Building Virtual Assistants with Rasa

Share & Learn Seminar

Yi Li

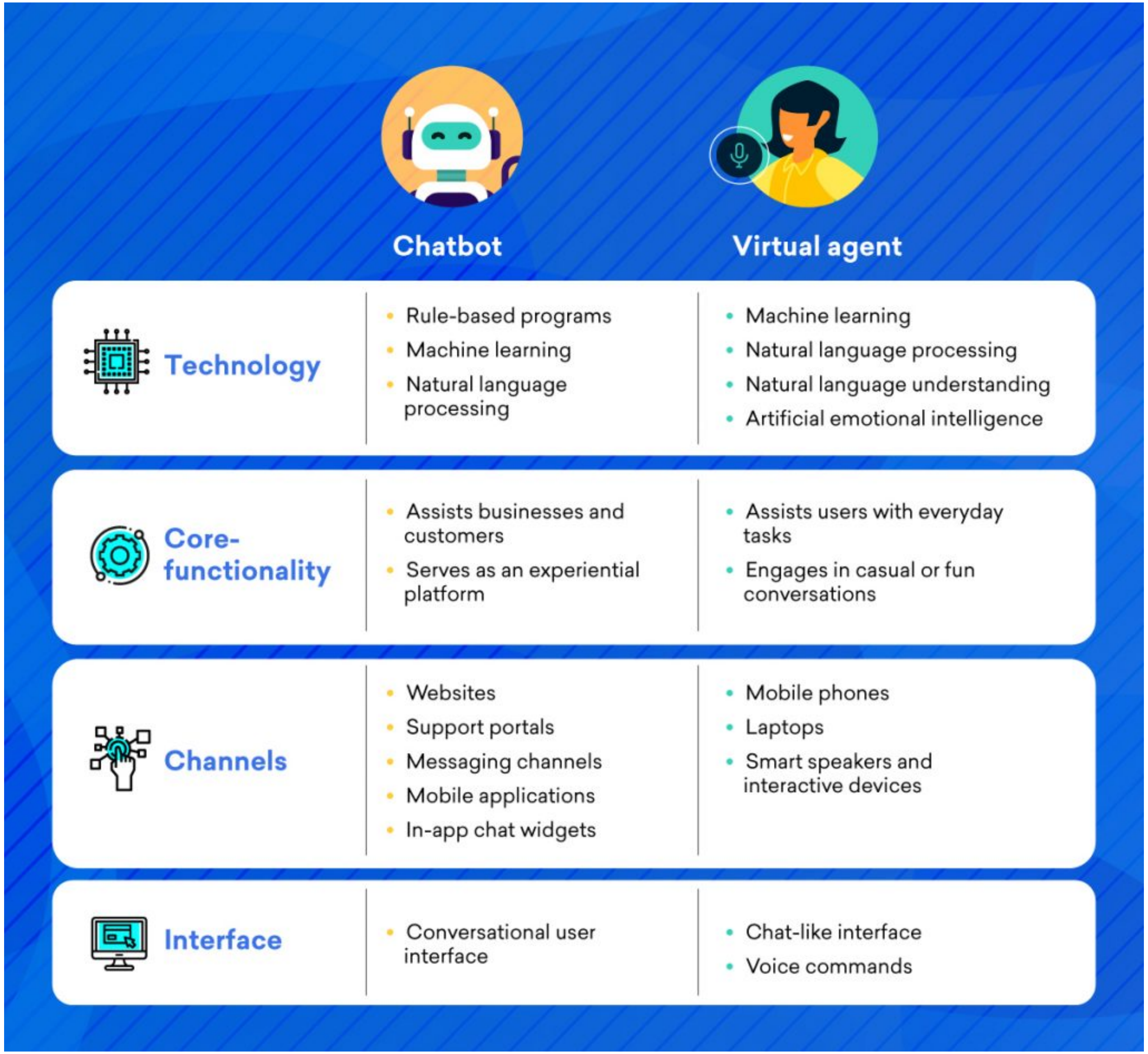
07/08/2022

Virtual Assistants vs Chatbots: What's the Difference

In general, a virtual assistant includes the functions of a chatbot, but also support voice rather than only text.

