underfittings in the learning curve which also require expertise [11].

One of the major issues in supervised ML is that models do not generalize from seen data to unseen data, which is called overfitting [12]. Due to overfitting, the model performs well on training data but performs poorly on testing data. This is due to the over-fitted model having difficulty in coping with the pieces of information in the testing data set, which may be different from the training data set. Also, over-fitted models are inclined to memorize all the data, including the unavoidable noise in the training data set, rather than learning the discipline behind the data set [13].

Generally, the cause for overfitting is categorized into three kinds [13]. The first is, noise learning on the training data set, this occurs either when the training data set is too small, has fewer representative data or else has too many noises. This creates a higher chance of the noises being learnt, which then acts as a basis for predictions [14]. The second category is hypothesis complexity, which refers to the trade-off in complexity, which is a trade-off between variance and bias. It refers to the equilibrium between accuracy and consistency. When the algorithm has too many inputs, the model becomes accurate on average with lower consistency [14]. This means the model can be drastically different on different data sets. The third category is multiple comparison procedures that are ubiquitous in induction algorithms, and in other AI algorithms [15]. During these processes, multiple items are compared based on the scores from an evaluation function and the item with the maximum score is selected. Nevertheless, this process is likely to choose items that will not improve, or even reduce the classification accuracy.

Underfitting is another issue that is faced when training a model, it occurs when the model is not able to accurately capture the relationship between the input and output variables, which produces a high error rate on both the training data set and the testing data set [16]. This occurs when a model is too simple, which can be due to the model requiring more training time, less regularization, or more input features [16]. Similar to overfitting, when underfitting occurs, a model is unable to establish the dominant trend within the data, which results in training errors and the model performs poorly. Low variance and high bias are good indicators of underfitting. Due to this behaviour being visible while using the training data set, it is easier to identify underfitted models than overfitted models.

Most users who are non-ML experts are unable to identify when a model either overfits or underfits, by looking at the learning curve. As shown in Figure 1.1.2, which depicts the data retrieved by the survey conducted, it is evident that the majority are



Can you identify when a model does "overfit" or "underfit" by just looking at the learning curves? 📈

82 responses

Figure 1.1.2. Summary of responses for whether people can identify whether a model is overfitting or underfitting by looking at the learning curve

unable to identify when a model overfits or underfits by looking at the learning curve. Most of these users are also unable to use methods like callbacks, early stopping and checkpointing to help stop the model training at the right epoch which is not too high to cause overfitting or else too low to cause underfitting.

1.2 Research Gap

As stated earlier, both the local server conversational AI development framework Rasa open-source and the cloud-based conversational AI development frameworks such as Google Dialogflow, Amazon Lex, and Microsoft LUIS have their advantages, as well as limitations. The advantages are that Rasa open-source has the ability to fully configure ML pipeline and policy components to train ML models for conversational AIs, and the cloud-based frameworks provide a GUI to build the conversational AI without having to work with the backend. However, Rasa open-source requires working with the backend to configure the YAML files to configure the ML pipeline and policy components, as well as executing CLI commands to train the model, both of which require significant expertise. When it comes to cloud-based frameworks', the ability to configure the ML pipeline and policy components is limited, so much so that users do not get to decide on what pipeline and policy components to use, instead they are required to train the model with whatever the configurations the cloud-platform provides.

The proposed solution to this was to combine the advantages of both the frameworks, which was Rasa open-sources' ability to configure Conversational AI ML pipeline and policy components to train the ML model, integrated into the GUI-based development console of cloud-based frameworks.

Analyzing a model for overfitting or underfitting requires expertise due to the steps that generally need to be followed [17]. , which is, splitting the data set into two, as training data and testing data, or else using a cross-validation scheme, where the data is repeatedly split into training and testing subsets [18]. The model is then trained and a function such as the learning_curve() function from scikit-learn can be used to plot the learning curve, which will depict two curves, one for the training data and the other for the testing data, which will be plotted showing the validation loss for each epoch [19]. For a non-ML expert, these curves would make no sense, since it requires a level of expertise to look at the curve and deduce whether the model is overfitting or underfitting, and what the correct number of epochs would be to stop training at.

Training a model with too many epochs may lead to overfitting, while too few can lead to underfitting. Methods like early stopping can be used to overcome this, where a large number of epochs can be given, and the model will stop training when the model's performance stops improving against a holdout testing dataset. Overfitting can also be overcome using methods like model checkpoint callbacks. These can be done using an Application Programming Interface (API) like Keras [20]. , yet they require having to work with the backend and having considerable knowledge about the Keras API. Also generating learning curves requires using a software like TensorFlow in backend to generate the learning curves which is not an easy task. As shown in Figure 1.2.1, which depicts the data retrieved by the survey conducted, it is evident that the majority agree that developers should be able to build AI-based chatbots without having to be ML experts, which requires training models and evaluating them.

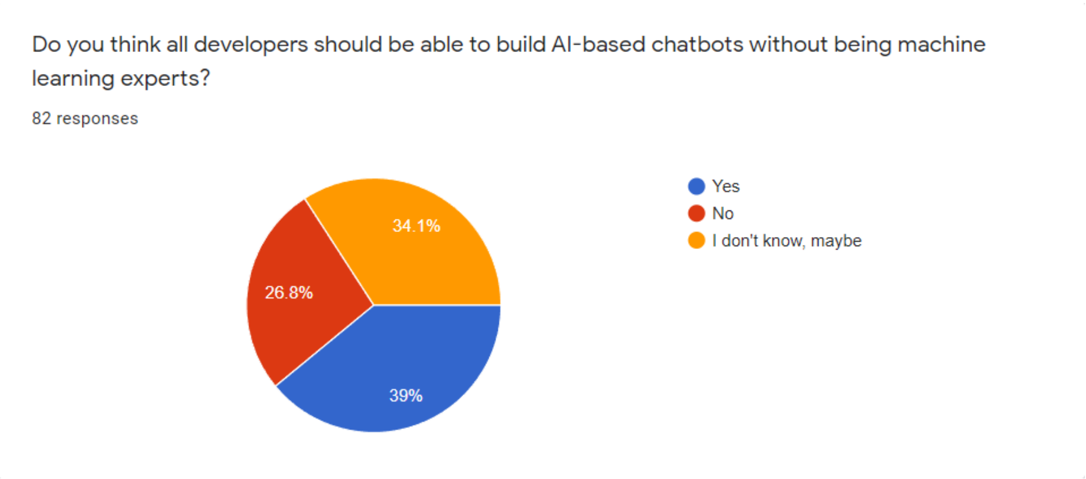


Figure 1.2.1. Summary of responses for whether people think that developers should be able to build conversational AIs without being ML experts

The proposed component was a User Interface (UI) that would enable non-ML experts to improve models used in Conversational AIs by training the models with just the click of a button. Once trained, it will denote how the model is performing by evaluating the learning curve and suggest the best epoch to stop training at, making the whole process of training models a lot easier than the current approach.

Few research papers and tools that exist for model improvement and model evaluation have been summarized in Table 1.2.1 and Table 1.2.2.

Table 1.2.1. Comparison with existing research/tools for model improvements

Name/ Reference	**Tool/ Research**	**Requires knowledge about the backend**	**Requires knowing CLI commands**
Rasa [6].	Tool	Yes	Yes
Rasa X [21].	Tool	Yes	No
This Research component	Tool	No	No

Table 1.2.2. Comparison with existing research/tools for model evaluation

Name/ Reference	**Tool/ Research**	**Suggest the best epoch range to train the model**	**Requires knowledge about the backend to understand the feedback**
Rasa [6].	Tool	Yes (Early stopping) but no visual indication	Yes
This Research component	Tool	Yes	No

2. RESEARCH PROBLEM

At present most businesses have integrated conversational AIs to their websites due to its convenience to customers, which allows the customers to receive the help they need faster, than having to browse the website or contact the business to retrieve the necessary information. According to a survey, 64% of people say they would rather message a business than call it [22]. The main reason people prefer messaging over calling is due to the number of times the calls generally do not go through, which can be frustrating to a customer [23].

The methods used to develop conversational AIs have changed over the years. At present, most developers use frameworks to build Conversational AIs. There are a few frameworks that are widely used at present. One of them is Rasa open-source, which is a local server-based framework that provides developers with the ability to build conversational AIs by configuring YAML files and then executing CLI commands to train the model required for the conversational AI. The other frameworks used are Google Dialogflow, Amazon Lex, and Microsoft LUIS, which are all cloud-based frameworks and come with a GUI, which simplifies the conversational AI model training process by eliminating the need to work with the backend. Both these types of frameworks have their limitations. Rasa open-source requires the developer to know how to configure the YAML file and use the relevant CLI commands to train the model, which requires expertise. On the other hand, even though the cloud-based frameworks eliminate the need to work with the backend, thus reducing the required expertise, they do not provide the ability to fully configure the ML pipeline and policy components required to train a model the way Rasa open-source does. Instead, the developers are expected to train the model with whichever configurations are suggested by the cloud platform. Putting the strengths of both these frameworks together was the suggested solution, which combines Rasa open-sources' ability to fully configure ML pipeline and policy components along with the cloud-based platforms' GUI, which eliminates the need to work with the backend and simplifies the configuring and training tasks.

Once a conversational AI model is trained, it is important to evaluate the model to ensure it will perform well. Model evaluation is generally done in the backend by the developers who build the conversational AI. It is done by generating accuracy and loss curves and then evaluating them to check whether the model is performing well or whether it is overfitting or underfitting. The process of generating validation curves and evaluating them requires expertise. Hence, the developers of an organization cannot be expected to know how it is to be done. Since there is a need to provide the developers of organizations with a GUI to train models, it is also important to include a feature to generate and evaluate the models using validation curves without having to know the technical know-how and suggest what the best epoch is to train a model.

Integrating a conversational AI into a business's website is not a one-time thing. To cater to customers without issues, the conversational AI must be constantly maintained and improved [24]. It can become a tedious task having to constantly contact the conversational AI service provider every time there is a need to improve the conversational AI. Hence, the proposed solution was to build a GUI that makes it possible to maintain, monitor, and improve the conversational AI without requiring any ML expertise.

3. RESEARCH OBJECTIVES

3.1 Main Objectives

The main objective of this individual research component was to develop RASAC, short for Rasa-Codeless, an efficient and code-less approach to provide the user with the ability to configure the ML pipeline and policy configurations required to train the model, and also implement an efficient model testing and evaluating technique with the zero-coding approach to allow non-ML experts to improve the conversational AI assistant models, all while eliminating the need to interact with the backend configuration files and the CLI, and requiring them to only work with the GUI.

3.2 Specific Objectives

To reach the main objective, the specific objectives that need to be achieved are as follows:

1. Collect Sinhala-English code-switched textual data required to train a Rasa conversational AI to develop, integrate, and test the RASAC implementations.

2. Develop a novel GUI with the ability to configure ML pipeline and policy configurations using just the GUI and train the machine learning model with just the click of a button while hiding the backend complexity from the user.

3. Develop an algorithm to suggest the best epoch of a trained model by evaluating the loss curve of the model.

4. Develop a GUI to view all the trained machine learning models along with their train and test accuracies and with the ability to delete and download those models and view the learning curves of these models along with the algorithm to suggest the best epoch integrated into it.

5. Develop the RASAC development console with the above-mentioned GUIs integrated.

6. Build and release RASAC as a modular open-source Python package that works with any Rasa chatbot model.

7. Conduct test-driven development on RASAC by designing test cases and conducting unit testing, integration testing, system testing, acceptance testing, and security testing.

8. Integrate the RASAC core and server packages with Kolloqe, the chatbot development framework introduced as the overall research component and test the integrated functionality.

9. Build and host an official documentation website for the RASAC Python package to be published and integrate the content of the documentation website with the Kolloqe documentation website, which is the documentation website of the integrated solution.

4. METHODOLOGY

4.1 Preparation of Datasets

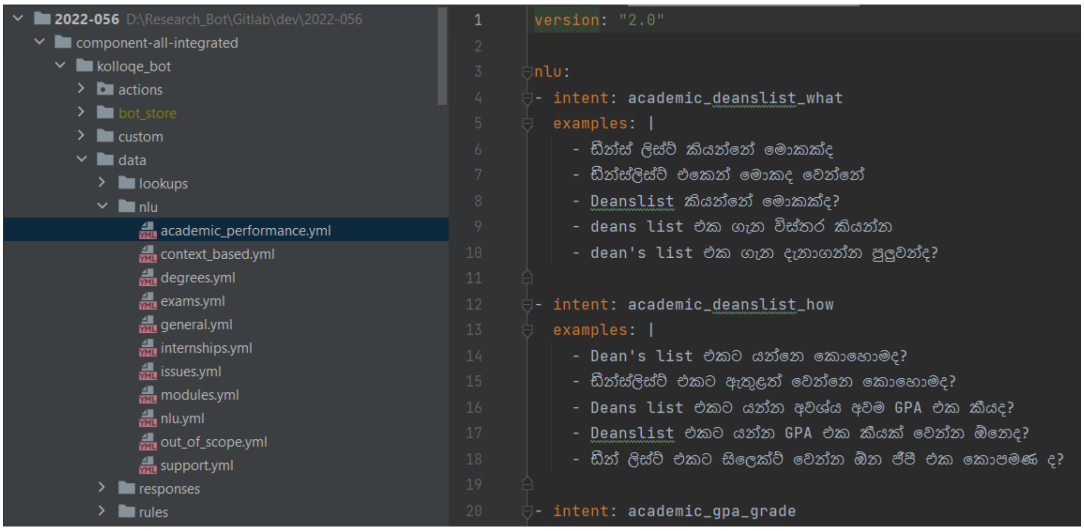
Two datasets were prepared for the overall research, both of which were domain-specific and comprised of Sinhala-English code-switched text data. Data augmentation techniques were utilized to overcome the low-resource nature of the Sinhala data gathered. All four members of the research group used the two gathered datasets to generate code-switched, Sinhala-English data in every possible writing pattern to avoid biases in the writing pattern of the dataset. Finally, any duplicate entries in the data were removed to maintain the quality of the dataset. The difference between the two datasets will be further explained in subsections 4.1.1 and 4.1.2.

4.1.1 General dataset for machine learning model training

The general dataset that was used to train the machine learning model consists of articles related to Sri Lanka Institute of Information Technology (SLIIT), which were scraped from publicly available news articles [25]. -[27]. and the websites related to SLIIT [28]. -[32]. . Also, the dataset contains publicly available documents such as Microsoft Word documents and Portable Document Format (PDF), taken from the same sources since they are both official and public. All the information that is in English was converted to Sinhala using Google translate, which was then rewritten in informal, spoken Sinhalese, which also contains English words where necessary, by the four members of the research team. This is a data augmentation approach to increase the volume of the available data to overcome the low-resource nature of the gathered Sinhala text. Using informal, spoken Sinhalese to train the model helps the Conversational AI perform better since most users use informal, spoken Sinhalese when messaging. After the data augmentation and cleaning of the dataset, the dataset contained 720 paragraphs and is publicly available as a public GitHub repository, where interested parties can clone and use it for research and other academic-related work [33].

4.1.2 Domain specific dataset for conversational AI training

The domain-specific dataset contains Frequently Asked Questions (FAQ) and answers written in Sinhala-English code-switched textual data, that was used to train the conversational AI's intent classification task according to the guidelines provided by Rasa [34]. As the previous dataset, data augmentation techniques were used to overcome the low-resource nature of the gathered data. The data was written in different permutations and spellings to cover all the possible ways of asking the question. And the corresponding answers to these questions were written as well. The data contains around seventy-eight intents (78 classes) and one thousand and seven hundred examples (1700 data points), initially each class contained a minimum of ten questions. The training dataset was later refined to create a new dataset with seventy-one intents (71 classes) and six hundred and thirty-five examples (635 data points). The number of questions per intent can be changed to increase the overall performance of the conversational AI at any given time. The training dataset resides in the same public GitHub repository mentioned in section 4.1.1, where interested parties can clone and utilize it for domain-specific chatbot training, other related academic work, or for future research.



```
version: "2.0"

nlu:
- intent: academic_deanslist_what
  examples: |
    - ඩීන්ස් ලිස්ට් කියන්නේ මොකක්ද
    - ඩීන්ස්ලිස්ට් එකෙන් මොකද වෙන්නේ
    - Deanslist කියන්නේ මොකක්ද?
    - deans list එක ගැන විස්තර කියන්න
    - dean's list එක ගැන දැනාගන්න පුලුවන්ද?

- intent: academic_deanslist_how
  examples: |
    - Dean's list එකට යන්නෙ කොහොමද?
    - ඩීන්ස්ලිස්ට් එකට ඇතුළත් වෙන්නෙ කොහොමද?
    - Deans list එකට යන්න අවශ්‍ය අවම GPA එක කීයද?
    - Deanslist එකට යන්න GPA එක කීයක් වෙන්න ඕනෙද?
    - ඩීන් ලිස්ට් එකට සිලෙක්ට් වෙන්න ඕන ජීපී එක කොපමණ ද?

- intent: academic_gpa_grade
```

Figure 4.1.2.1. A part of the prepared dataset

4.2 Functional and Non-Functional Requirements

The requirements-gathering phase mainly involved working with the Rasa backend to get familiarized with the process of configuring pipeline and policy components and then invoking the required CLI commands to train the Rasa model and working with TensorFlow to generate the learning curve of trained models and studying existing research on how to identify how a model is performing and when a model is either overfitting or underfitting based on the learning curve of a model.

Based on the analyzed data, the following functional and non-functional requirements were derived for the RASAC component.

4.2.1 Functional Requirements

The functional requirements of RASAC are as follows,

1. Users should be able configure machine learning pipeline and policy components and train a model using just the UI without having any knowledge on dependencies required by each component.

2. Users should be able to view all the trained models along with their train and test accuracies.

3. The algorithm should be able to suggest the best epoch of a given model when the user provides the patience interval value

4.2.2 Non-Functional Requirements

The non-functional requirements of RASAC are as follows,

1. The UI used to configure the ML pipeline and policy components need to be user friendly and minimalistic, reducing the possibility of errors that can happen by providing users with dropdowns and radio buttons to choose components and their relevant values instead of typing values into text fields.

2. Model evaluations done should be reliable.

3. Model evaluations should be consistent.

4. Generating model evaluations should not hinder the model training time.

5. When suggesting the best epoch, the UI must be responsive to change the suggested epoch without a lag each time the user changes the patience interval value.

All the functional and non-functional requirements stated above were successfully addressed during the development of RASAC, and the upcoming sections will further elaborate on how these requirements were fulfilled.

4.3 Individual Component Architecture

The developed RASAC component comprises several sub-components, as depicted in **Error! Reference source not found.**, the high-level architectural diagram of RASAC. The RASAC Python package has four logical sub-packages: Server, Shared, Utils, and Core. The Server package is responsible for serving a production ready RASAC developer console GUI and contains the frontend and server build files. The frontend is completely React-based. The Shared package contains the RASAC-specific custom exception classes and constants that the other packages require. The Utils package contains I/O (Input/ Output) and other widely utilized utility functions. And the Core package contains the core logic of the RASAC component.

The developers can configure the ML pipeline and policy components and train the model using the GUI which is denoted by *ML Pipeline Configurations & Training* in the high-level architectural diagram of RASAC. The *Curve Insights LCE (Best Epoch)* section denotes where the users can view the trained models and their learning curves, which also includes the suggestion of the best epoch suggested by the developed algorithm. **Error! Reference source not found.** depicts how RASAC fits into the overall system architecture after all the research components are integrated.

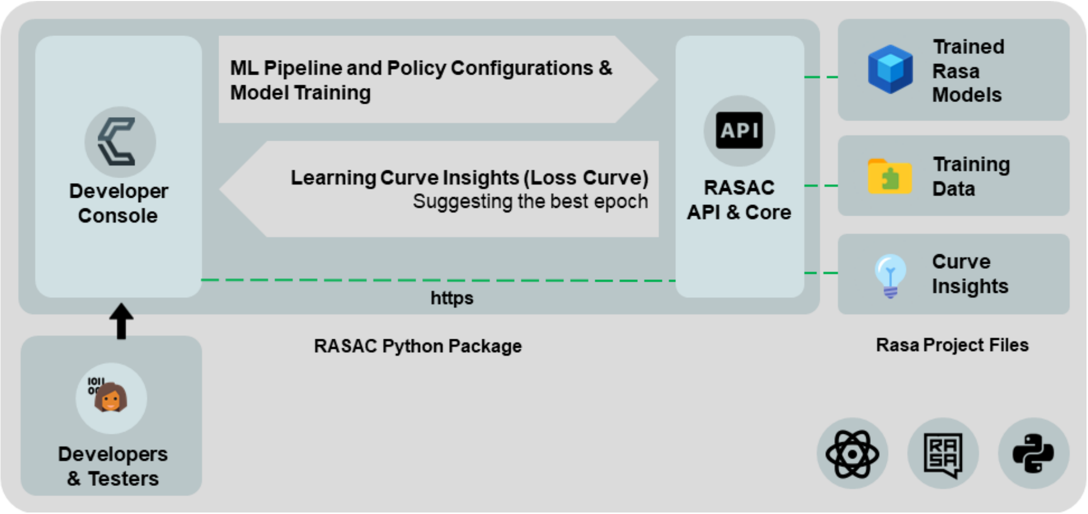


Figure 4.3.2. High-level architectural diagram of RASAC

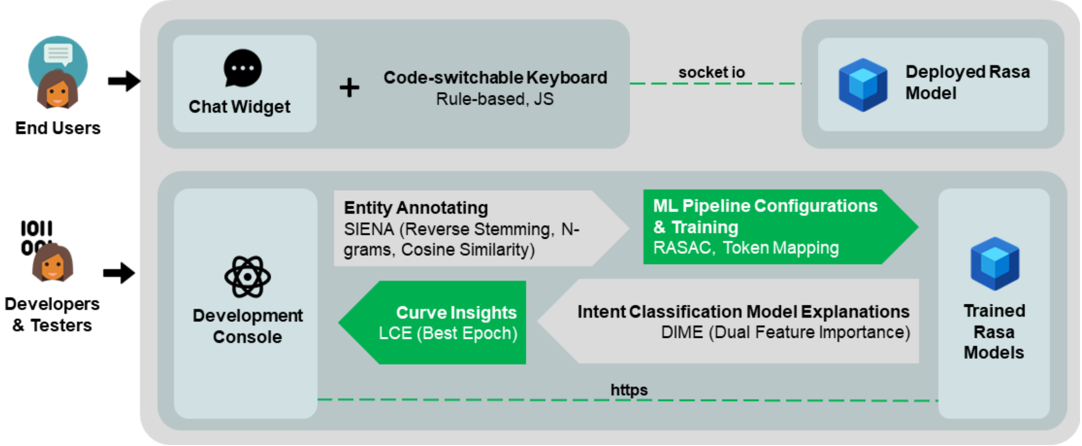


Figure 4.3.1. High-level architectural diagram of Kolloqe, the overall research, and the parts highlighted in green indicate where RASAC was utilized

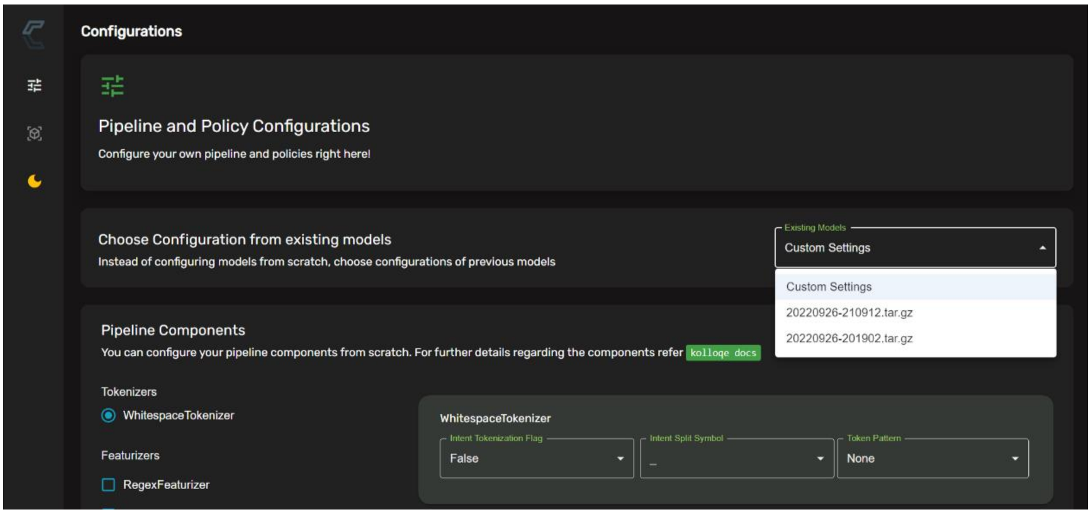
4.4 Codeless Model Improvement

Users have the option to configure the pipeline from scratch, which is denoted as *Custom Settings* or else choose a configuration from an existing model from the dropdown menu as seen in Figure 4.4.1. If the configuration of an existing model is chosen, users have to ability to change the values of the chosen components as well as add or remove other components before training the model.

Figure 4.4.2 depicts what the UI looks like when the *Custom Settings* option is chosen from the dropdown menu, and Figure 4.4.3 depicts what the UI looks like when the configuration of an existing model is chosen. Users can also get familiarized with the available pipeline and policy components by clicking on *Kolloqe docs* which can be seen right below the headings *Pipeline Components* and *Policy Components*, which will redirect them to the documentation website as seen in Figure 4.4.4 which explains what each component does and how to configure each variable.

The pipeline and policies can be configured using the UI as seen in the figures by ticking the checkboxes that are required that come under the categories Tokenizers, Featurizers, Classifiers, and Extractors for pipeline components and under Policies for policy components, and then configuring the variables in each component by choosing values for them from the provided dropdown menus, checkboxes, and the few text boxes for numeric values, and finally training the model by clicking the *train model*. Once the button is clicked, the configurations from the UI are passed to the backend where the YAML configuration file is overwritten using the provided configurations as seen in Figure 4.4.5, and then all the necessary CLI commands are invoked using bash command calls to the backend which can be seen in Figure 4.4.6.

Users can abort once a model has been put to train, using the *Abort* button which can be seen in Figure 4.4.7. When a model is first put to train, a Universally Unique Identifier (UUID) v4 is generated in the frontend to uniquely identify the frontend user that the request is coming from, this is then sent to the backend where all the necessary CLI commands to train the model are invoked using bash commands, this is then put into a training queue along with the UUID v4, and the process id. And when a user aborts a training session, that user’s UUID v4 is sent to the backend and checked against the training queue to check whether an entry exists with the said UUID v4, if so the necessary CLI commands to abort the training is invoked using bash command calls for the process id, which in turn aborts the training.



Configurations

Pipeline and Policy Configurations

Configure your own pipeline and policies right here!

Choose Configuration from existing models

Instead of configuring models from scratch, choose configurations of previous models

Existing Models: Custom Settings

- Custom Settings
- 20220926-210912.tar.gz
- 20220926-201902.tar.gz

Pipeline Components

You can configure your pipeline components from scratch. For further details regarding the components refer `kolloqe docs`

Tokenizers

● WhitespaceTokenizer

Featurizers

□ RegexFeaturizer

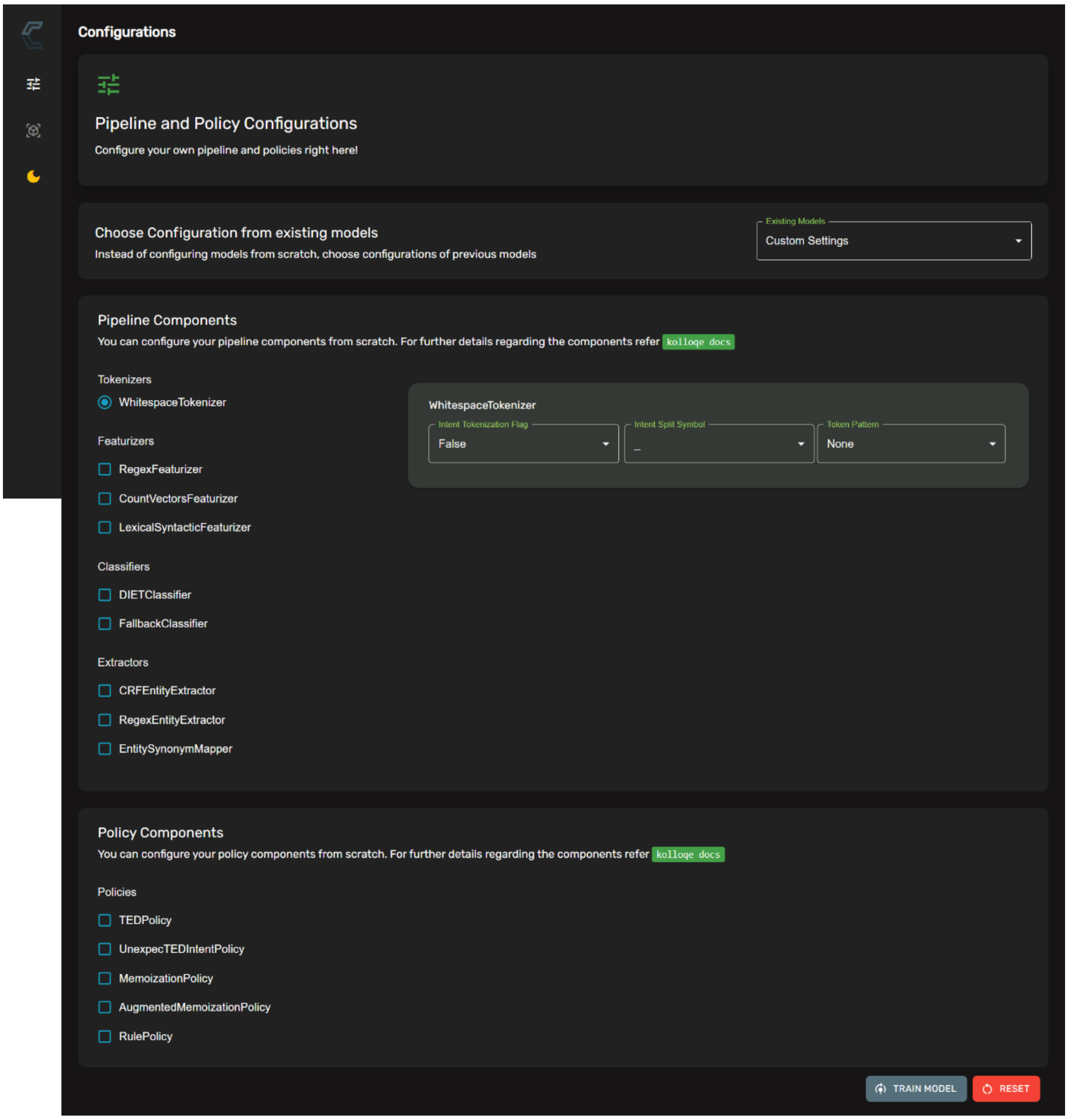
WhitespaceTokenizer

Intent Tokenization Flag: False

Intent Split Symbol: _

Token Pattern: None

Figure 4.4.1. Option to choose *Custom Settings* to configure the pipeline and policy components from scratch or else choose the configurations of an existing model



Configurations

Pipeline and Policy Configurations

Configure your own pipeline and policies right here!

Choose Configuration from existing models

Instead of configuring models from scratch, choose configurations of previous models

Existing Models: Custom Settings

Pipeline Components

You can configure your pipeline components from scratch. For further details regarding the components refer `kolloqe docs`

Tokenizers

● WhitespaceTokenizer

Featurizers

□ RegexFeaturizer

□ CountVectorsFeaturizer

□ LexicalSyntacticFeaturizer

Classifiers

□ DIETClassifier

□ FallbackClassifier

Extractors

□ CRFEntityExtractor

□ RegexEntityExtractor

□ EntitySynonymMapper

WhitespaceTokenizer

Intent Tokenization Flag: False

Intent Split Symbol: _

Token Pattern: None

Policy Components

You can configure your policy components from scratch. For further details regarding the components refer `kolloqe docs`

Policies

□ TEDPolicy

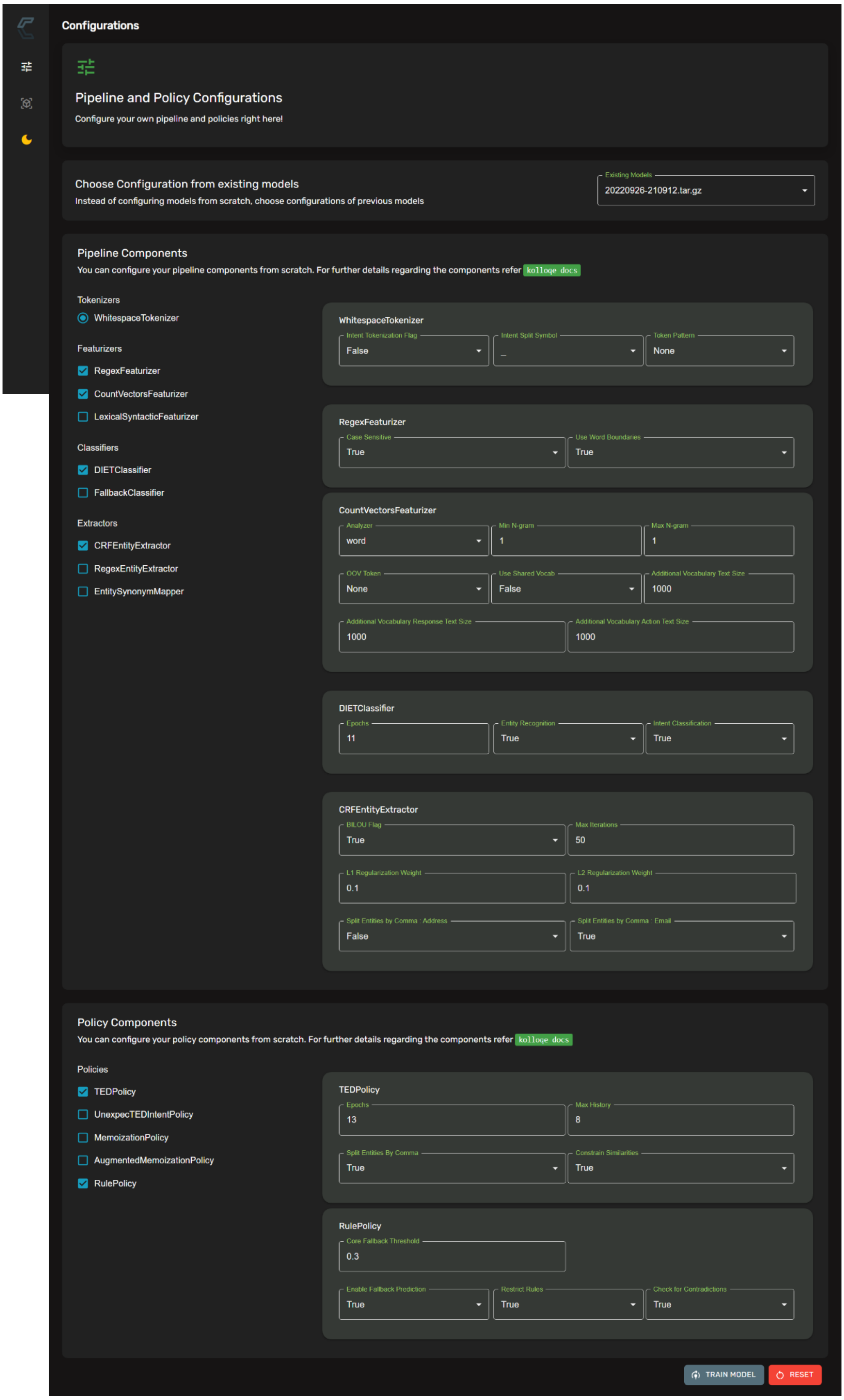
□ UnexpecTEDIntentPolicy

□ MemoizationPolicy

□ AugmentedMemoizationPolicy

□ RulePolicy

Figure 4.4.2. The UI when the *Custom Settings* option is chosen from the dropdown menu



Configurations

Pipeline and Policy Configurations

Configure your own pipeline and policies right here!

Choose Configuration from existing models

Instead of configuring models from scratch, choose configurations of previous models

Existing Models: 20220926-210912.tar.gz

Pipeline Components

You can configure your pipeline components from scratch. For further details regarding the components refer `kolloqe docs`

Tokenizers

● WhitespaceTokenizer

Featurizers

☑ RegexFeaturizer
☑ CountVectorsFeaturizer
☐ LexicalSyntacticFeaturizer

Classifiers

☑ DIETClassifier
☐ FallbackClassifier

Extractors

☑ CRFEntityExtractor
☐ RegexEntityExtractor
☐ EntitySynonymMapper

WhitespaceTokenizer

Intent Tokenization Flag: False
Intent Split Symbol: _
Token Pattern: None

RegexFeaturizer

Case Sensitive: True
Use Word Boundaries: True

CountVectorsFeaturizer

Analyzer: word
Min N-gram: 1
Max N-gram: 1
OOV Token: None
Use Shared Vocab: False
Additional Vocabulary Text Size: 1000
Additional Vocabulary Response Text Size: 1000
Additional Vocabulary Action Text Size: 1000

DIETClassifier

Epochs: 11
Entity Recognition: True
Intent Classification: True

CRFEntityExtractor

BILOU Flag: True
Max Iterations: 50
L1 Regularization Weight: 0.1
L2 Regularization Weight: 0.1
Split Entities by Comma : Address: False
Split Entities by Comma : Email: True

Policy Components

You can configure your policy components from scratch. For further details regarding the components refer `kolloqe docs`

Policies

☑ TEDPolicy
☐ UnexpecTEDIntentPolicy
☐ MemoizationPolicy
☐ AugmentedMemoizationPolicy
☑ RulePolicy

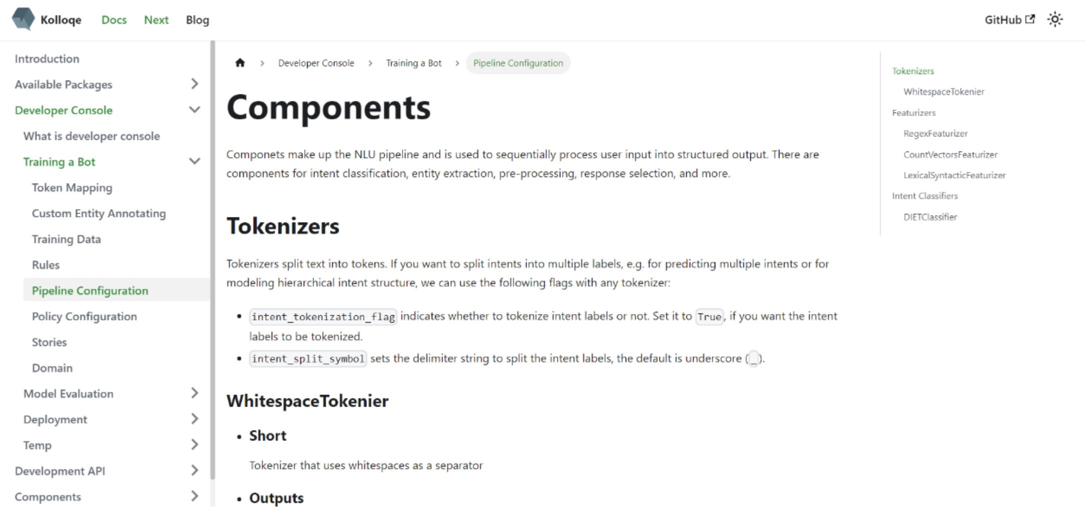
TEDPolicy

Epochs: 13
Max History: 8
Split Entities By Comma: True
Constrain Similarities: True

RulePolicy

Core Fallback Threshold: 0.3
Enable Fallback Prediction: True
Restrict Rules: True
Check for Contradictions: True

Figure 4.4.3. The UI when the configuration of an existing model is chosen from the dropdown menu



Kolloqe Docs Next Blog GitHub

- Introduction
- Available Packages
- Developer Console
  - What is developer console
  - Training a Bot
    - Token Mapping
    - Custom Entity Annotating
    - Training Data
    - Rules
    - Pipeline Configuration
    - Policy Configuration
    - Stories
    - Domain
  - Model Evaluation
  - Deployment
  - Temp
- Development API
- Components

Developer Console > Training a Bot > Pipeline Configuration

Components

Componets make up the NLU pipeline and is used to sequentially process user input into structured output. There are components for intent classification, entity extraction, pre-processing, response selection, and more.

Tokenizers

Tokenizers split text into tokens. If you want to split intents into multiple labels, e.g. for predicting multiple intents or for modeling hierarchical intent structure, we can use the following flags with any tokenizer:

- `intent_tokenization_flag` indicates whether to tokenize intent labels or not. Set it to `True`, if you want the intent labels to be tokenized.
- `intent_split_symbol` sets the delimiter string to split the intent labels, the default is underscore (`_`).