Three main technologies of a virtual assistant:

- Automatic Speech Recognition (ASR)
- Text To Speech (TTS)
- Natural Language Understanding (NLU)



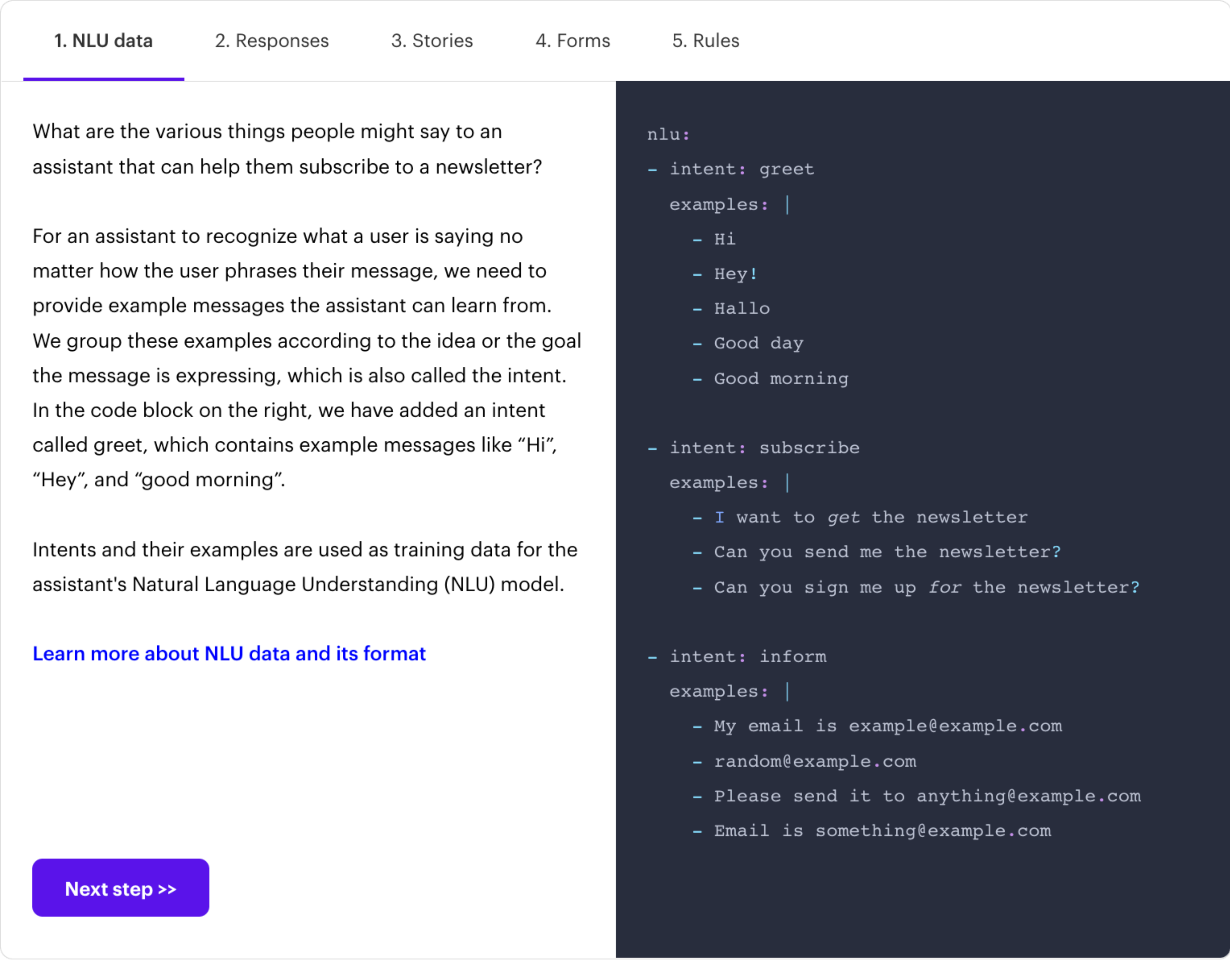
	Chatbot	Virtual agent
Technology	• Rule-based programs • Machine learning • Natural language processing	• Machine learning • Natural language processing • Natural language understanding • Artificial emotional intelligence
Core-functionality	• Assists businesses and customers • Serves as an experiential platform	• Assists users with everyday tasks • Engages in casual or fun conversations
Channels	• Websites • Support portals • Messaging channels • Mobile applications • In-app chat widgets	• Mobile phones • Laptops • Smart speakers and interactive devices
Interface	• Conversational user interface	• Chat-like interface • Voice commands