WhitespaceTokenier

- **Short**

  Tokenizer that uses whitespaces as a separator

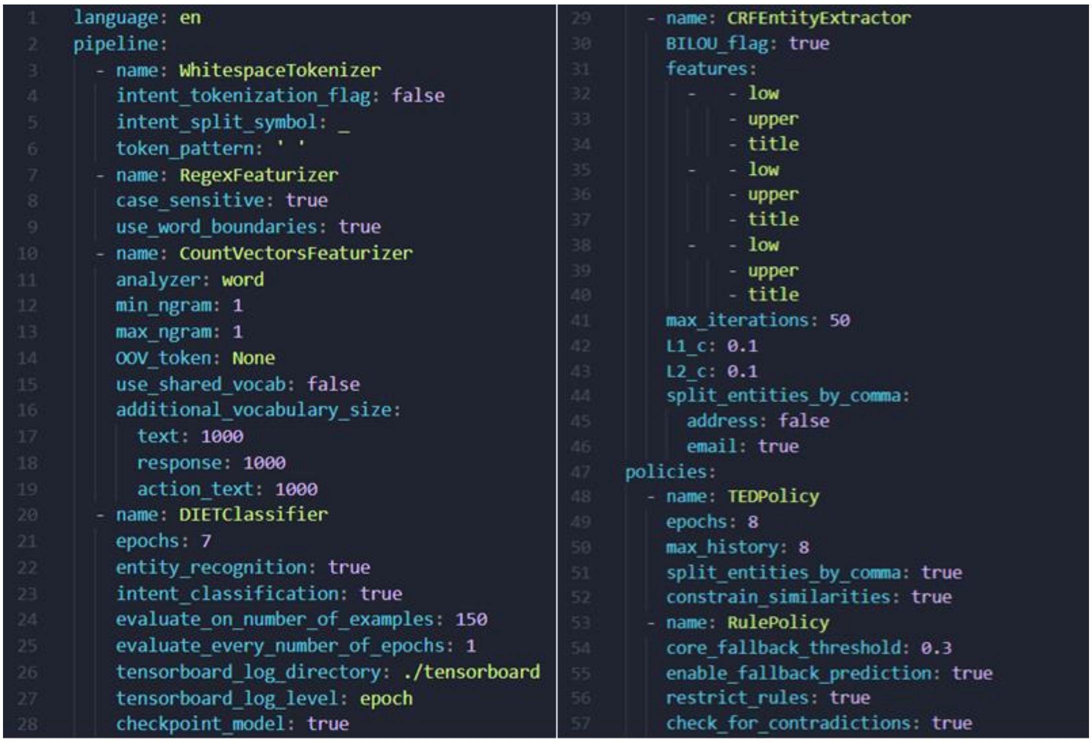
- **Outputs**

Tokenizers
- WhitespaceTokenier

Featurizers
- RegexFeaturizer
- CountVectorsFeaturizer
- LexicalSyntacticFeaturizer

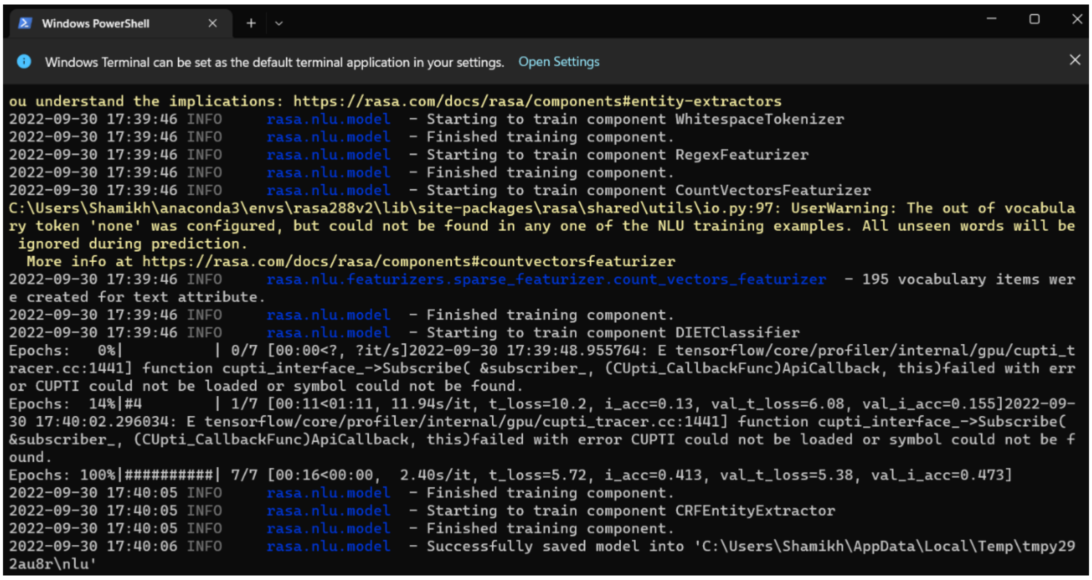
Intent Classifiers
- DIETClassifier

Figure 4.4.4. The pipeline configuration section of the Kolloqe documentation



```yaml
language: en
pipeline:
  - name: WhitespaceTokenizer
    intent_tokenization_flag: false
    intent_split_symbol: _
    token_pattern: ' '
  - name: RegexFeaturizer
    case_sensitive: true
    use_word_boundaries: true
  - name: CountVectorsFeaturizer
    analyzer: word
    min_ngram: 1
    max_ngram: 1
    OOV_token: None
    use_shared_vocab: false
    additional_vocabulary_size:
      text: 1000
      response: 1000
      action_text: 1000
  - name: DIETClassifier
    epochs: 7
    entity_recognition: true
    intent_classification: true
    evaluate_on_number_of_examples: 150
    evaluate_every_number_of_epochs: 1
    tensorboard_log_directory: ./tensorboard
    tensorboard_log_level: epoch
    checkpoint_model: true
  - name: CRFEntityExtractor
    BILOU_flag: true
    features:
      -   - low
          - upper
          - title
      -   - low
          - upper
          - title
      -   - low
          - upper
          - title
    max_iterations: 50
    L1_c: 0.1
    L2_c: 0.1
    split_entities_by_comma:
      address: false
      email: true
policies:
  - name: TEDPolicy
    epochs: 8
    max_history: 8
    split_entities_by_comma: true
    constrain_similarities: true
  - name: RulePolicy
    core_fallback_threshold: 0.3
    enable_fallback_prediction: true
    restrict_rules: true
    check_for_contradictions: true
```

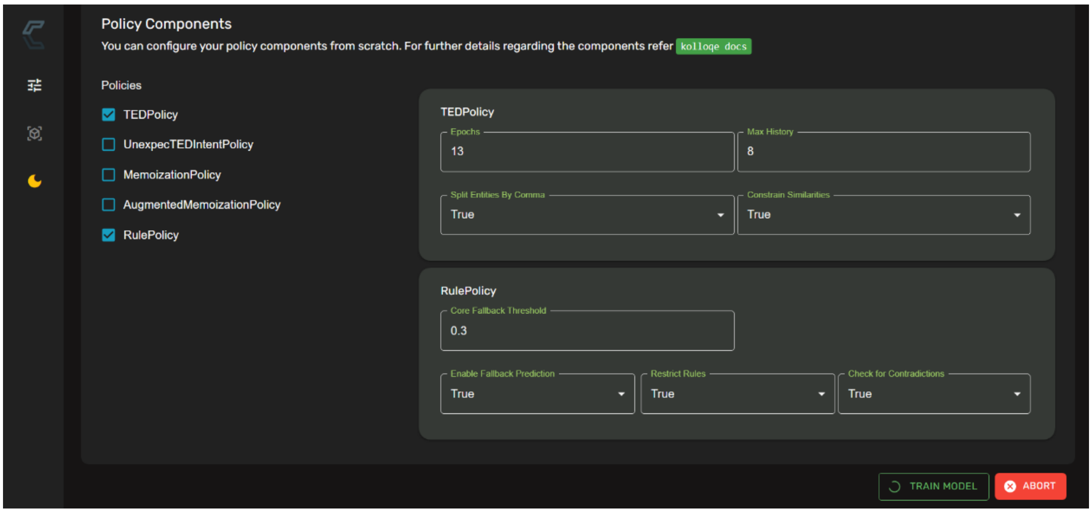
Figure 4.4.5. The backend YAML configuration file used to train the ML model for the conversational AI



Windows PowerShell
Windows Terminal can be set as the default terminal application in your settings.
Open Settings

```
ou understand the implications: https://rasa.com/docs/rasa/components#entity-extractors
2022-09-30 17:39:46 INFO     rasa.nlu.model  - Starting to train component WhitespaceTokenizer
2022-09-30 17:39:46 INFO     rasa.nlu.model  - Finished training component.
2022-09-30 17:39:46 INFO     rasa.nlu.model  - Starting to train component RegexFeaturizer
2022-09-30 17:39:46 INFO     rasa.nlu.model  - Finished training component.
2022-09-30 17:39:46 INFO     rasa.nlu.model  - Starting to train component CountVectorsFeaturizer
C:\Users\Shamikh\anaconda3\envs\rasa288v2\lib\site-packages\rasa\shared\utils\io.py:97: UserWarning: The out of vocabula
ry token 'none' was configured, but could not be found in any one of the NLU training examples. All unseen words will be
 ignored during prediction.
  More info at https://rasa.com/docs/rasa/components#countvectorsfeaturizer
2022-09-30 17:39:46 INFO     rasa.nlu.featurizers.sparse_featurizer.count_vectors_featurizer  - 195 vocabulary items wer
e created for text attribute.
2022-09-30 17:39:46 INFO     rasa.nlu.model  - Finished training component.
2022-09-30 17:39:46 INFO     rasa.nlu.model  - Starting to train component DIETClassifier
Epochs:   0%|          | 0/7 [00:00<?, ?it/s]2022-09-30 17:39:48.955764: E tensorflow/core/profiler/internal/gpu/cupti_t
racer.cc:1441] function cupti_interface_->Subscribe( &subscriber_, (CUpti_CallbackFunc)ApiCallback, this)failed with err
or CUPTI could not be loaded or symbol could not be found.
Epochs:  14%|#4        | 1/7 [00:11<01:11, 11.94s/it, t_loss=10.2, i_acc=0.13, val_t_loss=6.08, val_i_acc=0.155]2022-09-
30 17:40:02.296034: E tensorflow/core/profiler/internal/gpu/cupti_tracer.cc:1441] function cupti_interface_->Subscribe(
&subscriber_, (CUpti_CallbackFunc)ApiCallback, this)failed with error CUPTI could not be loaded or symbol could not be f
ound.
Epochs: 100%|##########| 7/7 [00:16<00:00,  2.40s/it, t_loss=5.72, i_acc=0.413, val_t_loss=5.38, val_i_acc=0.473]
2022-09-30 17:40:05 INFO     rasa.nlu.model  - Finished training component.
2022-09-30 17:40:05 INFO     rasa.nlu.model  - Starting to train component CRFEntityExtractor
2022-09-30 17:40:05 INFO     rasa.nlu.model  - Finished training component.
2022-09-30 17:40:06 INFO     rasa.nlu.model  - Successfully saved model into 'C:\Users\Shamikh\AppData\Local\Temp\tmpy29
2au8r\nlu'
```

Figure 4.4.6. All the necessary CLI commands to train the model invoked using bash command calls to the backend



Policy Components

You can configure your policy components from scratch. For further details regarding the components refer kolloqe docs

Policies

☑ TEDPolicy
□ UnexpecTEDIntentPolicy
□ MemoizationPolicy
□ AugmentedMemoizationPolicy
☑ RulePolicy

TEDPolicy

Epochs: 13
Max History: 8
Split Entities By Comma: True
Constrain Similarities: True

RulePolicy

Core Fallback Threshold: 0.3
Enable Fallback Prediction: True
Restrict Rules: True
Check for Contradictions: True

Figure 4.4.7. The UI when the model training is in progress and the *Abort* button is visible to abort training

4.5 Model Performance Evaluation

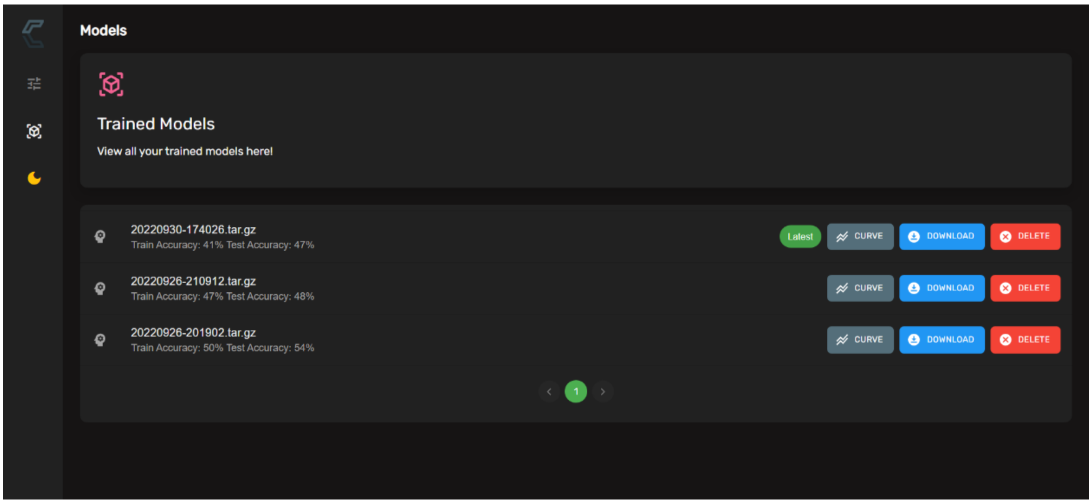
Figure 4.5.1 shows the UI where all the trained models are displayed along with the name of the model and each model's train and test accuracy below the model's name. Users can click the *Curve* button to view both the accuracy curve and loss curve of each model as shown in Figure 4.5.2 and Figure 4.5.3 respectively.

Figure 4.5.4 shows the UI where the loss curve is plotted, along with the best epoch suggested on top based on the patience interval chosen by the user. The moving average and the moving standard deviation are calculated for the train and test curves taking into account the chosen patience interval, which is the number of epochs to calculate the moving average with. Then 2 times the moving standard deviation is used to plot another two curves above and below the train and test curve. The training curve is depicted by the red curve and the red band above it is the positive 2 times moving standard deviation curve, while the red band below it is the negative 2 times standard deviation curve. The same is done for the test curve which is depicted in blue. The best is suggested by taking the lowest test epoch that is within the positive and negative moving standard deviations of the training curve, which are also known as Bollinger Bands. Initially, the first epoch is taken as the lowest epoch and then each corresponding lower epoch is taken as the lowest epoch, the number of epochs checked once an epoch is taken as the lowest depends on the chosen patience interval, which is taken as the number of epochs to check before terminating the algorithm and suggesting the current lowest epoch as the best epoch.

Algorithm 1 CURVEEXPLAINER($trainLoss, testLoss, patienceInterval$)

```
$trainBolinger \leftarrow$ BOLL($trainLoss$, $patienceInterval$, 2)    ▷ Calling the boll function from bollinger_band NPM package
$minLoss \leftarrow testLoss[0]$    ▷ Assuming the minimum loss is in the first epoch
$bestEpoch \leftarrow 1$
for $index \in \{0,\ldots,$ LENGTH($trainLoss$)$\}$ do    ▷ Iterate through epochs
    if $index < patienceInterval - 1$ then
        if $testLoss[index] < minLoss$ then
            $minLoss \leftarrow testLoss[index]$
            $bestEpoch \leftarrow index + 1$
    else
        if $testLoss[index] < trainBollinger[upper][index]$ and $testLoss[index] < trainBollinger[lower][index]$ then
            if $testLoss[index] < minLoss$ then
                $minLoss \leftarrow testLoss[index]$
                $bestEpoch \leftarrow index + 1$
        else
            $toBreak \leftarrow true$
            for $j \in \{\ i + 1,\ldots, j < i + 1 + patienceInterval)\}$ do
                if $testLoss[j] < minLoss$ and
                $testLoss[j] < trainBollinger[upper][j]$ and
                $testLoss[j] < trainBollinger[lower][j]$ do
                    $minLoss \leftarrow testLoss[j]$
                    $bestEpoch \leftarrow j + 1$
                    $toBreak \leftarrow false$
                    $i \leftarrow j$
                else
                    $toBreak \leftarrow true$
            if $toBreak == true$ then
                $break$
    if $testLoss[i] < minLoss$ do
        $minLoss \leftarrow testLoss[index]$
        $bestEpoch \leftarrow index + 1$
    return $bestEpoch$
```

The above pseudocode *curveExplainer* shows the algorithm used to suggest the best epoch of a given model when the train and test loss curve data points are fed to the algorithm along with the patience interval (number of epochs to consider when calculating the moving average) value provided by the user which are denoted by *trainLoss*, *testLoss*, and *patienceInterval* in the algorithm respectively. The moving average and moving standard deviation are then calculated using a Node Package Manager (NPM) package known as *bollinger-bands* [35]. , which outputs the upper, middle and lower bands, where the middle band is the moving average, and the upper band is twice the standard deviation added to the moving average, and the lower band is twice the standard deviation deducted from the moving average. And then the upper and lower bands are plotted around the train loss curve and the algorithm looks for datapoints in the test loss curve which are in-between the upper and lower bands of the training loss curve to make sure the points have not deviated too far away from the train loss curve, and once a point is chosen, the number of epochs that the algorithm will search for to find a lower epoch is determined by the value of the patience interval. If no new point is found the algorithm terminates and returns the current lowest epoch as the best epoch.



Models

Trained Models

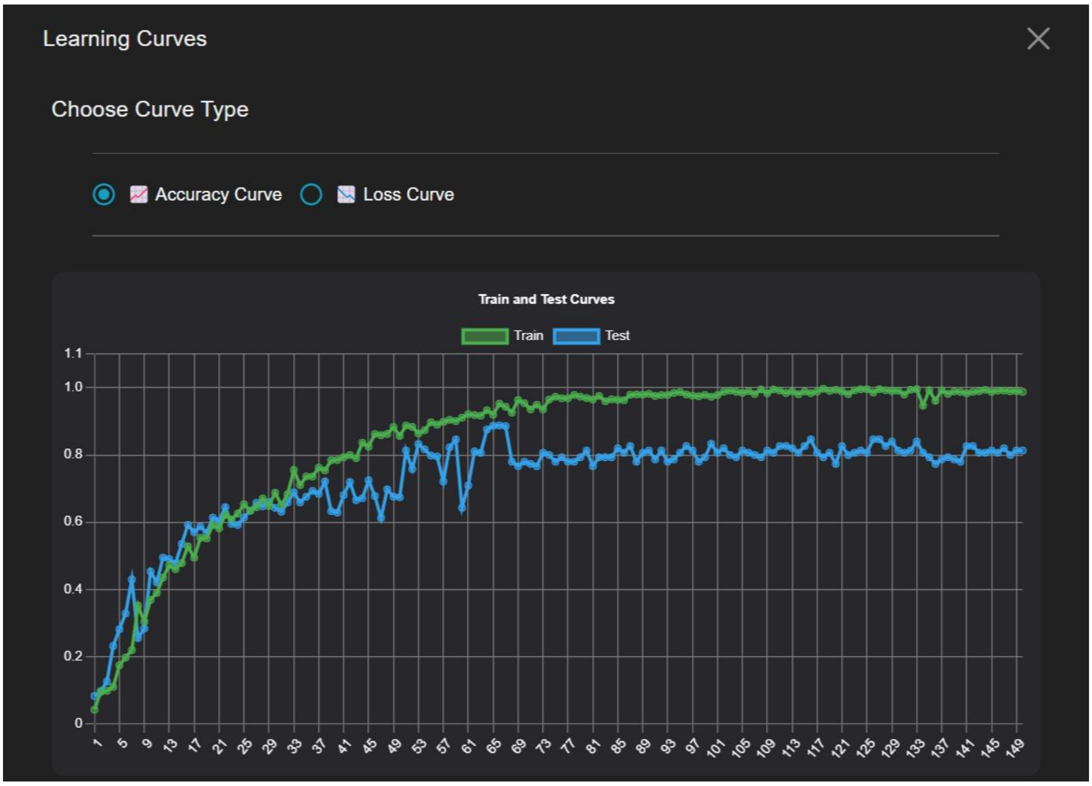
View all your trained models here!

20220930-174026.tar.gz
Train Accuracy: 41% Test Accuracy: 47%

20220926-210912.tar.gz
Train Accuracy: 47% Test Accuracy: 48%

20220926-201902.tar.gz
Train Accuracy: 50% Test Accuracy: 54%

Figure 4.5.1. The UI listing all the trained models along with their train and test accuracies

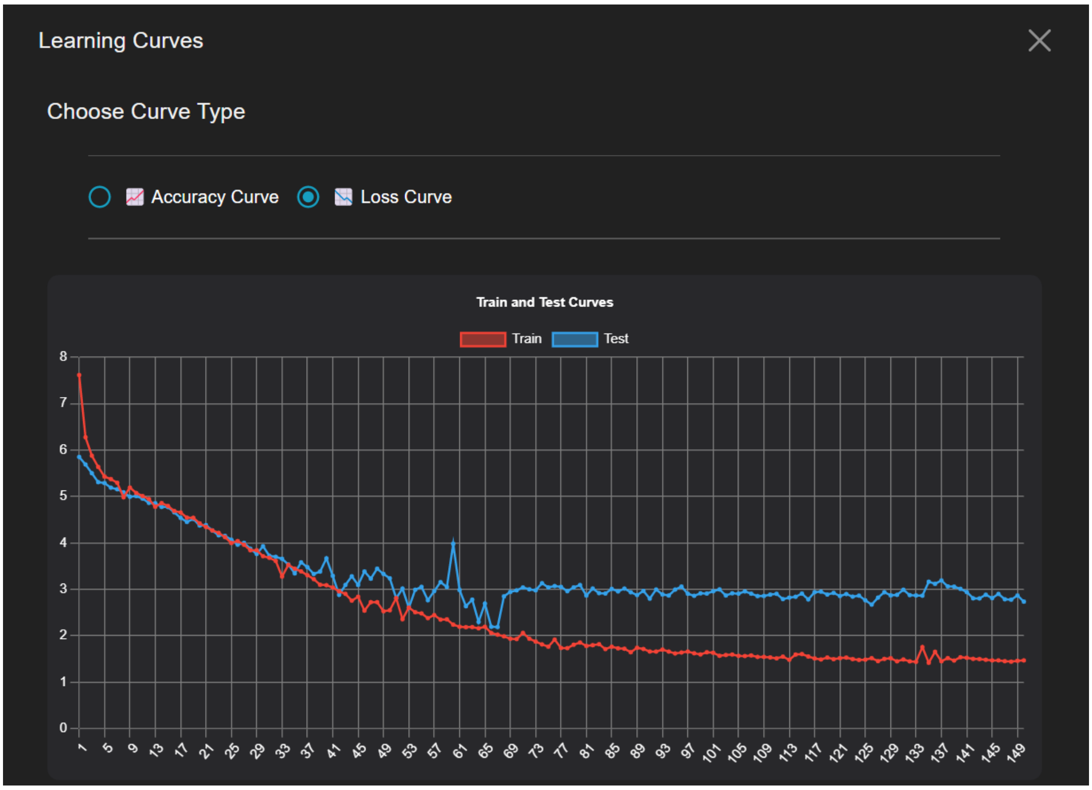


Learning Curves

Choose Curve Type

● Accuracy Curve ○ Loss Curve

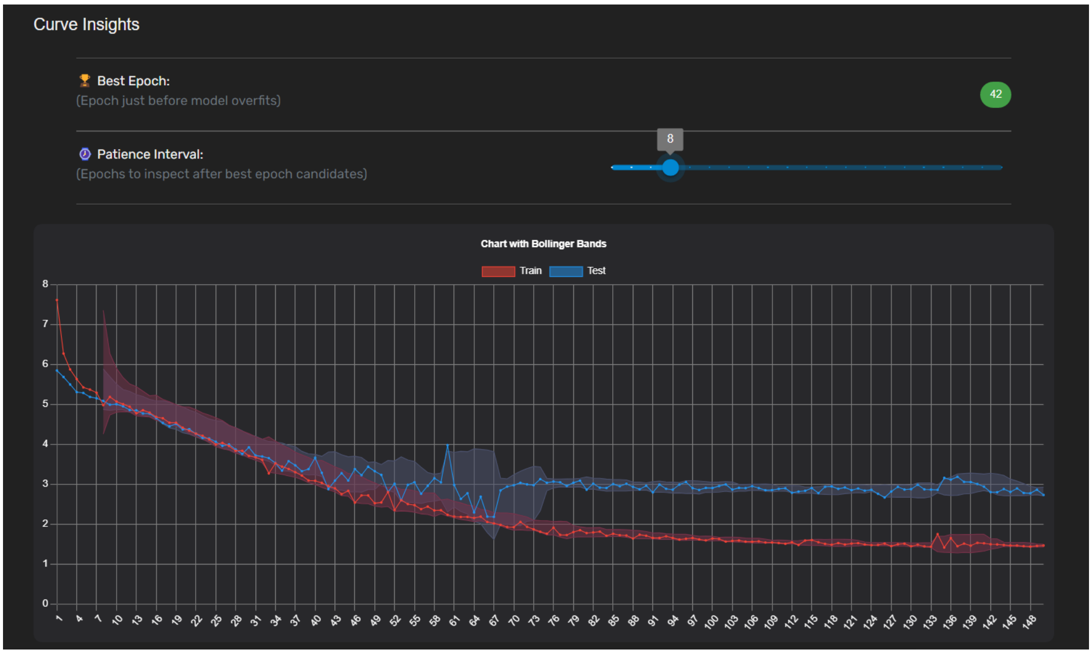
Figure 4.5.2. The accuracy curve of a model


Learning Curves

Choose Curve Type

○ Accuracy Curve ● Loss Curve

Figure 4.5.3. The loss curve of a model


Curve Insights

🏆 Best Epoch:
(Epoch just before model overfits)
42

Patience Interval:
(Epochs to inspect after best epoch candidates)
8

Figure 4.5.4. The best epoch suggested based on the chosen patience interval

4.6 RASAC Python Package Implementation

This individual research component, RASAC has been built into a modular Python package that developers can integrate with any Rasa chatbot project. RASAC has been released as an open-source Python package which anyone install by simply running *pip install rasac* [36]. on any Python 3.7 or 3.8 supported machine. The RASAC Python package adheres to semantic versioning [37]. , and each stable RASAC build is available at Python Package Index (PyPI) as a new version. The package is available under the Apache 2.0 license.