https://freshdesk.com/customer-engagement/virtual-assistant-chatbot-blog/

Rasa Open Source

Rasa is an open source machine learning framework for automated text and voice-based conversations. Understand messages, hold conversations, and connect to messaging channels and APIs.

Building Virtual Assistants - Basics



What are the various things people might say to an assistant that can help them subscribe to a newsletter?

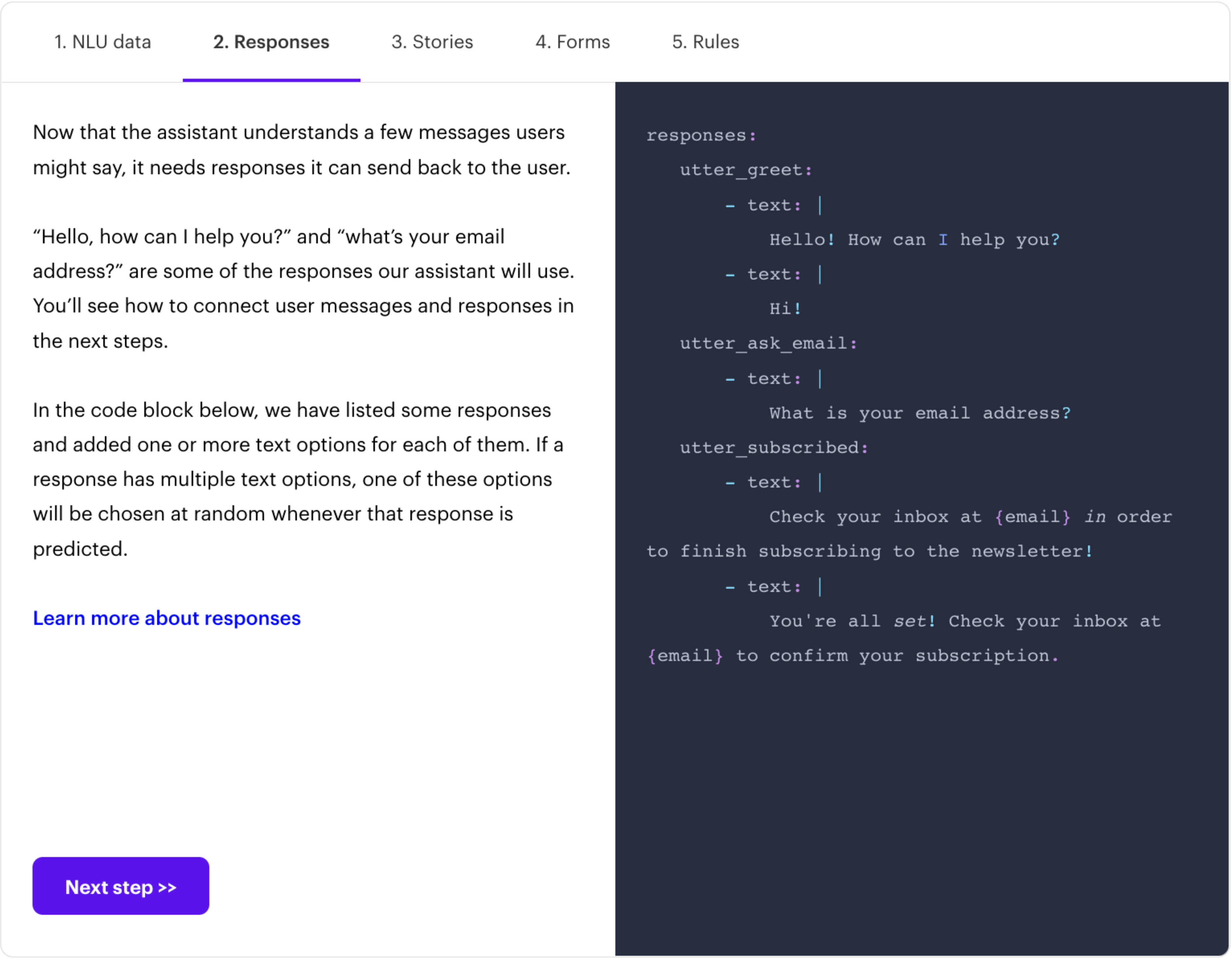
For an assistant to recognize what a user is saying no matter how the user phrases their message, we need to provide example messages the assistant can learn from. We group these examples according to the idea or the goal the message is expressing, which is also called the intent. In the code block on the right, we have added an intent called greet, which contains example messages like “Hi”, “Hey”, and “good morning”.