Table 4.6.1 shows the compatibility between RASAC and Rasa Python packages. Developers should pick the correct RASAC package that is fully compatible with the Rasa Python package they use.

Table 4.6.1. RASAC and Rasa compatibility matrix

RASAC Developer Console Version		**Rasa Open-Source Version**
MAJOR.MINOR	**PATCH (Released Bug fixes)**	
2.1.x	2.1.1, 2.1.0	2.8.8
2.0.x	2.0.0	
1.0.x	1.0.1, 1.0.0	
0.0.x	0.0.1a2, 0.0.1a1	

4.7 Tools and Technologies

4.7.1 RASAC Package Implementation

RASAC has utilized Python 3.8 as the backend programming language. The RASAC implementation supports both Python 3.7 and 3.8. This version restriction is due to Rasa 2.8.8 which does not support newer Python versions. The Rasa open-source package was used to train a chatbot and test the RASAC functionalities.

4.7.2 RASAC Server Implementation

The RASAC server and API are a single Flask-based server that facilitates both local and production-level deployments. The production deployment of the RASAC server relies on a Web Server Gateway Interface (WSGI) server, waitress [38]. The waitress server was primarily selected due to its extensive support for Windows and Linux operating systems, which is provided out-of-the-box. Developers can run the RASAC server in debug mode by running the command *rasac server* followed by the --*debug* flag, which disables the waitress production deployment mode and executes the flask server directly.

4.7.3 RASAC Server Frontend Development

React was utilized as the frontend development framework. It uses Node 16.15.0 and utilizes axios, material-ui, prop-types, uuid, framer-motion, react-router, and bolllinger-bands NPM packages. The entire frontend resides within the flask backend server which is explained in section 4.7.2, and does not require a separate deployment.

4.7.4 Integrated Development Environment

PyCharm (backend) and Visual Studio Code (frontend) were used as the primary Integrated Development Environments (IDEs) and Anaconda Python distribution to efficiently manage the Python virtual environments. Also *.env* files are used to hold the required environmental variables and credentials and uses *requirements.txt* files to state required Python packages for future references and deployment.

4.7.5 Source Code, Version, and Release Management

GitLab is used as the source code management platform for the overall research, while GitHub is used for the individual research component for version and release management of the RASAC open-source package. Stable releases of the RASAC Python packages are available in PyPI, where any developer can download and install RASAC similar to any other Python package. All releases of RASAC adhere to semantic versioning, making it easy to identify major, minor, and patch releases and breaking changes.

Table 4.7.5.1 summarizes the tools and technologies states in all sub-sections of section 4.7

4.8 Commercialization Aspect of the Product

Although the individual research component holds a commercial value, it was decided to release the research component as an open-source Python package due to the following reasons.

1. RASAC is a useful product to the NLP, ML and Developers who are non-ML experts since there no tools providing the ability to configure ML pipeline and policy components and train models solely using the GUI.

2. Since the algorithm used to suggest the best epoch used in RASAC is a novel approach, there is potential for it be improved when used by a vast audience with access to the research.

However, RASAC adds commercial value to the Kolloqe chatbot development platform, the overall research solution, by providing the ability to configure ML pipeline and policy components and train models using just and GUI and then evaluate the trained models using the learning curves generated and displayed in the UI along with the suggested best epoch to use to train the model. Most of these features are not available in most of the chatbot development frameworks, such as Google DialogFlow, Amazon Lex, Microsoft LUIS, and Rasa. The following facts explain how RASAC increases the overall commercial value of Kolloqe apart from what is mentioned above.

1. RASAC comes bundled with a custom tokenizer for Sinhala-English text data which can be configured using the configuration UI of RASAC.

Table 4.7.5.1. Summary of Tools and Technologies utilized

Task	Tools to be used
RASAC Python package development	Python 3.8, NumPy, Pandas, sklearn, Rasa 2.8.8, PyCharm, Visual Studio Code, Google CoLab, Anaconda Distribution, Dotenv
RASAC server frontend implementation and algorithm development	Flask, Flask-Cors, psutil, waitress, Bootstrap, React JS, Axios, UUID4, Material-UI, framer-motion, prop-types, react-router, bollinger-bands
Process queue implementation	SQLite3
Chatbot development	Rasa 2.8.8, Rasa-SDK 2.8.0
Source code and version management	GitHub, GitLab, semantic-versioning, PyPI
Testing	Jest, Docker

5. TESTING & IMPLEMENTATION RESULTS & DISCUSSION

5.1 Results

The results section consists of various testing approaches incorporated to test different aspects of the RASAC implementation.

5.1.1 Server Testing

The test cases in Table 5.1.1.1 and Table 5.1.1.2 covers the tasks of setting up the RASAC server in development and production mode respectively, using the CLI.

5.1.2 Codeless Model Improvement Results

One of the main reasons behind implementing a GUI to configure the ML pipeline and policy components was to reduce the time taken to configure the model compared to manually typing the YAML configuration file in the backend. To test the time difference between the two approaches, a controlled experiment was set up where four (4) users were given a YAML configuration file and were asked to manually type the file again to mimic developers manually typing the YAML configuration file (this experiment makes the task easy since the test subjects are provided with all the configurations already which saves time from having to go through documentations to know what the required syntax is for a component), and they were also asked to look at the provided file and configure the components using the GUI, and the timing taken for the two tasks were recorded and the recorded timings can be seen in Table 5.1.2.1. As we can see, there is a huge improvement in the efficiency when configuring the

Table 5.1.2.1. Comparing the time taken to manually type a YAML configuration file for a ML model vs using the RASAC GUI to configure the model

User	**Time taken to manually type the config file (min:sec)**	**Time taken to configure using the GUI (min:sec)**	**Efficiency Percentage (%)**
User 1	10:26	2:30	317%
User 2	14:28	2:28	486%
User 3	15:15	2:25	531%
User 4	09:36	1:39	482%
Average	12:43	2:16	449%

model using the RASAC GUI compared to manually typing the configuration file, with the average efficiency being 449%.

The test cases from Table 5.1.2.2 to Table 5.1.2.5 covers the tasks of configuring the ML pipeline and policy components from scratch, configuring ML pipeline and policy components using existing model configurations, configuring training models, and aborting a model that is in the process of being trained respectively, using the *Configurations* GUI.

Table 5.1.1.1. Test Case for Testing RASAC Server in Debug Mode

Project ID: 2022-056-IT19064932					
Project Name: RASAC					
Project Function: RASAC Development Server Deployment					
Test case ID: 001			**Test case designed by:** **ID No:** IT19064932 **Name:** Hameed M.S.		
Test Priority (High/Medium/Low): Medium					
Test Description: Test the RASAC server initialization in debug mode					
Prerequisite: RASAC Python package must be installed in the virtual environment					
Test Steps: Step 1: Open a terminal and activate the conda environment Step 2: Go to the project directory in the terminal Step 3: Run *rasac server* command with *–debug* to turn on debugging Step 4: Wait until the RASAC server starts in the debug mode and click the URL to open a web browser and confirm the server is up and running					
Test ID	Test Inputs	Expected Outputs	Actual Output	Result (Pass/Fail)	Comments
001	*rasac server --debug* command	RASAC server must initialize in the debug mode and serve the RASAC developer console on port 6069	It can be seen in the console that the server is starting in *development* mode. The server loads as expected in port 6069. The developer console opens confirming that the server hosts the RASAC developer console without any issues.	Pass	RASAC server is served in debug mode as required

Table 5.1.1.2. Test Case for Testing RASAC Server in Production Mode

Project ID: 2022-056-IT19064932					
Project Name: RASAC					
Project Function: RASAC Production Server Deployment					
Test case ID: 002			**Test case designed by:** **ID No:** IT19064932 **Name:** Hameed M.S.		
Test Priority (High/Medium/Low): Medium					
Test Description: Test the RASAC server initialization in production mode					
Prerequisite: RASAC Python package must be installed in the virtual environment					
Test Steps: Step 1: Open a terminal and activate the conda environment Step 2: Go to the project directory in the terminal Step 3: Run *rasac server* command Step 4: Wait until the RASAC server starts in production mode and click the URL to open a web browser and confirm the server is up and running					
Test ID	Test Inputs	Expected Outputs	Actual Output	Result (Pass/Fail)	Comments
002	*rasac server* command	RASAC server must initialize in the production mode and serve the RASAC developer console on port 6069	It can be seen in the console that the server is starting in *production* mode. The server loads as expected in port 6069. The developer console opens confirming that the server hosts the RASAC developer console without any issues.	Pass	RASAC server is served in production mode as required

Table 5.1.2.2. Test Case for Testing the Pipeline and Policy Components in the Configurations GUI

<table>
<tr><td colspan="6"><b>Project ID:</b> 2022-056-IT19064932</td></tr>
<tr><td colspan="6"><b>Project Name:</b> RASAC</td></tr>
<tr><td colspan="6"><b>Project Function:</b> Pipeline and Policy Configuration Selection</td></tr>
<tr><td colspan="3"><b>Test case ID:</b> 001</td><td colspan="3"><b>Test case designed by:</b>
<b>ID No:</b> IT19064932
<b>Name:</b> Hameed M.S.</td></tr>
<tr><td colspan="6"><b>Test Priority (High/Medium/Low):</b> High</td></tr>
<tr><td colspan="6"><b>Test Description:</b> Configurations selected in the UI should be passed to the config file in the backend</td></tr>
<tr><td colspan="6"><b>Prerequisite:</b> The RASAC server must be running</td></tr>
<tr><td colspan="6"><b>Test Steps:</b>
Step 1: Select the required pipeline and policy components from the configuration UI
Step 2: Click the <i>train</i> button in the UI
Step 3: Check the YAML config file in the backend</td></tr>
<tr><td>Test ID</td><td>Test Inputs</td><td>Expected Outputs</td><td>Actual Output</td><td>Result (Pass/Fail)</td><td>Comments</td></tr>
<tr><td>001</td><td>Choosing the required pipeline and policy components in the UI</td><td>The backend YAML config file being populated with the selected pipeline and policy components</td><td>The backend YAML config file is populated with the selected pipeline and policy components</td><td>Pass</td><td>Pipeline and policy component selection works as required</td></tr>
</table>

Table 5.1.2.3. Test Case for Testing the Option to Populate the Pipeline and Policy Components in the Configurations GUI using existing models

<table>
<tr><td colspan="6"><b>Project ID:</b> 2022-056-IT19064932</td></tr>
<tr><td colspan="6"><b>Project Name:</b> RASAC</td></tr>
<tr><td colspan="6"><b>Project Function:</b> Pipeline and Policy Existing Model Configurations</td></tr>
<tr><td colspan="3"><b>Test case ID:</b> 002</td><td colspan="3"><b>Test case designed by:</b>
<b>ID No:</b> IT19064932
<b>Name:</b> Hameed M.S.</td></tr>
<tr><td colspan="6"><b>Test Priority (High/Medium/Low):</b> High</td></tr>
<tr><td colspan="6"><b>Test Description:</b> Selecting pipeline and policy component configurations from existing models</td></tr>
<tr><td colspan="6"><b>Prerequisite:</b> The RASAC server must be running</td></tr>
<tr><td colspan="6"><b>Test Steps:</b>
Step 1: Click the dropdown denoted by <i>Existing Models</i>
Step 2: Select a model from the dropdown
Step 3: Check the pipeline and policy components automatically selected in the UI against the components previously selected when configuring the selected model</td></tr>
<tr><td>Test ID</td><td>Test Inputs</td><td>Expected Outputs</td><td>Actual Output</td><td>Result (Pass/Fail)</td><td>Comments</td></tr>
<tr><td>002</td><td>Select a model from the <i>Existing Models</i> dropdown</td><td>The pipeline and policy components automatically selected in the UI are the same as the ones selected when training the selected model</td><td>The correct pipeline and policy components are automatically selected in the UI</td><td>Pass</td><td>Choosing existing model configurations work as required</td></tr>
</table>

Table 5.1.2.4. Test Case for Testing the Option to Train a Model in the Configurations GUI

<table>
<tr><td colspan="6"><b>Project ID:</b> 2022-056-IT19064932</td></tr>
<tr><td colspan="6"><b>Project Name:</b> RASAC</td></tr>
<tr><td colspan="6"><b>Project Function:</b> Successfully train a ML model</td></tr>
<tr><td colspan="3"><b>Test case ID:</b> 003</td><td colspan="3"><b>Test case designed by:</b>
<b>ID No:</b> IT19064932
<b>Name:</b> Hameed M.S.</td></tr>
<tr><td colspan="6"><b>Test Priority (High/Medium/Low):</b> High</td></tr>
<tr><td colspan="6"><b>Test Description:</b> Training an ML model using the UI after selecting the required components</td></tr>
<tr><td colspan="6"><b>Prerequisite:</b> The RASAC server must be running</td></tr>
<tr><td colspan="6"><b>Test Steps:</b>
Step 1: Select the required pipeline and policy components from the configuration UI
Step 2: Click the <i>train</i> button in the UI
Step 3: Check for the toast message once training ends</td></tr>
<tr><td>Test ID</td><td>Test Inputs</td><td>Expected Outputs</td><td>Actual Output</td><td>Result (Pass/Fail)</td><td>Comments</td></tr>
<tr><td>003</td><td>Select the required configurations and train the model</td><td>• Receiving a toast message saying the model was trained successfully
• The trained model can be viewed in the <i>Models</i> tab</td><td>• A toast message is displayed saying the model was trained successfully
• The trained model is available to be viewed in the <i>Models</i> tab</td><td>Pass</td><td>Model training works as required</td></tr>
</table>

Table 5.1.2.5. Test Case for Testing the Option to Populate the Pipeline and Policy Components in the Configurations GUI using existing models

Project ID: 2022-056-IT19064932	
Project Name: RASAC	
Project Function: Successfully cancelling a model that is training	
Test case ID: 004	**Test case designed by:** **ID No:** IT19064932 **Name:** Hameed M.S.
Test Priority (High/Medium/Low): High	
Test Description: Aborting the training process of a model that is currently training	
Prerequisite: The RASAC server must be running	
Test Steps: Step 1: Select the required pipeline and policy components from the configuration UI Step 2: Click the *train* button in the UI Step 3: Check whether the model is training (a circular progress bar appears in front of the *train* button) Step 4: Click the *Abort* button (only appears when the model training is in progress)	

Test ID	Test Inputs	Expected Outputs	Actual Output	Result (Pass/Fail)	Comments
004	Click the *Abort* button when the model training is in progress	Model training aborts with a toast message indicating the model training aborted successfully	Model training aborts with a toast message indicating the model training was aborted successfully	Pass	Model training abortion works as required

5.1.3 Model Performance Evaluation Results

Table 5.1.3.1 shows the results of a test conducted, where four users who are data science undergraduates, were shown the loss curve of two ML models and were asked to state what they thought was the best epoch. And then the algorithm was used to find the best epoch on the said two curves while changing the patience interval values between 5, 10, 15, 20, and 25.

The test cases from Table 5.1.3.2 to Table 5.1.3.5 covers the tasks of checking a model's training and testing accuracy, viewing the accuracy curve of a model, viewing the loss curve of a model, and testing out the suggesting the best epoch feature respectively, using the *Models* GUI.

Table 5.1.3.1. Comparing the best epoch according to four users against the best epoch suggested by the algorithm

User	Best epoch according to user	Best epoch suggested by algorithm at different patient interval values				
		5	10	15	20	25
Model with 150 Epochs (Figure 5.1.3.1)						
User 1	67	42	53	67	67	67
User 2	67					
User 3	67					
User 4	67					
Model with 90 Epochs (Figure 5.1.3.2)						
User 1	36	36	36	52	60	68
User 2	36					
User 3	36					

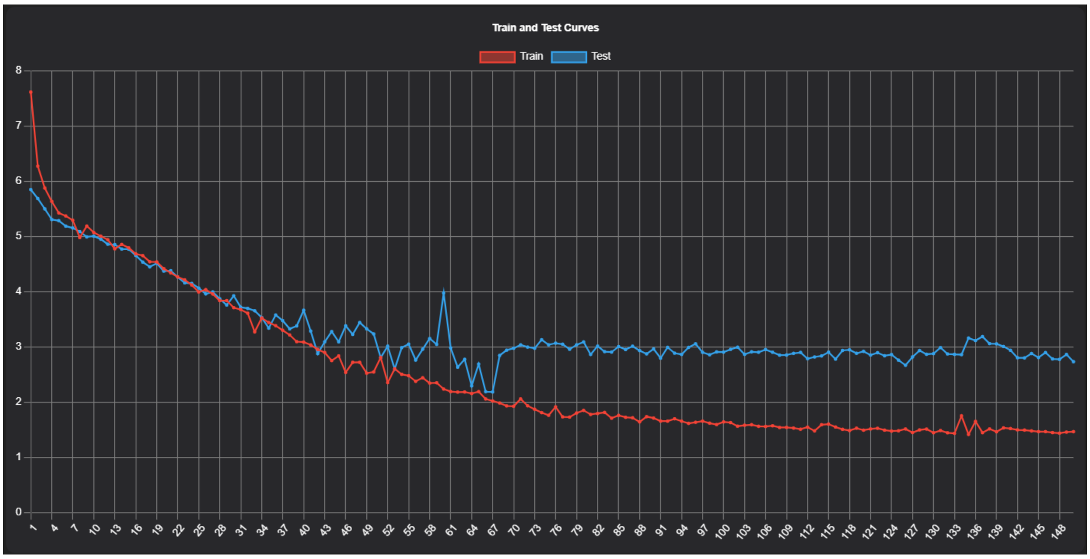


Figure 5.1.3.1. A loss curve with 150 epochs

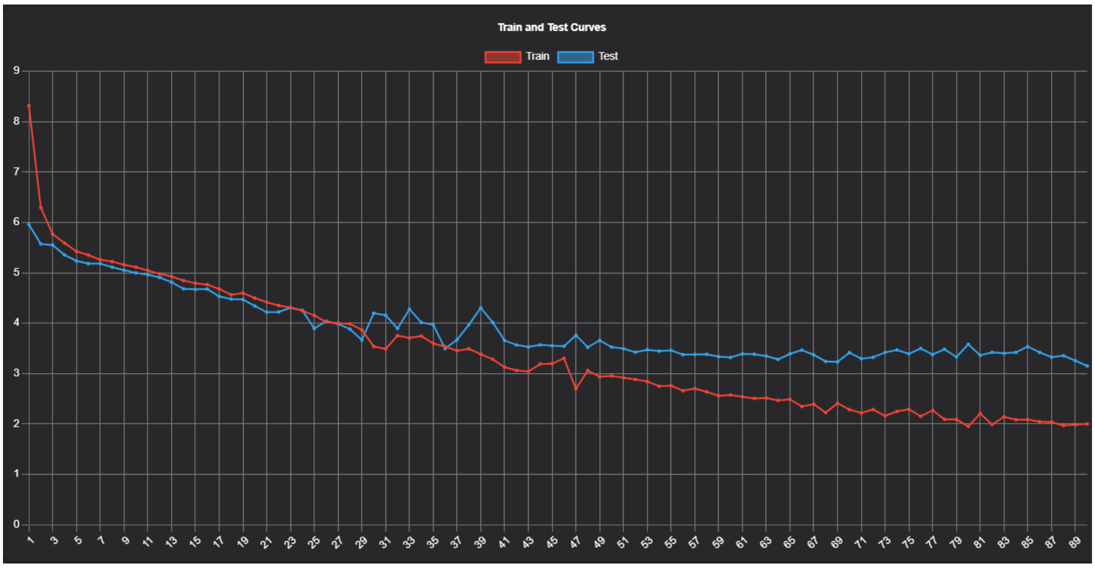


Figure 5.1.3.2. A loss curve with 90 epochs

Table 5.1.3.2. Test Case for Testing the Option to Check the Training and Testing Accuracy of a Model in the Models GUI

Project ID: 2022-056-IT19064932	
Project Name: RASAC	
Project Function: Check a model's training and testing accuracy	
Test case ID: 001	**Test case designed by:** **ID No:** IT19064932 **Name:** Hameed M.S.
Test Priority (High/Medium/Low): High	
Test Description: Checking whether the training and testing accuracy of a model is displayed once it is trained	
Prerequisite: Having a model that has already been trained	
Test Steps: Step 1: Select the *Models* UI Step 2: Check below the name of the trained model for the training and testing accuracy	

Test ID	Test Inputs	Expected Outputs	Actual Output	Result (Pass/Fail)	Comments
001	Select the *Models* UI and check the training and testing accuracy below the name of the trained model	The models training and testing accuracy are displayed below the name of the model	The models training and testing accuracy are displayed below the name of the model	Pass	Model accuracy is displayed as required. If an error occurred with displaying the accuracy it would display *N/A* indicating the accuracy is Not Available

Table 5.1.3.3. Test Case for Testing the Option to View the Accuracy Curve of a Model in the Models GUI

Project ID: 2022-056-IT19064932	
Project Name: RASAC	
Project Function: View a model’s accuracy curve	
Test case ID: 002	**Test case designed by:** **ID No:** IT19064932 **Name:** Hameed M.S.
Test Priority (High/Medium/Low): High	
Test Description: Viewing a model’s accuracy curve	
Prerequisite: Having a model that has already been trained	
Test Steps: Step 1: Select the *Models* UI Step 2: Click the *Curve* button of the trained model Step 3: Click the *Accuracy Curve* radio button under *Choose Curve Type* in the new sliding window (selected by default)	

Test ID	Test Inputs	Expected Outputs	Actual Output	Result (Pass/Fail)	Comments
002	Select the *Accuracy Curve* from the Models curve types	The model’s accuracy curve is displayed	The model’s accuracy curve is displayed	Pass	The model’s accuracy curve is displayed as required

Table 5.1.3.4. Test Case for Testing the Option to View the Loss Curve of a Model in the Models GUI

Project ID: 2022-056-IT19064932	
Project Name: RASAC	
Project Function: View a model’s loss curve	
Test case ID: 003	**Test case designed by:** **ID No:** IT19064932 **Name:** Hameed M.S.
Test Priority (High/Medium/Low): High	
Test Description: Viewing a model’s loss curve	
Prerequisite: Having a model that has already been trained	
Test Steps: Step 1: Select the *Models* UI Step 2: Click the *Curve* button of the trained model Step 3: Click the *Loss Curve* radio button under *Choose Curve Type* in the new sliding window	

Test ID	Test Inputs	Expected Outputs	Actual Output	Result (Pass/Fail)	Comments
003	Select the *Loss Curve* from the Models curve types	The model’s loss curve is displayed	The model’s loss curve is displayed	Pass	The model’s loss curve is displayed as required

Table 5.1.3.5. Test Case for Testing the Option to View the Accuracy Curve of a Model in the Models GUI

Project ID: 2022-056-IT19064932	
Project Name: RASAC	
Project Function: Suggesting the best epoch	
Test case ID: 004	**Test case designed by:** **ID No:** IT19064932 **Name:** Hameed M.S.
Test Priority (High/Medium/Low): High	
Test Description: Checking the best epoch suggested by the UI	
Prerequisite: Having a model that has already been trained	
Test Steps: Step 1: Select the *Models* UI Step 2: Click the *Curve* button of the trained model Step 3: Scroll down in the new sliding window to the *Curve Insights* section Step 4: Slide to change the *Patience Interval* value as required	

Test ID	Test Inputs	Expected Outputs	Actual Output	Result (Pass/Fail)	Comments
004	Changing the *Patience Interval* value by sliding the slider back and forth	An appropriate value is suggested as the best epoch. (The lowest epoch at a given point is returned as the best epoch. Ideally, larger patience interval value returns a higher best epoch value)	An appropriate value is suggested as the best epoch	Pass	The algorithm suggesting the best epoch works as required

5.2 Research Findings

5.2.1 Codeless Model Improvement Research Findings

Initially, the idea was to create a drag and drop UI, where users can drag and drop the components they need. But the drag and drop approach did not seem intuitive since each component has their own set of parameters that need to be set and dragging and dropping components complicated the process beating the whole purpose of the research which was to simplify the process.

It was then decided to provide checkboxes to tick the required components and reduce the number of textboxes where possible and replace them with dropdowns to reduce the possibility of the user inserting an invalid input. Also a feature was added to select the configurations of an existing model, which makes it convenient for users that plan on using the same components while only changing values of a few parameters.

5.2.2 Model Performance Evaluation Research Findings

Initially, the idea was to try and predict where the learning curve of a model would head based on the curve generated by the existing epochs to decide whether the model can be trained further or not. To achieve this, accuracy curve data points of a little over 100 epochs were taken and several models were used to predict where the curve would lead up until the 500th epoch mark as seen in Figure 5.2.2.1. As can be seen, neither of the models provided a satisfactory prediction of the accuracy curve. The models used and denoted in the Figure are explained further below.

1. **original_y** – These are the normal data points

2. **interpolated_y** – Using interpolation to try and guess the remaining data points

3. **smoothed_ interpolated_y –** Smoothing the curves obtained from 2 above

4. **numpy_polyfit_y_deg=3 –** Using the *polyfit* method from the NumPy library. This uses least squares polynomial fitting with a 3rd degree fitting polynomial.

5. **svr_poly_deg=3** – Using Epsilon-Support Vector Regression with a *poly* kernel and the degree of the polynomial kernel function set to 3.

6. **svr_rbf -** Using Epsilon-Support Vector Regression with the hyperparameters being kernel=”rbf”, C=1.0, epsilon=0.2.

7. **m_svr_rbf** – Using Epsilon-Support Vector Regression with the hyperparameters being kernel=”rbf”, C=100, gamma=0.1, and epsilon=0.1.

8. **sklearn_poly_deg=??** – sklearn’s linear regression with polynomial features were used along with grid search to auto adjust the degree value between 2 and 10.

Since the attempt at trying to predict the path of the accuracy curve was unsuccessful, it was decided to try and suggest what the best epoch is for a given model based on the epochs used to train the model. To achieve this an algorithm was built, by using the moving average, moving standard deviation, and the number of epochs to take into account when performing the calculations on the values of the loss curve data points of each epoch, to suggest the best epoch. This is done on the train and test data points of the loss curves, first, the training dataset is taken and the moving average is calculated for the data points taking into account the number of epochs to consider for it, which is referred to as the patience interval in this component and is chosen by the user. Once the moving average is calculated, the two times positive and the negative moving standard deviation is calculated for the data points, and the two curves are plotted above and below the original train loss curve. These two bands are also called Bollinger bands which are used in stock trading. Next, the testing data points are considered, and the first data point is considered as the lowest epoch, and then each corresponding epoch which is lower than the previous and is within the two, test curve moving standard deviations is considered as the new lowest epoch. The reason for checking whether the epoch is within the two, test curve moving standard deviations is to make sure the test curve is not deviating away from the training curve more than necessary which would suggest the model is overfitting, and also to avoid considering epochs which are outliers. Once an epoch is taken as the lowest epoch, the number given as the patience interval is the number of epochs that will be checked from the current lowest epoch to find an epoch than is lower than the current epoch, and if a lower epoch is not found within the number of epochs mentioned by the patience interval, the algorithm is terminated, and the current lowest epoch is returned as the best epoch.

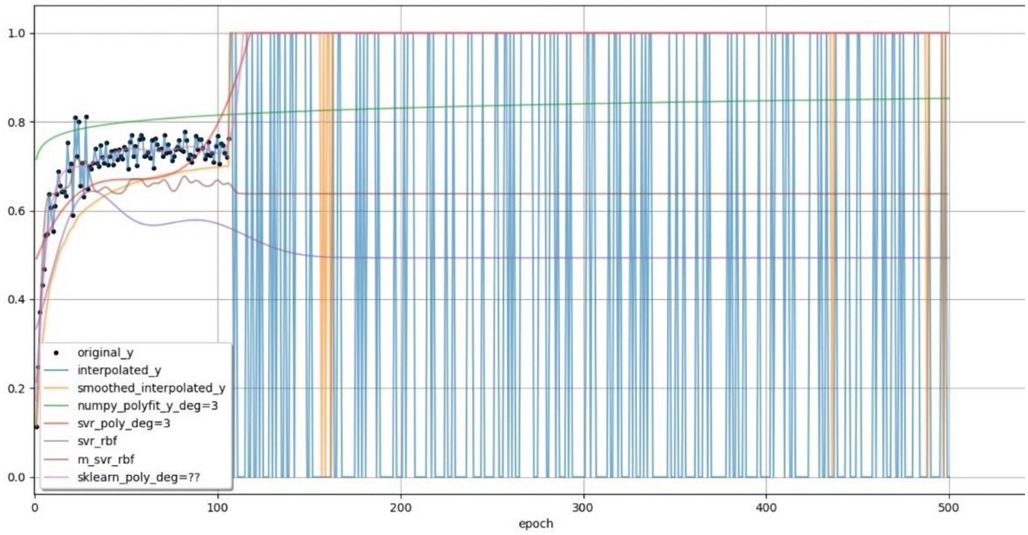


Figure 5.2.2.1. ML models used to predict the path of the accuracy curve

5.3 Discussion

In this component, it was found that using the GUI created to configure the ML pipeline and policy components drastically reduced the configuration time as explained in section 5.1.2. According to which, there is a 449% increase in efficiency. To put it in timing, that on average 12 minutes and 43 seconds to manually type a YAML configuration file compared to 2 mins and 16 seconds on average to use the GUI to configure the same configurations. Also using the GUI to configure the components eliminates the need to be familiar with the syntax of each component and their parameters, and users do not face any runtime errors due to syntax errors which they normally would when manually typing and configuring a YAML configuration file.

Section 5.1.3 proves the reliability of the algorithm and the result is shown instantaneously proving the efficiency of the algorithm. This enables users to effortlessly evaluate models choose the best performing model, or else improve the existing model based on the provided evaluation.

RASAC has been extensively tested and can run on both Windows and Linux operating systems and on Docker without any issues. RASAC can be utilized by any Rasa chatbot developer and integrate it with any project that uses Rasa 2.8.8 since RASAC is available as an open-source package. The selected Rasa version is an actively developed version by Rasa HQ, making RASAC available to more extensive set of developers. As mentioned in sections 5.1.1 through 5.1.3, the RASAC package is constantly worked on to improve and enhance the provided feature sets.

Since the RASAC package is open-source and the implemented solutions are relatively novel, close attention has been paid to the maintainability of the code. All Python-based code adheres to the PEP8 style guide, and each module utilizes Python docstrings to state the purpose of the functions, classes, and methods. RASAC also provides a comprehensive documentation on RASAC CLI, GUI, and Python API. It makes it easy for an interested individual to easily contribute to the project or refer to and understand the source code with minimum effort.

5.4 Individual Contribution to the Overall Research

Section 4.8 clearly states in the detail the individual research component's contribution towards the overall outcome of the research. This research component is responsible for enabling conversational AI model configuration, training, and validation in the Kolloqe chatbot development platform, by simplifying the aforementioned tasks. Refer to sections 4.4 and 4.5 for more detailed explanations regarding the individual research contributions.

6. CONCLUSION

Conversational AI models need to constantly be configured and trained to keep the model updated to be able to handle complex queries that are handed to them and converse in natural conversations. Configuring and training models requires expertise since the developers need to be familiar with the framework used, understand how to configure the backend files, and invoke the necessary CLI commands to train a model. Then again, just training alone does not suffice since it is vital that the model that performs best is chosen. Choosing the best performing model is vital since a model could have high accuracy but that would not directly make it the best-performing model since the model could be overfitted. Hence, it is important to evaluate a model before deciding on the best model. Sections 1 and 2 of this dissertation, discuss in depth the challenges mentioned above and compare existing solutions and their drawbacks.

To the address the issues and drawbacks mentioned above, the author of this research has successfully delivered RASAC, a novel approach that provides the user with a GUI to configure ML pipeline and policy components, train the configured models, and evaluate the trained models, all using just the GUI, without having to deal with the backend or any CLI commands. RASAC comes as a efficient, modular, open-source, and production-grade Python package that developers can integrate with Rasa 2.8.8 chatbot projects. Interested individuals can contribute to the open-source package and utilize it for research or other related work in the NLP domain.

Section 5 of this dissertation elaborates on how testing and evaluation was conducted on the research approaches. The experiments carried out demonstrated how the implemented solution for configuring pipeline and policy components drastically increased the efficiency by over 400% by reducing the time taken to configure a model compared to the traditional approach. Also, it was proven that the algorithm was able to predict the best epoch similar to how a person that is well versed in evaluating learning curves would have predicted it.

Finally, this research component, RASAC, was integrated into Kolloqe: the conversational AI development platform delivered by the overall research, allowing developers to configure, train, and evaluate ML models using just GUI.

REFERENCES

[1]. Turing, A.M.: Computing machinery and intelligence. Mind 59, 433–460 (1950). https://doi.org/10.1093/mind/LIX.236.433

[2]. Brandtzaeg, P.B., Følstad, A.: Why people use chatbots. In: Kompatsiaris, I., et al. (eds.) Internet Science, pp. 377–392. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70284-1_30

[3]. Weizenbaum, J.: ELIZA—a computer program for the study of natural language communication between man and machine. Commun. ACM 9, 36–45 (1966). https://doi.org/10.1145/365153.365168

[4]. M. Mnasri, "Recent advances in conversational NLP : Towards the standardization of Chatbot building," arXiv [cs.CL], 2019.

[5]. Scopus.com. [Online]. Available: https://www.scopus.com. [Accessed: 17-Oct-2022].

[6]. T. Bocklisch, J. Faulkner, N. Pawlowski, A. Nichol, "Rasa: Open Source Language Understanding and Dialogue Management," 2017. [Online]. Available: https://arxiv.org/abs/1712.05181

[7]. "Dialogflow," Google Cloud. [Online]. Available: https://cloud.google.com/dialogflow. [Accessed: 17-Oct-2022].

[8]. Amazon.com. [Online]. Available: https://aws.amazon.com/lex/. [Accessed: 17-Oct-2022].

[9]. Luis.ai. [Online]. Available: https://www.luis.ai/. [Accessed: 17-Oct-2022].

[10]. G. Caldarini, S. Jaf, K. McGarry, "A Literature Survey of Recent Advances in Chatbots," 2022. [Online]. Available: https://arxiv.org/abs/2201.06657

[11]. Z. Luvsandorj, “Introduction to NLP - part 4: Supervised text classification model in python,” Medium, 19-Dec-2021. [Online]. Available: https://towardsdatascience.com/introduction-to-nlp-part-4-supervised-text-classification-model-in-python-96e9709b4267. [Accessed: 06-Feb-2022].

[12]. “Conversational AI: What it is, why you should care and how to do it right,” Social Media Marketing & Management Dashboard, 15-Dec-2021. [Online]. Available: https://blog.hootsuite.com/conversational-ai/. [Accessed: 28-Jan-2022].

[13]. X. Ying, “An overview of overfitting and its solutions”, in Journal of Physics: Conference Series, 2019, vol 1168, bl 022022.

[14]. G. Paris, D. Robilliard, en C. Fonlupt, “Exploring overfitting in genetic programming”, in Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, bll 267–277.

[15]. D. D. Jensen en P. R. Cohen, “Multiple comparisons in induction algorithms”, Machine Learning, vol 38, no 3, bll 309–338, 2000.

[16]. By: IBM Cloud Education, “What is underfitting?,” IBM. [Online]. Available: https://www.ibm.com/cloud/learn/underfitting. [Accessed: 28-Jan-2022].

[17]. M. A. Babyak, “What you see may not be what you get: a brief, nontechnical introduction to overfitting in regression-type models”, Psychosomatic medicine, vol 66, no 3, bll 411–421, 2004.

[18]. G. Varoquaux, L. Buitinck, G. Louppe, O. Grisel, F. Pedregosa, en A. Mueller, “Scikit-learn: Machine learning without learning the machinery”, GetMobile: Mobile Computing and Communications, vol 19, no 1, bll 29–33, 2015.

[19]. F. Pedregosa et al., "Scikit-learn: Machine learning in Python. the Journal of machine Learning research 12 ", 2011.

[20]. K. Team, "Simple. flexible. powerful.," Keras. [Online]. Available: https://keras.io/. [Accessed: 08-Feb-2022].

[21]. "Introduction to rasa X," Open source conversational AI, 10-Dec-2021. [Online]. Available: https://rasa.com/docs/rasa-x/. [Accessed: 22-Jan-2022].

[22]. "Facebook," Facebook.com. [Online]. Available: https://www.facebook.com/business/news/insights/5-reasons-messaging-is-taking-flight-with-travelers. [Accessed: 17-Oct-2022].

[23]. J. Lee, "Do virtual reference librarians dream of digital reference questions?: A qualitative and quantitative analysis of email and chat reference", Australian Academic & Research Libraries, vol 35, no 2, bll 95–110, 2004.

[24]. M. A. Nezami en R. Rukham, "Crowdsourced NLP Retraining Engine in Chatbots", in International Conference on Emerging Technologies and Intelligent Systems, 2021, bll 311–320.

[25]. "Print Edition - The Sunday Times, Sri Lanka," Sundaytimes.lk. [Online]. Available: https://www.sundaytimes.lk/. [Accessed: 17-Oct-2022].

[26]. Dailynews.lk. [Online]. Available: http://www.dailynews.lk. [Accessed: 17-Oct-2022].

[27]. Sundayobserver.lk. [Online]. Available: http://www.sundayobserver.lk/. [Accessed: 17-Oct-2022].

[28]. "SLIIT support desk," Sliit.lk. [Online]. Available: https://support.sliit.lk/. [Accessed: 17-Oct-2022].

[29]. “Curtin university Australian degrees,” SLIIT, 08-Sep-2021. [Online]. Available: https://sliitinternational.lk/. [Accessed: 17-Oct-2022].

[30]. “SLIIT,” SLIIT. [Online]. Available: https://www.sliit.lk/. [Accessed: 17-Oct-2022].

[31]. “William angliss institute @ SLIIT - William angliss institute @ SLIIT,” William Angliss Institute @ SLIIT. [Online]. Available: https://www.cahm.lk/. [Accessed: 17-Oct-2022].

[32]. Sliit.lk. [Online]. Available: https://library.sliit.lk/. [Accessed: 17-Oct-2022].

[33]. “datasets: Textual datasets scraped and collected from publicly available websites and digital documents.” [Online]. Available: https://github.com/kolloqe/datasets. [Accessed: 17-Oct-2022].

[34]. “Training data format,” Rasa.com. [Online]. Available: https://rasa.com/docs/rasa/training-data-format/. [Accessed: 17-Oct-2022].

[35]. “Npm: Bollinger-bands,” npm. [Online]. Available: https://www.npmjs.com/package/bollinger-bands. [Accessed: 16-Oct-2022].

[36]. “Rasac,” PyPI. [Online]. Available: https://pypi.org/project/rasac/. [Accessed: 16-Oct-2022].

[37]. T. Preston-Werner, “Semantic Versioning 2.0.0,” Semantic Versioning. [Online]. Available: https://semver.org/. [Accessed: 16-Oct-2022].

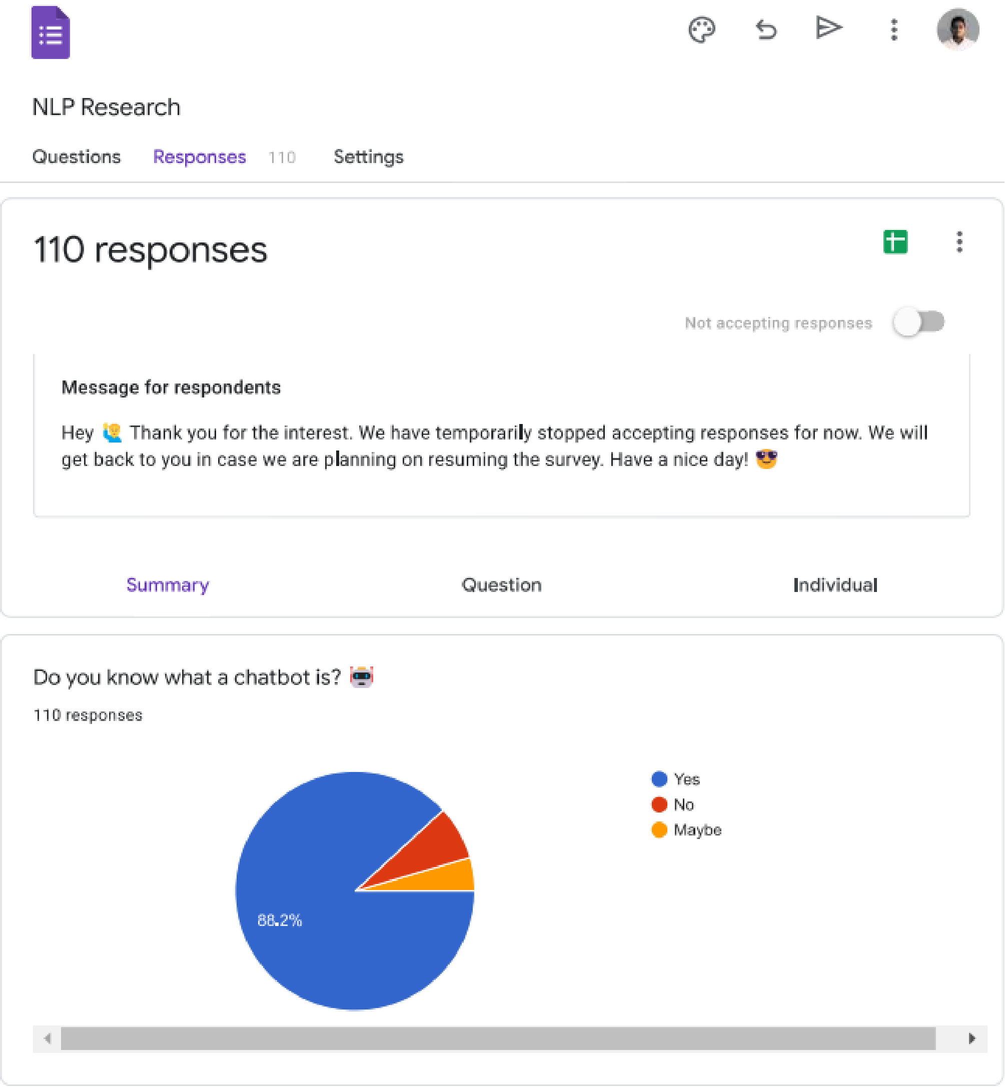
[38]. “Waitress,” PyPI. [Online]. Available: https://pypi.org/project/waitress/. [Accessed: 17-Oct-2022].