Intents and their examples are used as training data for the assistant's Natural Language Understanding (NLU) model.

Learn more about NLU data and its format

```
nlu:
- intent: greet
  examples: |
    - Hi
    - Hey!
    - Hallo
    - Good day
    - Good morning

- intent: subscribe
  examples: |
    - I want to get the newsletter
    - Can you send me the newsletter?
    - Can you sign me up for the newsletter?

- intent: inform
  examples: |
    - My email is example@example.com
    - random@example.com
    - Please send it to anything@example.com
    - Email is something@example.com
```

Building Virtual Assistants - Basics



Now that the assistant understands a few messages users might say, it needs responses it can send back to the user.

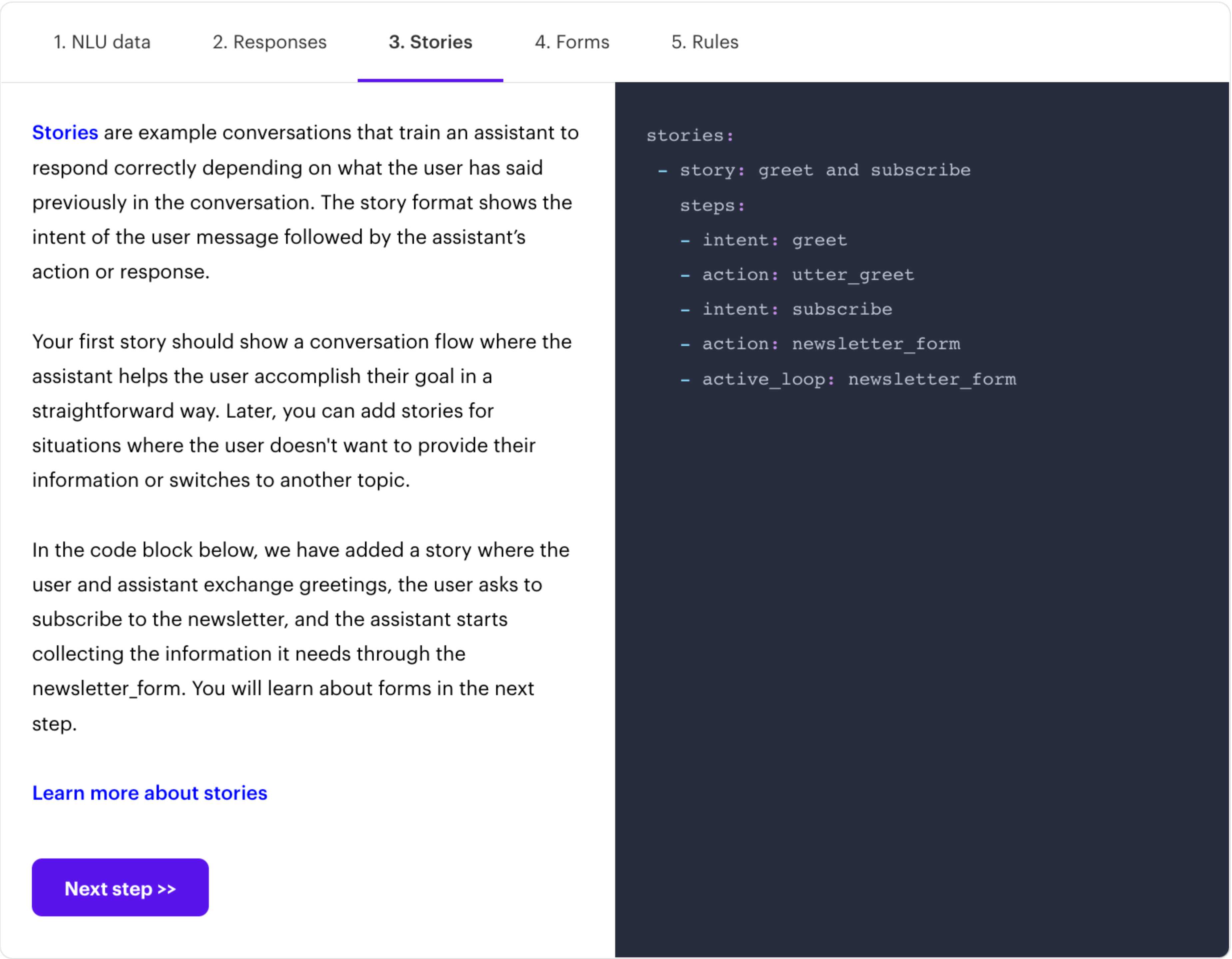
“Hello, how can I help you?” and “what’s your email address?” are some of the responses our assistant will use. You’ll see how to connect user messages and responses in the next steps.

In the code block below, we have listed some responses and added one or more text options for each of them. If a response has multiple text options, one of these options will be chosen at random whenever that response is predicted.

Learn more about responses

```
responses:
   utter_greet:
      - text: |
          Hello! How can I help you?
      - text: |
          Hi!
   utter_ask_email:
      - text: |
          What is your email address?
   utter_subscribed:
      - text: |
          Check your inbox at {email} in order
to finish subscribing to the newsletter!
      - text: |
          You're all set! Check your inbox at
{email} to confirm your subscription.
```

Building Virtual Assistants - Basics



Stories are example conversations that train an assistant to respond correctly depending on what the user has said previously in the conversation. The story format shows the intent of the user message followed by the assistant's action or response.