GLOSSARY

Term	Definition
Artificial Intelligence (AI)	A field of study and often refers to the intelligence demonstrated by machines in contrast to natural intelligence.
Epoch	The number of passes of the entire training dataset the machine learning algorithm has completed
Machine Learning (ML)	A subfield of AI that concerns development of methods that leverages on data to improve performance in executing a task or set of tasks.
Natural Language Processing (NLP)	A subfield of linguistics, computer science, and artificial intelligence that concerns how to enable machines to process enormous quantities of natural language data.
Rasa Open-Source	An open-source conversational AI development platform developed and maintained by Rasa HQ
Training Dataset	A comprehensive set of data utilized to train machine learning and deep learning models
Holdout Set	A portion of the dataset held back to provide a final estimation of a machine learning or deep learning model performance after training

APPENDICES

Appendix A: Survey Form



110 responses

Not accepting responses

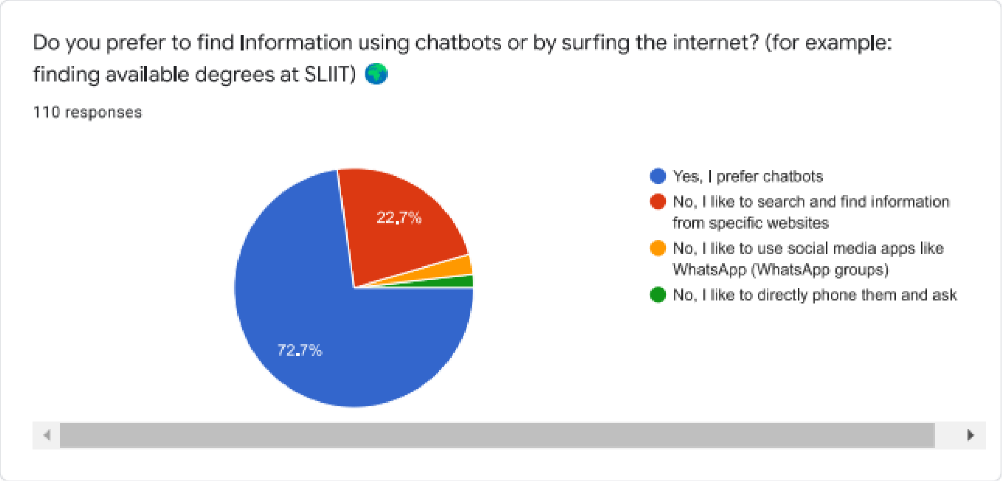
Message for respondents

Hey 🙋 Thank you for the interest. We have temporarily stopped accepting responses for now. We will get back to you in case we are planning on resuming the survey. Have a nice day! 😎

Do you know what a chatbot is? 🤖

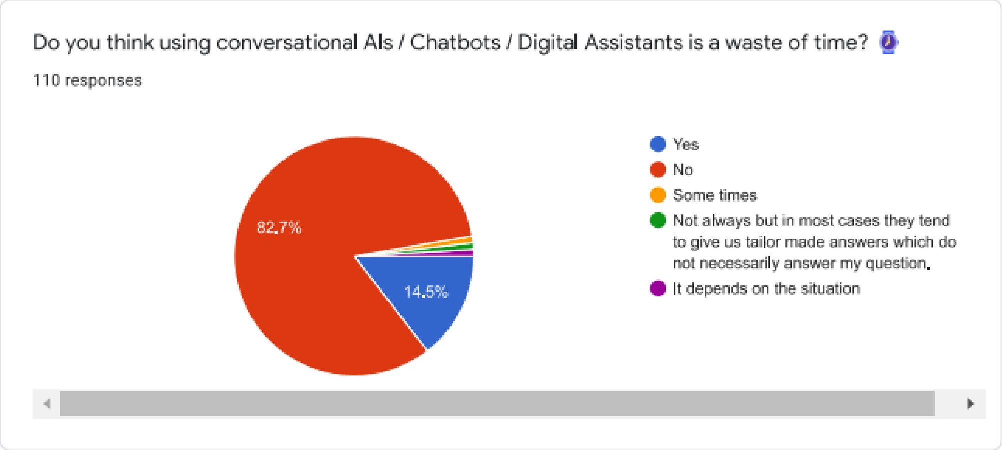
110 responses

Figure A.1: Complete survey form questions and responses – part 1



Do you prefer to find Information using chatbots or by surfing the internet? (for example: finding available degrees at SLIIT) 🌍

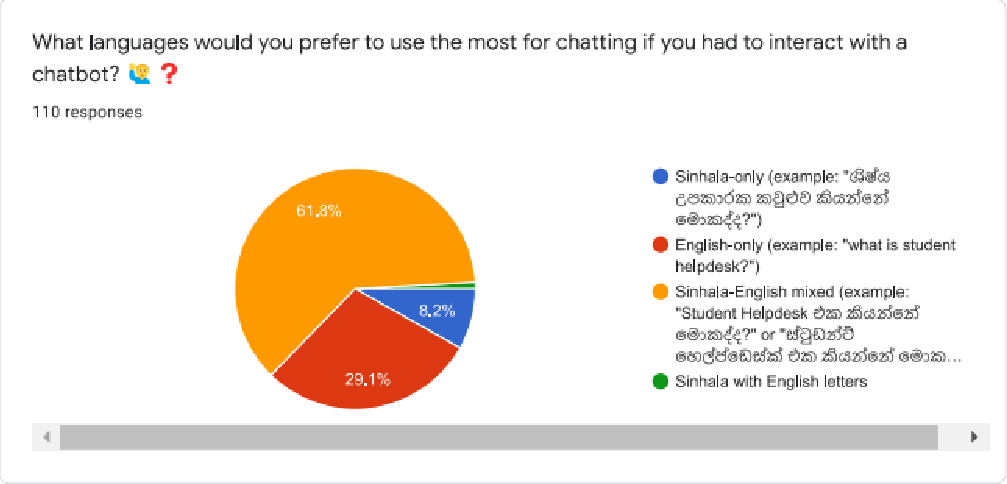
110 responses



Do you think using conversational AIs / Chatbots / Digital Assistants is a waste of time? ⌚

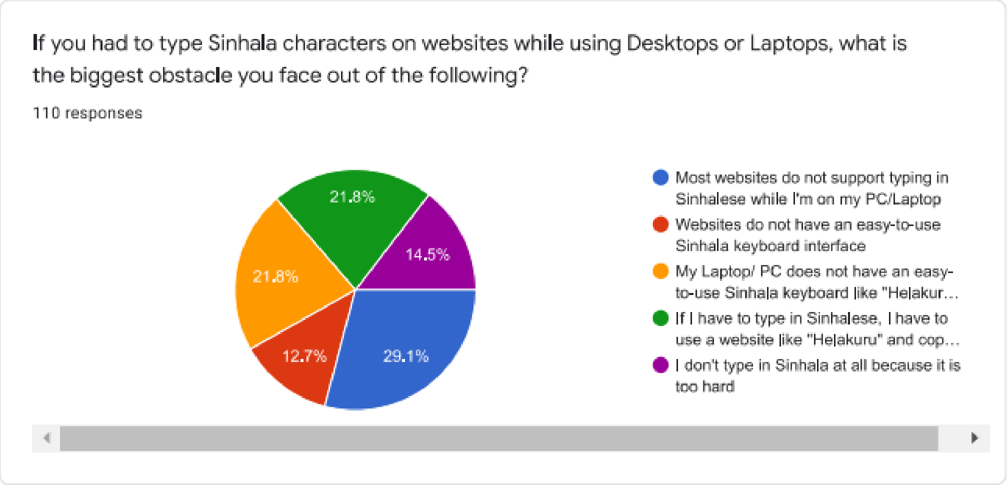
110 responses

Figure A.2: Complete survey form questions and responses – part 2



What languages would you prefer to use the most for chatting if you had to interact with a chatbot? 🙋 ❓

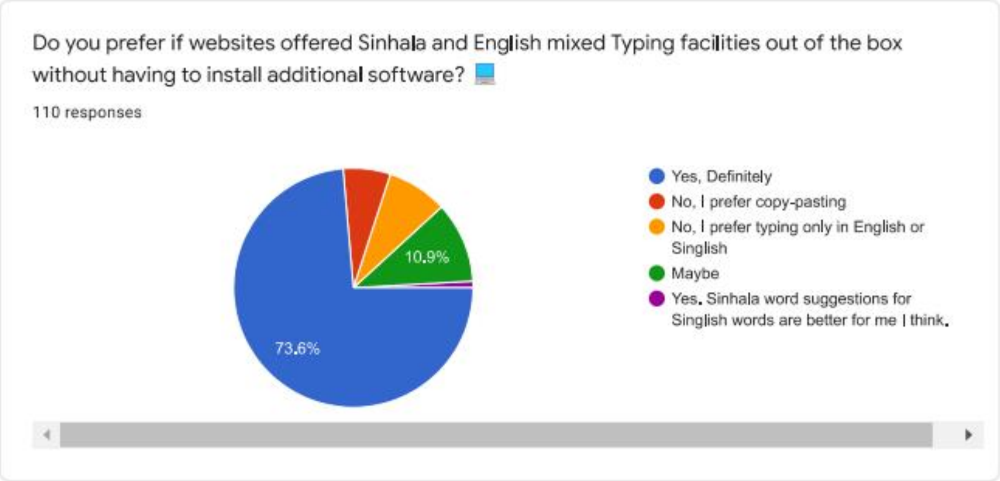
110 responses



If you had to type Sinhala characters on websites while using Desktops or Laptops, what is the biggest obstacle you face out of the following?

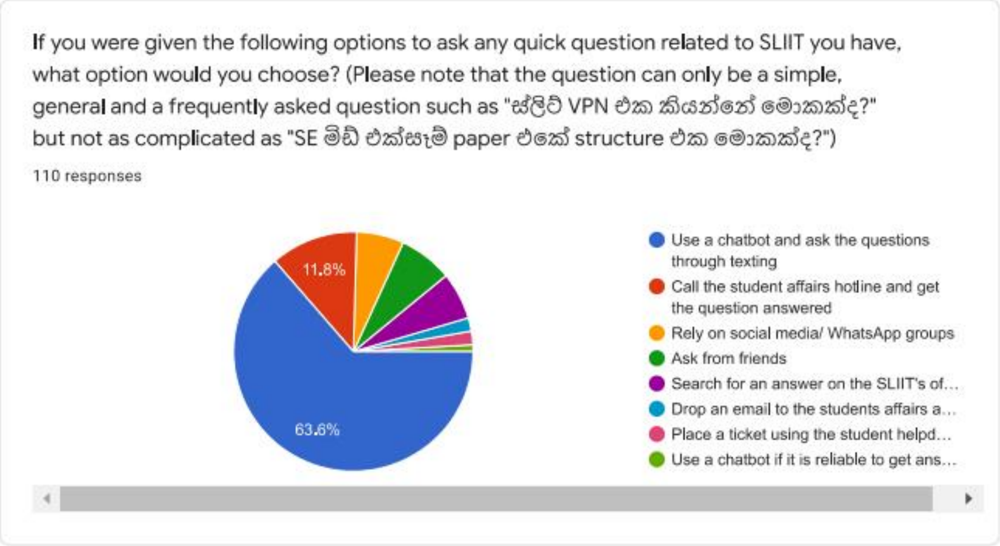
110 responses

Figure A.3: Complete survey form questions and responses – part 3



Do you prefer if websites offered Sinhala and English mixed Typing facilities out of the box without having to install additional software? 💻

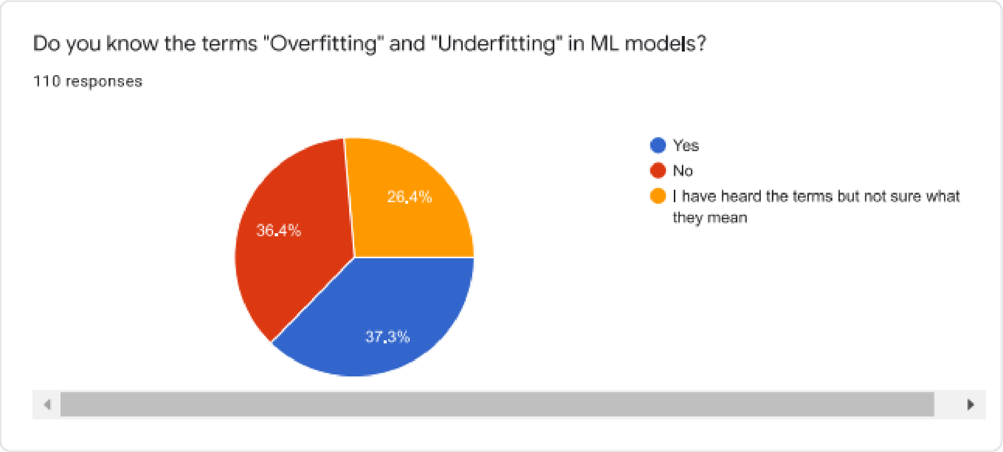
110 responses



If you were given the following options to ask any quick question related to SLIIT you have, what option would you choose? (Please note that the question can only be a simple, general and a frequently asked question such as "ස්ලිට් VPN එක කියන්නේ මොකක්ද?" but not as complicated as "SE මිඩ් එක්සෑම් paper එකේ structure එක මොකක්ද?")

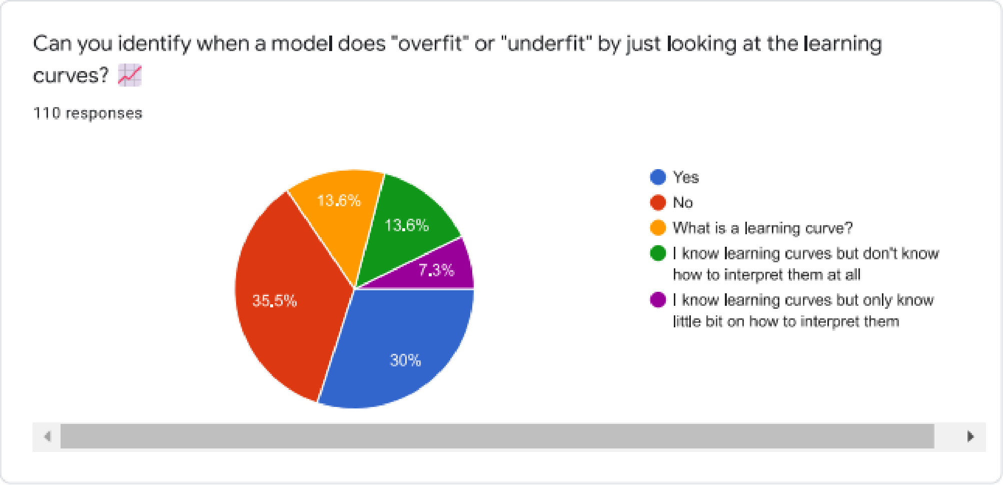
110 responses

Figure A.4: Complete survey form questions and responses – part 4



Do you know the terms "Overfitting" and "Underfitting" in ML models?

110 responses

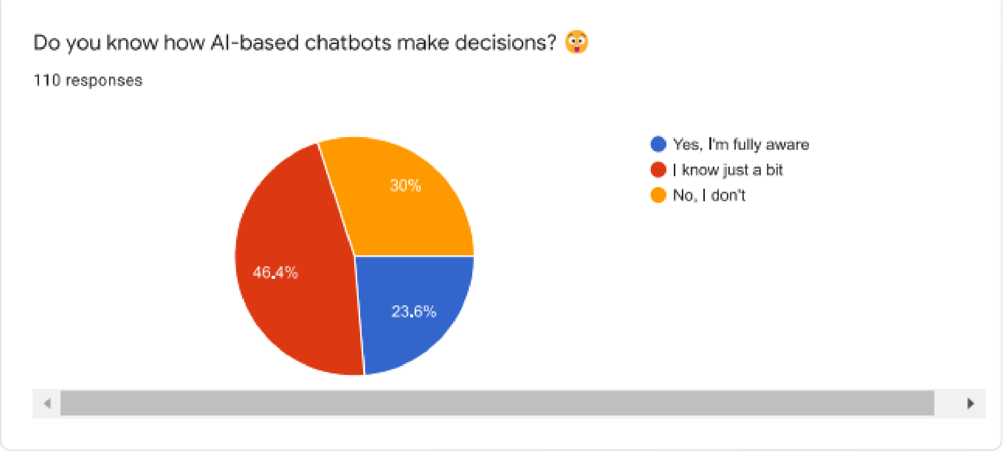


Can you identify when a model does "overfit" or "underfit" by just looking at the learning curves? 📈

110 responses

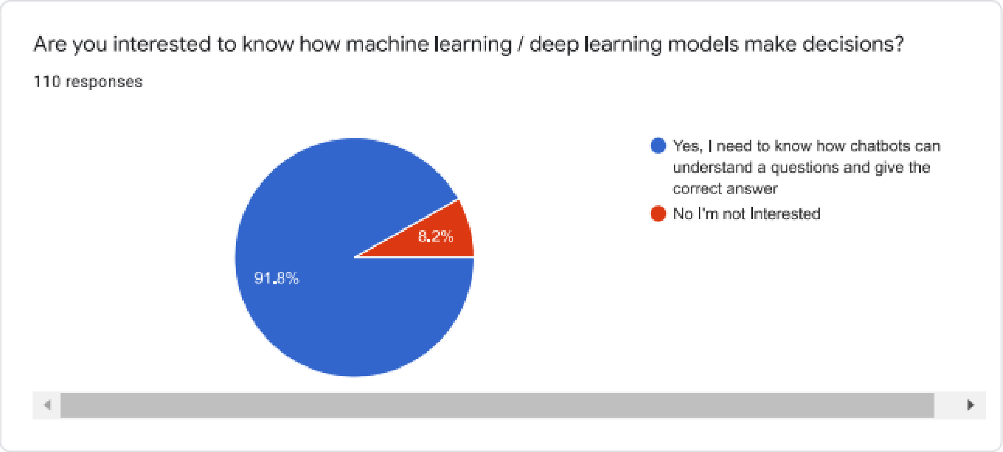
13.6%
13.6%
7.3%
35.5%
30%
● Yes
● No
● What is a learning curve?
● I know learning curves but don't know how to interpret them at all
● I know learning curves but only know little bit on how to interpret them

Figure A. 5: Complete survey form questions and responses – part 5



Do you know how AI-based chatbots make decisions? 😳

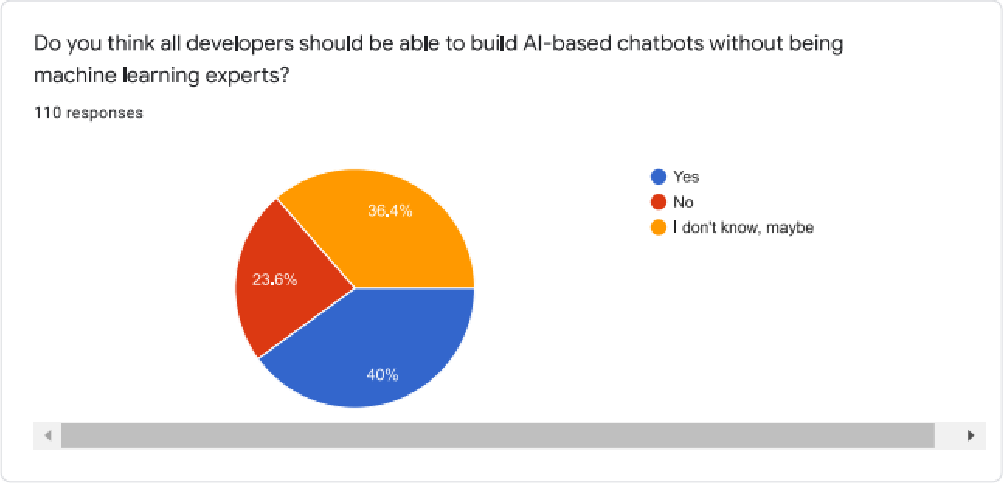
110 responses



Are you interested to know how machine learning / deep learning models make decisions?

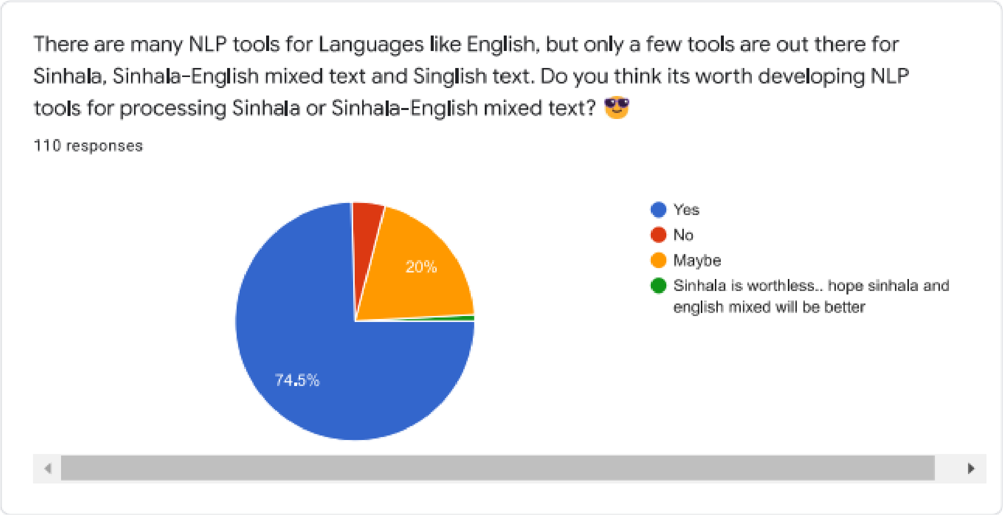
110 responses

Figure A.6: Complete survey form questions and responses – part 6



Do you think all developers should be able to build AI-based chatbots without being machine learning experts?

110 responses

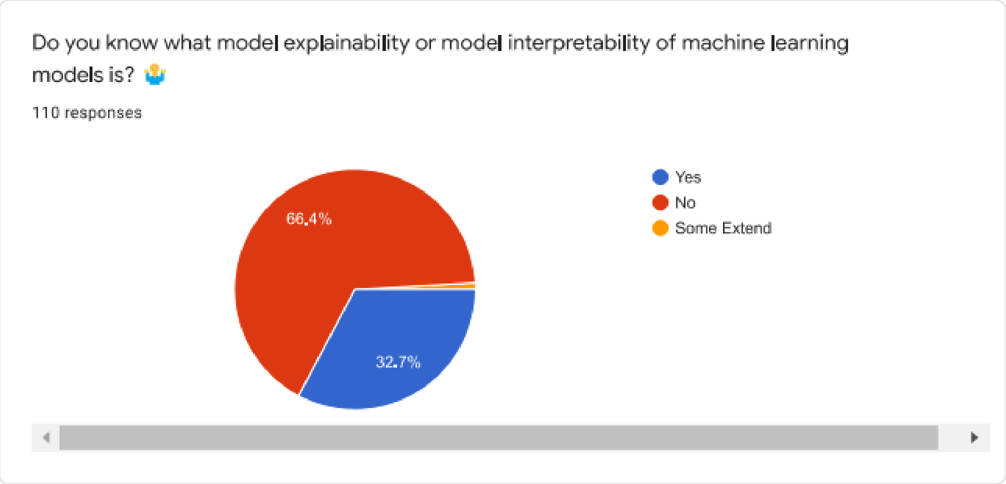


There are many NLP tools for Languages like English, but only a few tools are out there for Sinhala, Sinhala-English mixed text and Singlish text. Do you think its worth developing NLP tools for processing Sinhala or Sinhala-English mixed text? 😎

110 responses

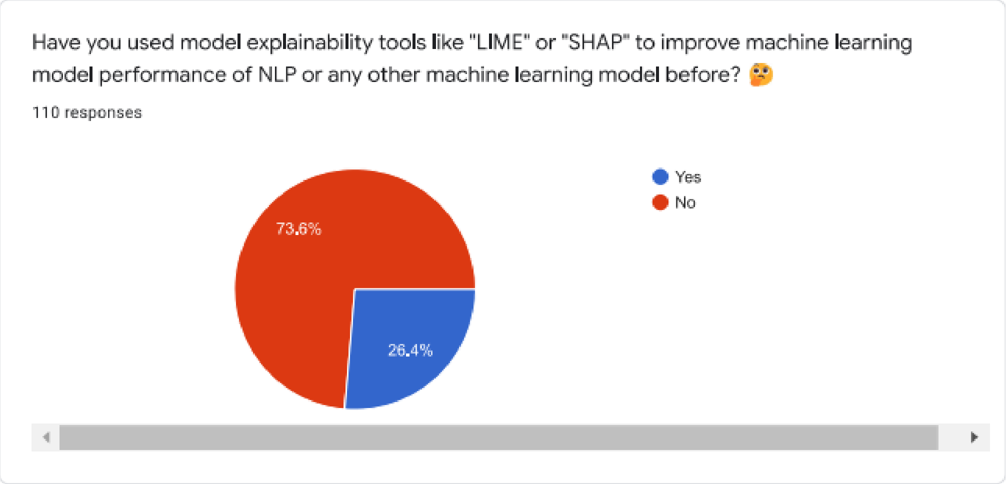
20%
74.5%
● Yes
● No
● Maybe
● Sinhala is worthless.. hope sinhala and english mixed will be better

Figure A.7: Complete survey form questions and responses – part 7



Do you know what model explainability or model interpretability of machine learning models is? 🤷

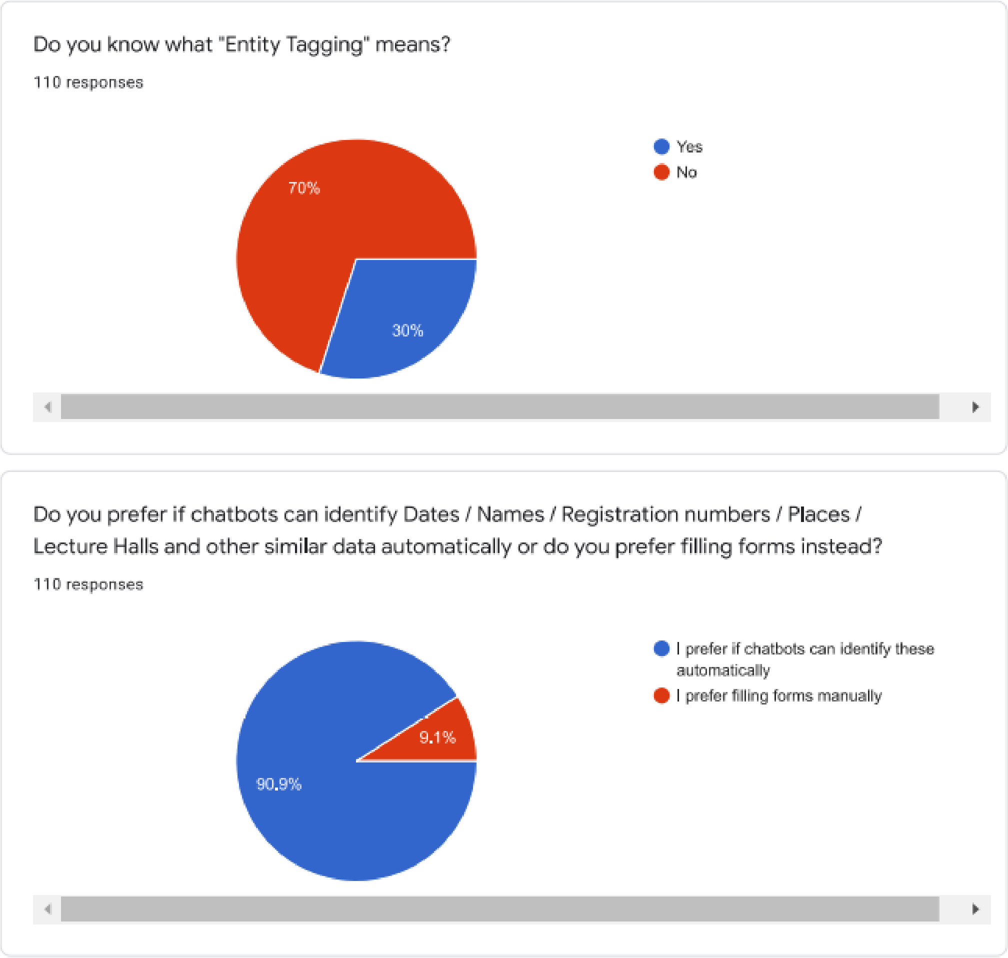
110 responses



Have you used model explainability tools like "LIME" or "SHAP" to improve machine learning model performance of NLP or any other machine learning model before? 🤔

110 responses

Figure A.8: Complete survey form questions and responses – part 8



Do you know what "Entity Tagging" means?

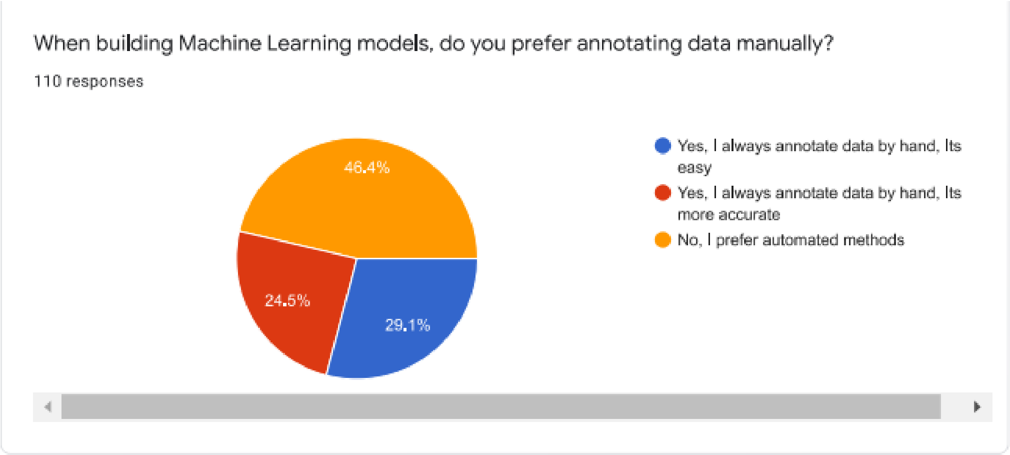
110 responses

Do you prefer if chatbots can identify Dates / Names / Registration numbers / Places / Lecture Halls and other similar data automatically or do you prefer filling forms instead?

110 responses

● I prefer if chatbots can identify these automatically
● I prefer filling forms manually
9.1%
90.9%

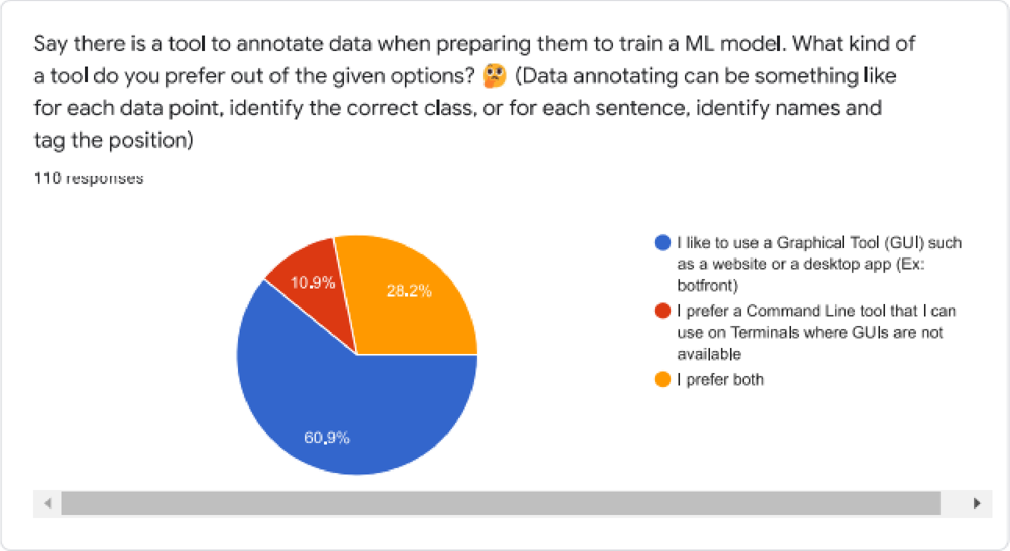
Figure A.9: Complete survey form questions and responses – part 9



When building Machine Learning models, do you prefer annotating data manually?

110 responses

46.4%
24.5%
29.1%
● Yes, I always annotate data by hand, Its easy
● Yes, I always annotate data by hand, Its more accurate
● No, I prefer automated methods

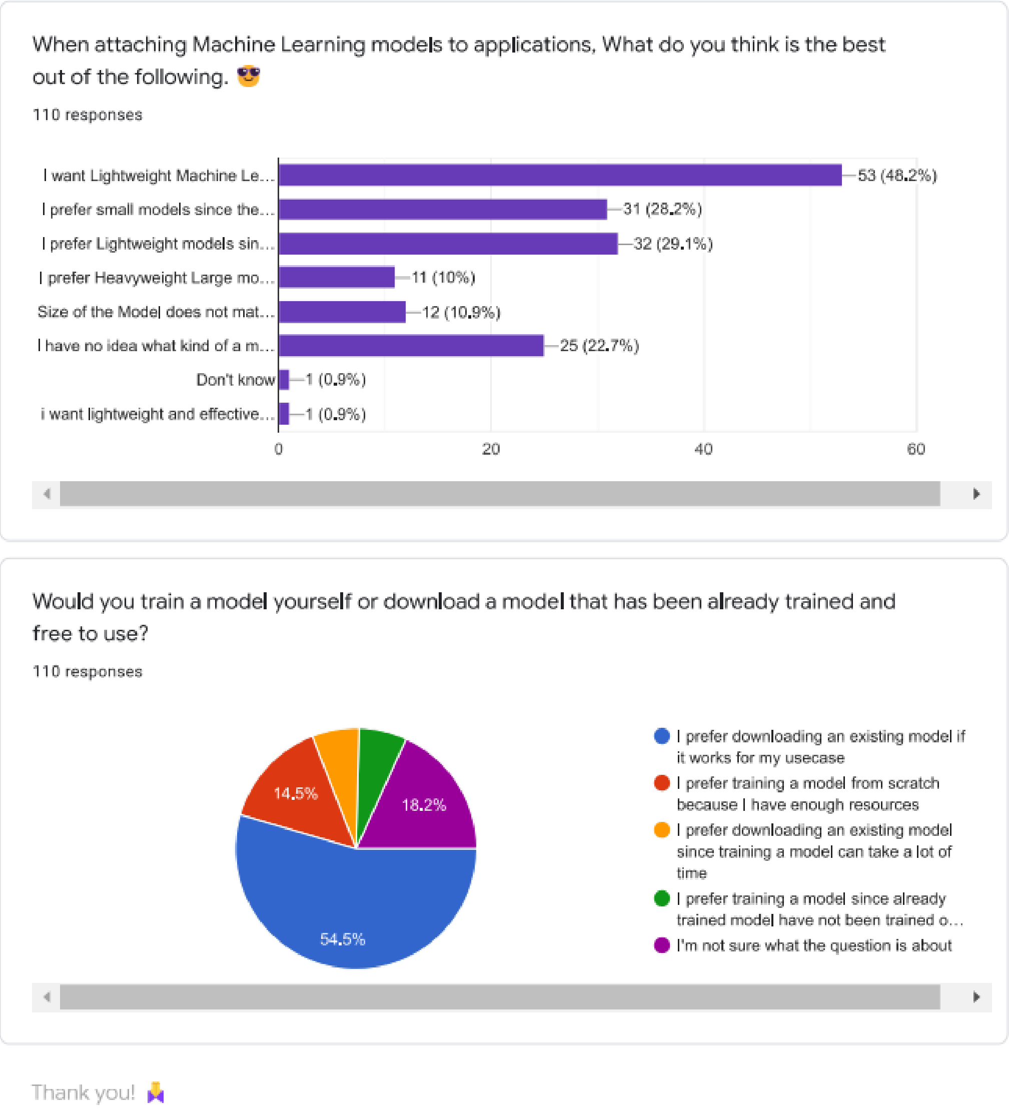


Say there is a tool to annotate data when preparing them to train a ML model. What kind of a tool do you prefer out of the given options? 🤔 (Data annotating can be something like for each data point, identify the correct class, or for each sentence, identify names and tag the position)

110 responses

10.9%
28.2%
60.9%
● I like to use a Graphical Tool (GUI) such as a website or a desktop app (Ex: botfront)
● I prefer a Command Line tool that I can use on Terminals where GUIs are not available
● I prefer both

Figure A.10: Complete survey form questions and responses – part 10



When attaching Machine Learning models to applications, What do you think is the best out of the following. 😎

110 responses

I want Lightweight Machine Le…
I prefer small models since the…
I prefer Lightweight models sin…
I prefer Heavyweight Large mo…
Size of the Model does not mat…
I have no idea what kind of a m…
Don't know
i want lightweight and effective…
53 (48.2%)
31 (28.2%)
32 (29.1%)
11 (10%)
12 (10.9%)
25 (22.7%)
1 (0.9%)
1 (0.9%)
0
20
40
60

Would you train a model yourself or download a model that has been already trained and free to use?

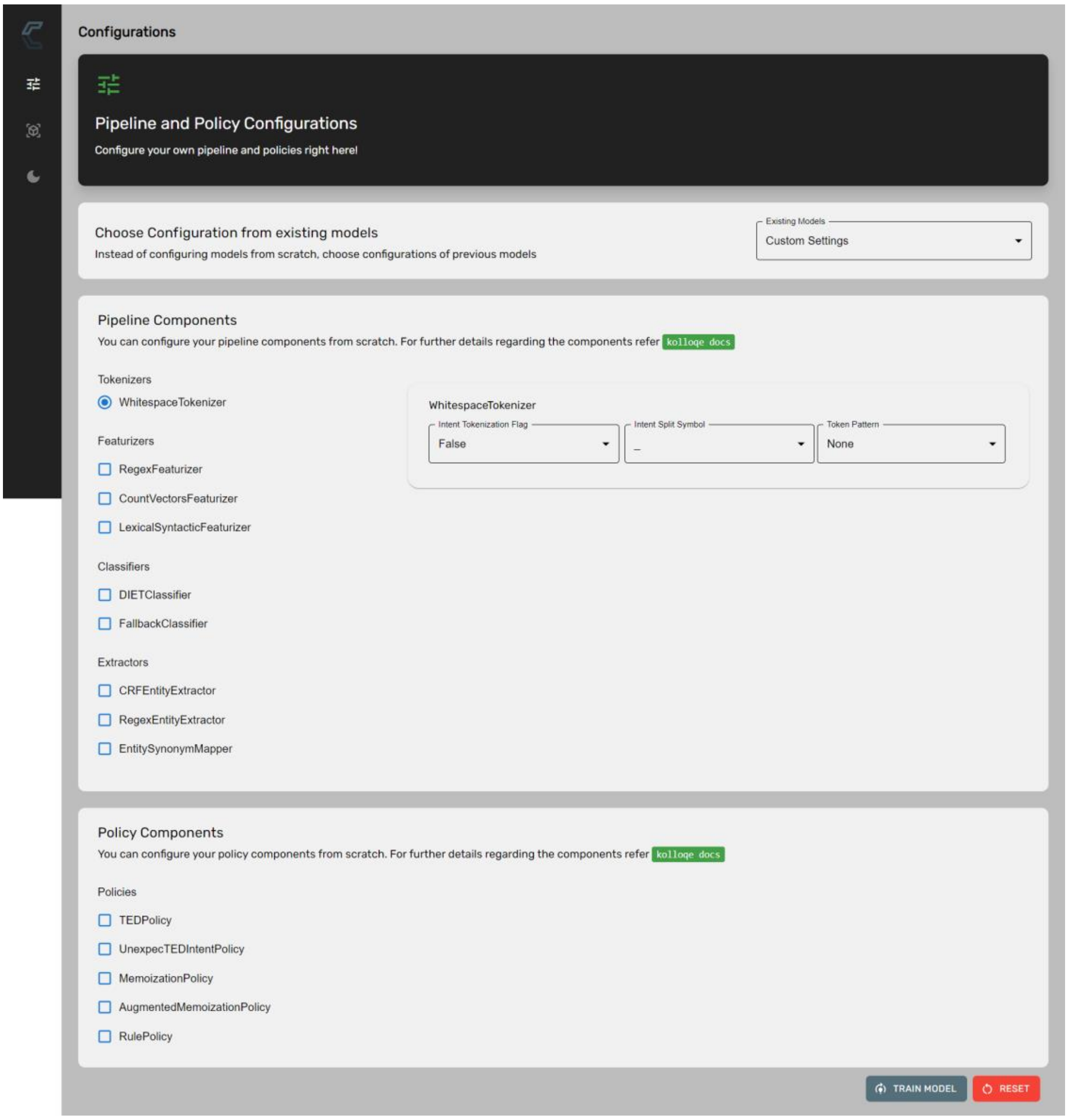
110 responses

14.5%
18.2%
54.5%
● I prefer downloading an existing model if it works for my usecase
● I prefer training a model from scratch because I have enough resources
● I prefer downloading an existing model since training a model can take a lot of time
● I prefer training a model since already trained model have not been trained o…
● I'm not sure what the question is about

Thank you! 🙇

Figure A.11: Complete survey form questions and responses – part 11

Appendix B: RASAC Server User Interfaces



Configurations

Pipeline and Policy Configurations

Configure your own pipeline and policies right here!

Choose Configuration from existing models

Instead of configuring models from scratch, choose configurations of previous models

Existing Models: Custom Settings

Pipeline Components

You can configure your pipeline components from scratch. For further details regarding the components refer `kolloqe docs`

Tokenizers

● WhitespaceTokenizer

WhitespaceTokenizer

Intent Tokenization Flag: False

Intent Split Symbol: _

Token Pattern: None

Featurizers

□ RegexFeaturizer

□ CountVectorsFeaturizer

□ LexicalSyntacticFeaturizer

Classifiers

□ DIETClassifier

□ FallbackClassifier

Extractors

□ CRFEntityExtractor

□ RegexEntityExtractor

□ EntitySynonymMapper

Policy Components

You can configure your policy components from scratch. For further details regarding the components refer `kolloqe docs`

Policies

□ TEDPolicy

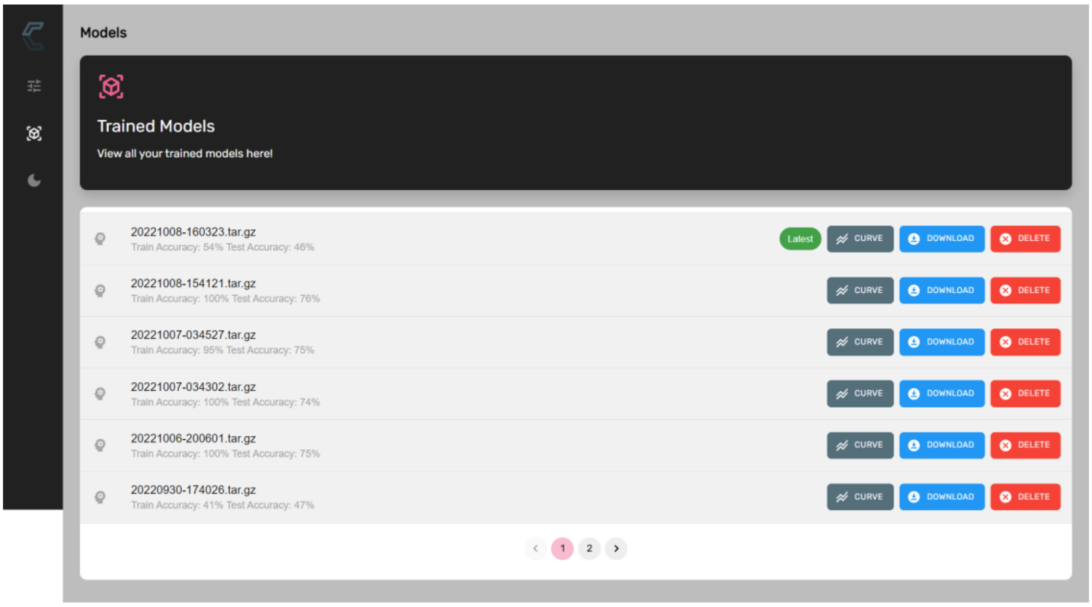
□ UnexpecTEDIntentPolicy

□ MemoizationPolicy

□ AugmentedMemoizationPolicy

□ RulePolicy

Figure B.1. RASAC Server Configurations Interface



Models

Trained Models

View all your trained models here!

20221008-160323.tar.gz
Train Accuracy: 54% Test Accuracy: 46%

20221008-154121.tar.gz
Train Accuracy: 100% Test Accuracy: 76%

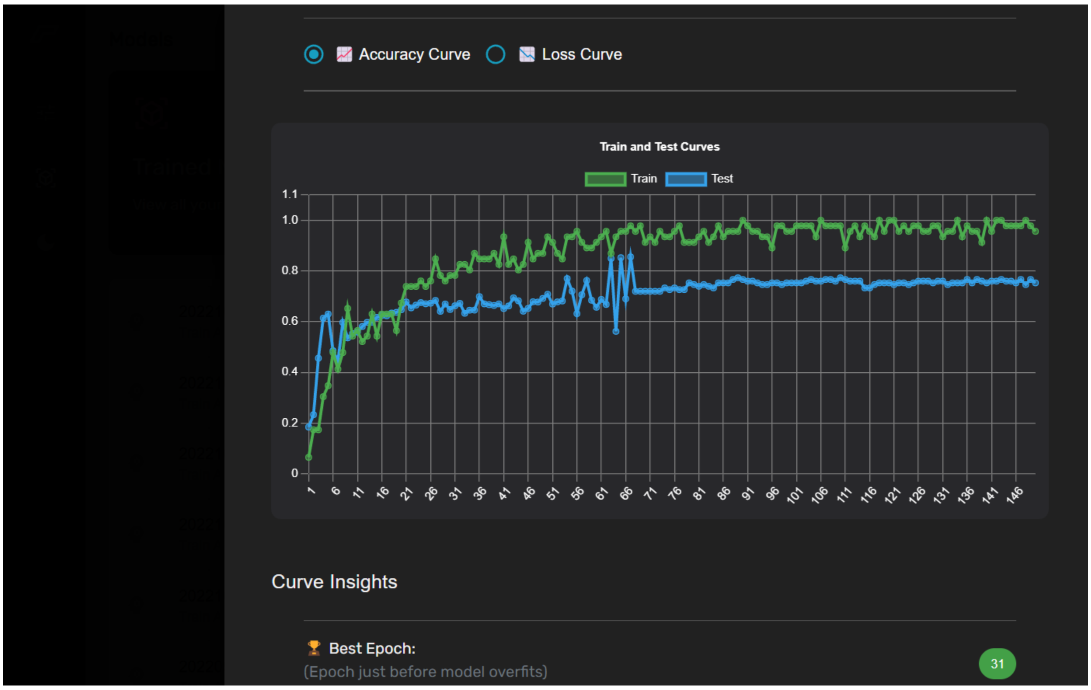
20221007-034527.tar.gz
Train Accuracy: 95% Test Accuracy: 75%

20221007-034302.tar.gz
Train Accuracy: 100% Test Accuracy: 74%

20221006-200601.tar.gz
Train Accuracy: 100% Test Accuracy: 75%

20220930-174026.tar.gz
Train Accuracy: 41% Test Accuracy: 47%

Figure B.2. RASAC Server Configurations Interface – List of Models



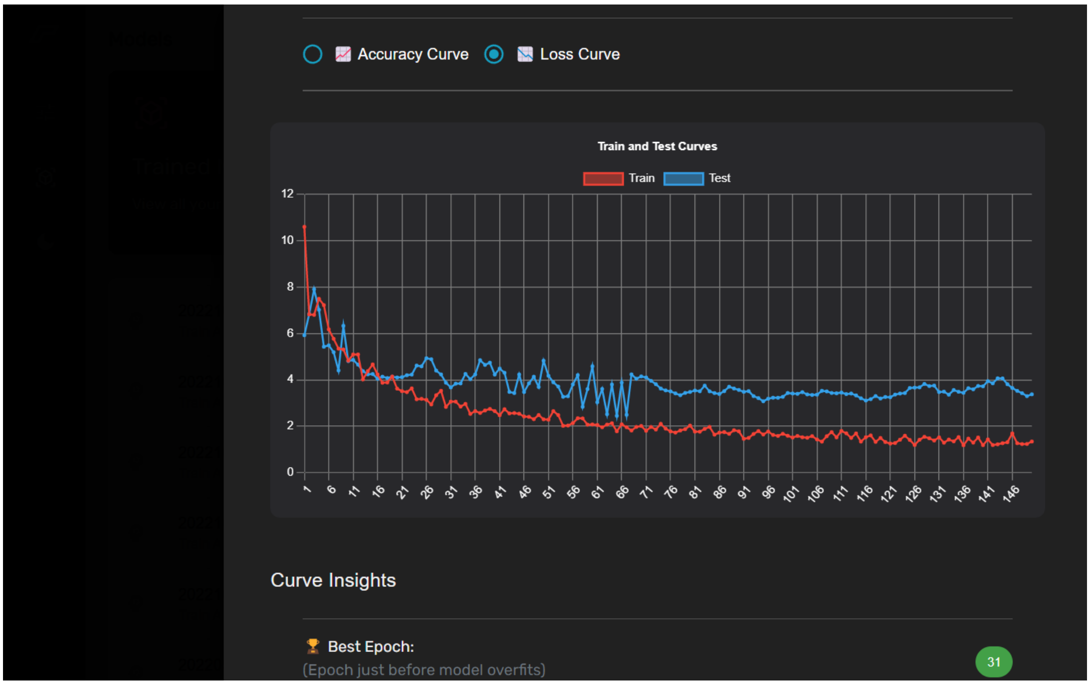
● Accuracy Curve ○ Loss Curve

Curve Insights

🏆 Best Epoch:
(Epoch just before model overfits)

31

Figure B.3. View of the Accuracy Curve of a Model



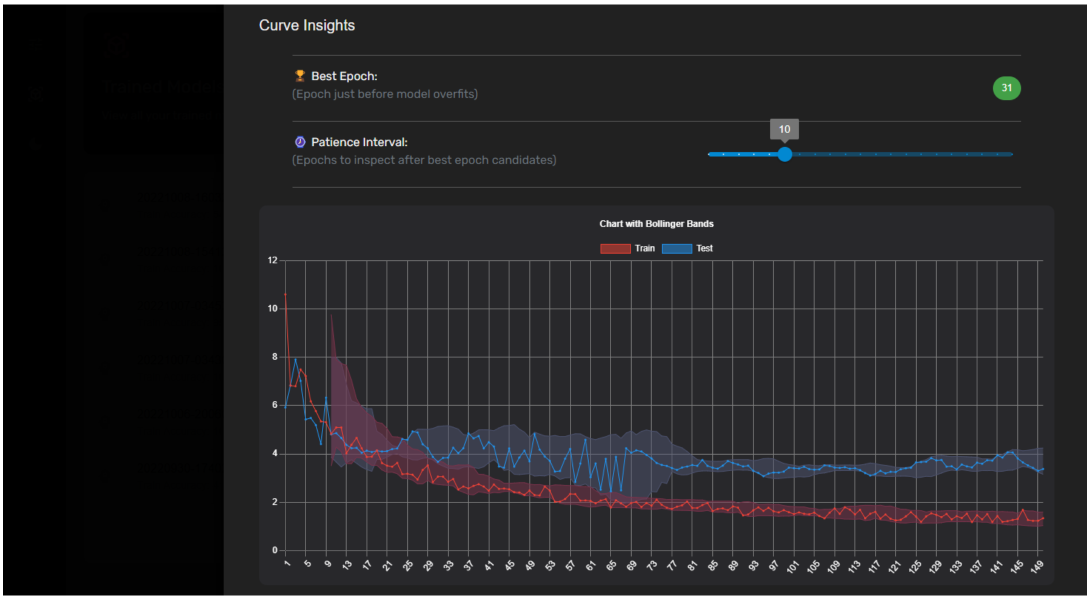
○ Accuracy Curve ● Loss Curve

Curve Insights

Best Epoch:
(Epoch just before model overfits)

31

Figure B.4. View of the Loss Curve of a Model



Curve Insights

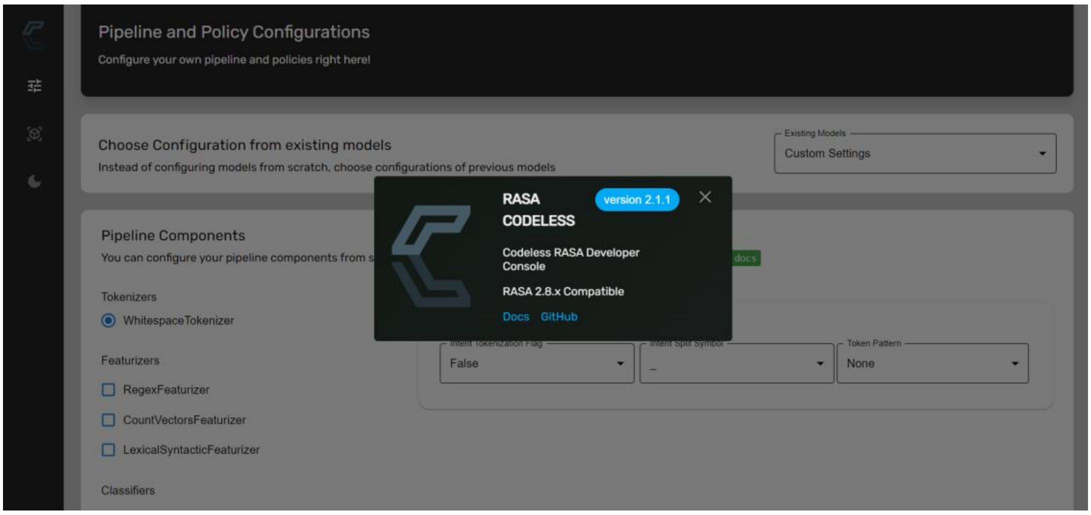
Best Epoch:
(Epoch just before model overfits)

31

Patience Interval:
(Epochs to inspect after best epoch candidates)

10

Figure B.5. View of the Best Epoch Suggested of a Model



Pipeline and Policy Configurations

Configure your own pipeline and policies right here!

Choose Configuration from existing models

Instead of configuring models from scratch, choose configurations of previous models

Existing Models: Custom Settings

RASA CODELESS version 2.1.1

Codeless RASA Developer Console

RASA 2.8.x Compatible

Pipeline Components

You can configure your pipeline components from s

Tokenizers

● WhitespaceTokenizer

Featurizers

□ RegexFeaturizer

□ CountVectorsFeaturizer

□ LexicalSyntacticFeaturizer

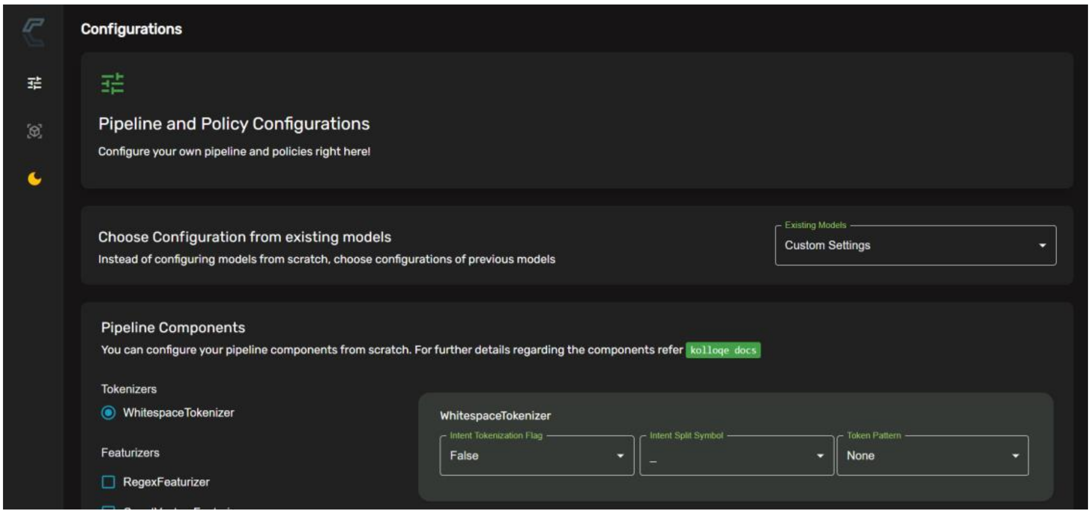
Classifiers

Intent Tokenization Flag: False

Intent Split Symbol: _

Token Pattern: None

Figure B.6. RASAC Server Version Overlay



Configurations

Pipeline and Policy Configurations

Configure your own pipeline and policies right here!

Choose Configuration from existing models

Instead of configuring models from scratch, choose configurations of previous models

Existing Models: Custom Settings

Pipeline Components

You can configure your pipeline components from scratch. For further details regarding the components refer kolloqe docs

Tokenizers

● WhitespaceTokenizer

Featurizers

□ RegexFeaturizer

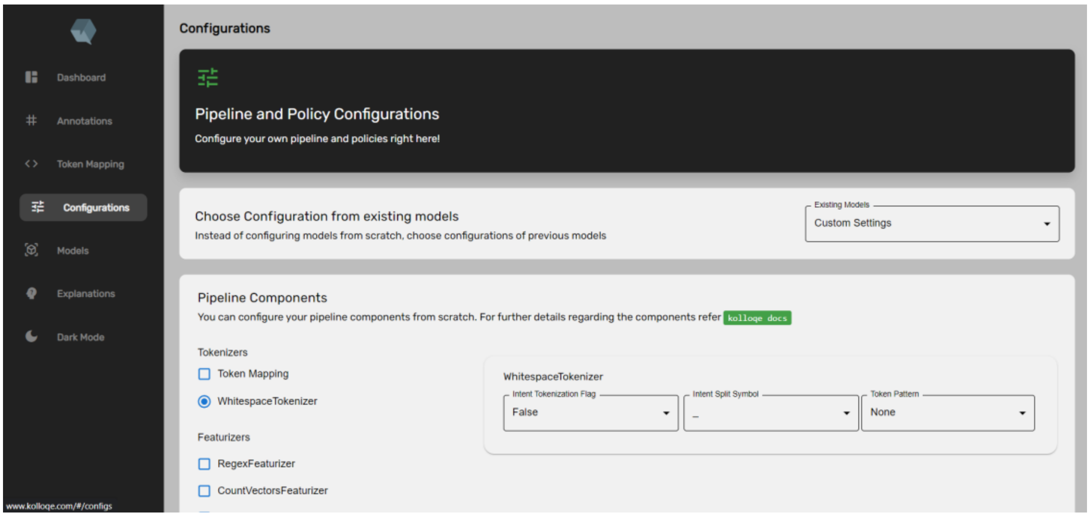
WhitespaceTokenizer

Intent Tokenization Flag: False

Intent Split Symbol: _

Token Pattern: None

Figure B.7. Kolloqe Server Dark Mode



Configurations

Pipeline and Policy Configurations

Configure your own pipeline and policies right here!

Choose Configuration from existing models

Instead of configuring models from scratch, choose configurations of previous models

Existing Models: Custom Settings

Pipeline Components

You can configure your pipeline components from scratch. For further details regarding the components refer kolloqe docs

Tokenizers

□ Token Mapping

● WhitespaceTokenizer

WhitespaceTokenizer

Intent Tokenization Flag: False

Intent Split Symbol: _

Token Pattern: None

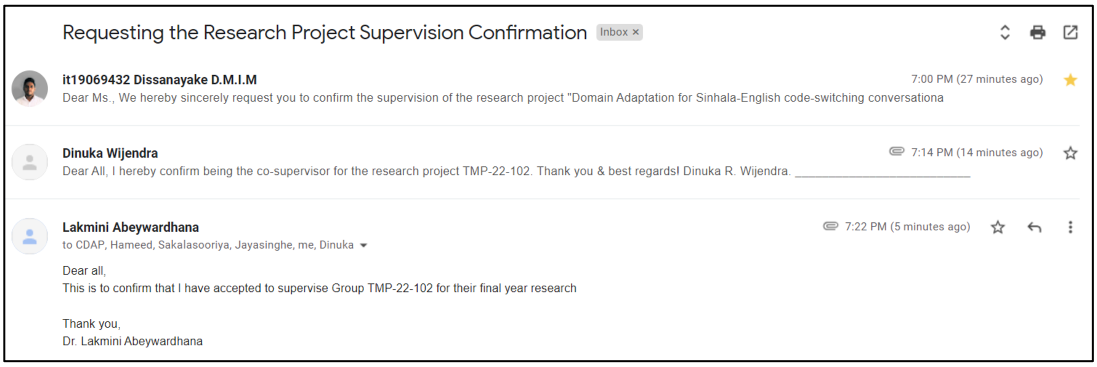
Featurizers

□ RegexFeaturizer

□ CountVectorsFeaturizer

Figure B.8. RASAC UI integrated into Kolloqe Developer Console

Appendix C: Supervision Confirmation Emails



Requesting the Research Project Supervision Confirmation

it19069432 Dissanayake D.M.I.M 7:00 PM (27 minutes ago)

Dear Ms., We hereby sincerely request you to confirm the supervision of the research project "Domain Adaptation for Sinhala-English code-switching conversationa

Dinuka Wijendra 7:14 PM (14 minutes ago)

Dear All, I hereby confirm being the co-supervisor for the research project TMP-22-102. Thank you & best regards! Dinuka R. Wijendra. ______________________________

Lakmini Abeywardhana 7:22 PM (5 minutes ago)

to CDAP, Hameed, Sakalasooriya, Jayasinghe, me, Dinuka

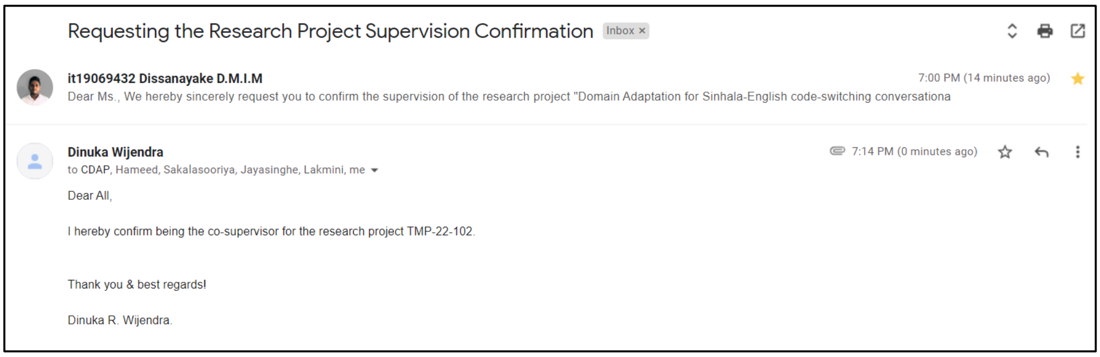
Dear all,

This is to confirm that I have accepted to supervise Group TMP-22-102 for their final year research

Thank you,

Dr. Lakmini Abeywardhana

Figure B.9: Research project supervision confirmation email



Requesting the Research Project Supervision Confirmation

it19069432 Dissanayake D.M.I.M 7:00 PM (14 minutes ago)

Dear Ms., We hereby sincerely request you to confirm the supervision of the research project "Domain Adaptation for Sinhala-English code-switching conversationa

Dinuka Wijendra 7:14 PM (0 minutes ago)

to CDAP, Hameed, Sakalasooriya, Jayasinghe, Lakmini, me

Dear All,

I hereby confirm being the co-supervisor for the research project TMP-22-102.

Thank you & best regards!

Dinuka R. Wijendra.

Figure B. 10: Research project co-supervision confirmation email