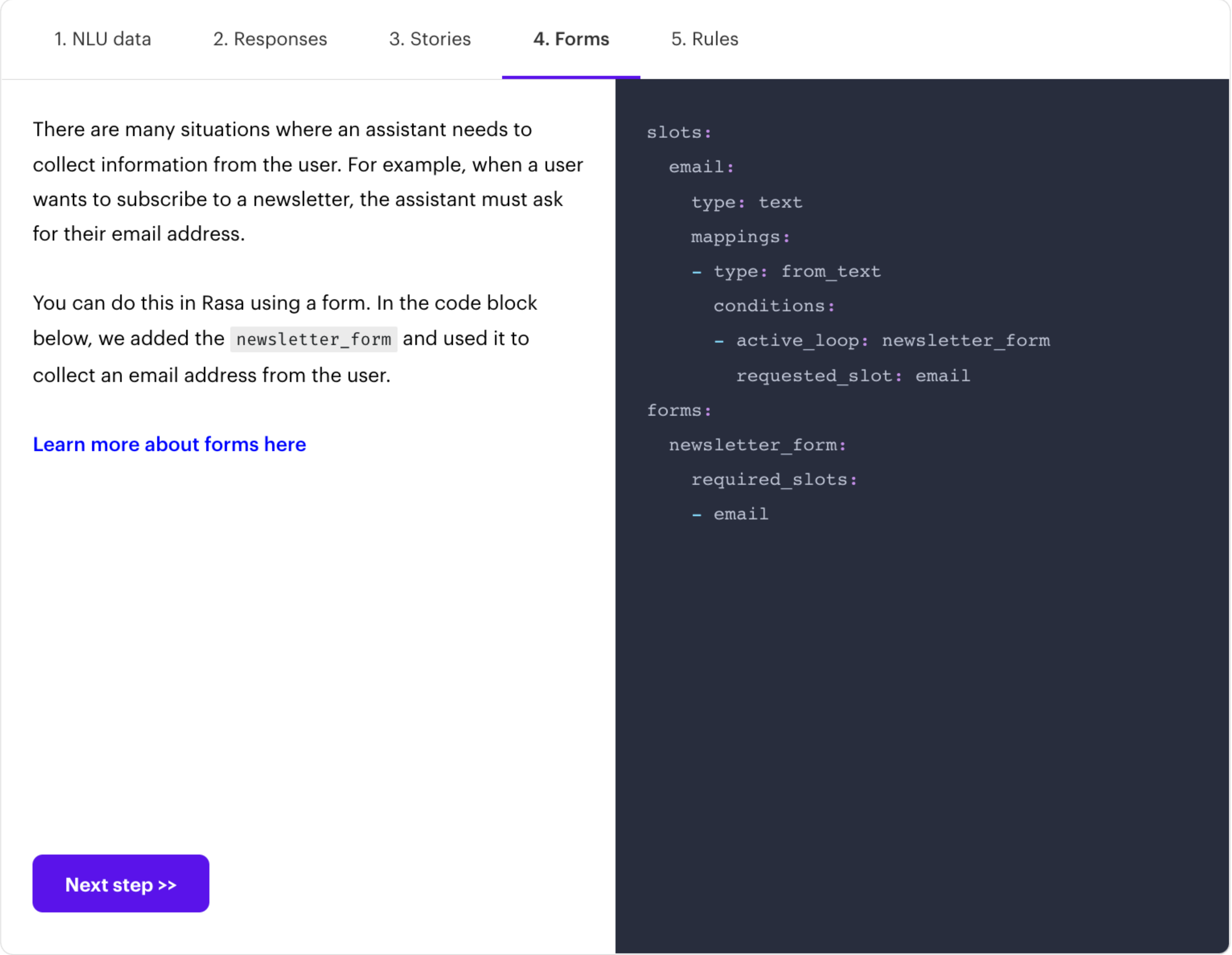
Your first story should show a conversation flow where the assistant helps the user accomplish their goal in a straightforward way. Later, you can add stories for situations where the user doesn't want to provide their information or switches to another topic.

In the code block below, we have added a story where the user and assistant exchange greetings, the user asks to subscribe to the newsletter, and the assistant starts collecting the information it needs through the newsletter_form. You will learn about forms in the next step.

Learn more about stories

```
stories:
 - story: greet and subscribe
   steps:
   - intent: greet
   - action: utter_greet
   - intent: subscribe
   - action: newsletter_form
   - active_loop: newsletter_form
```

Building Virtual Assistants - Basics



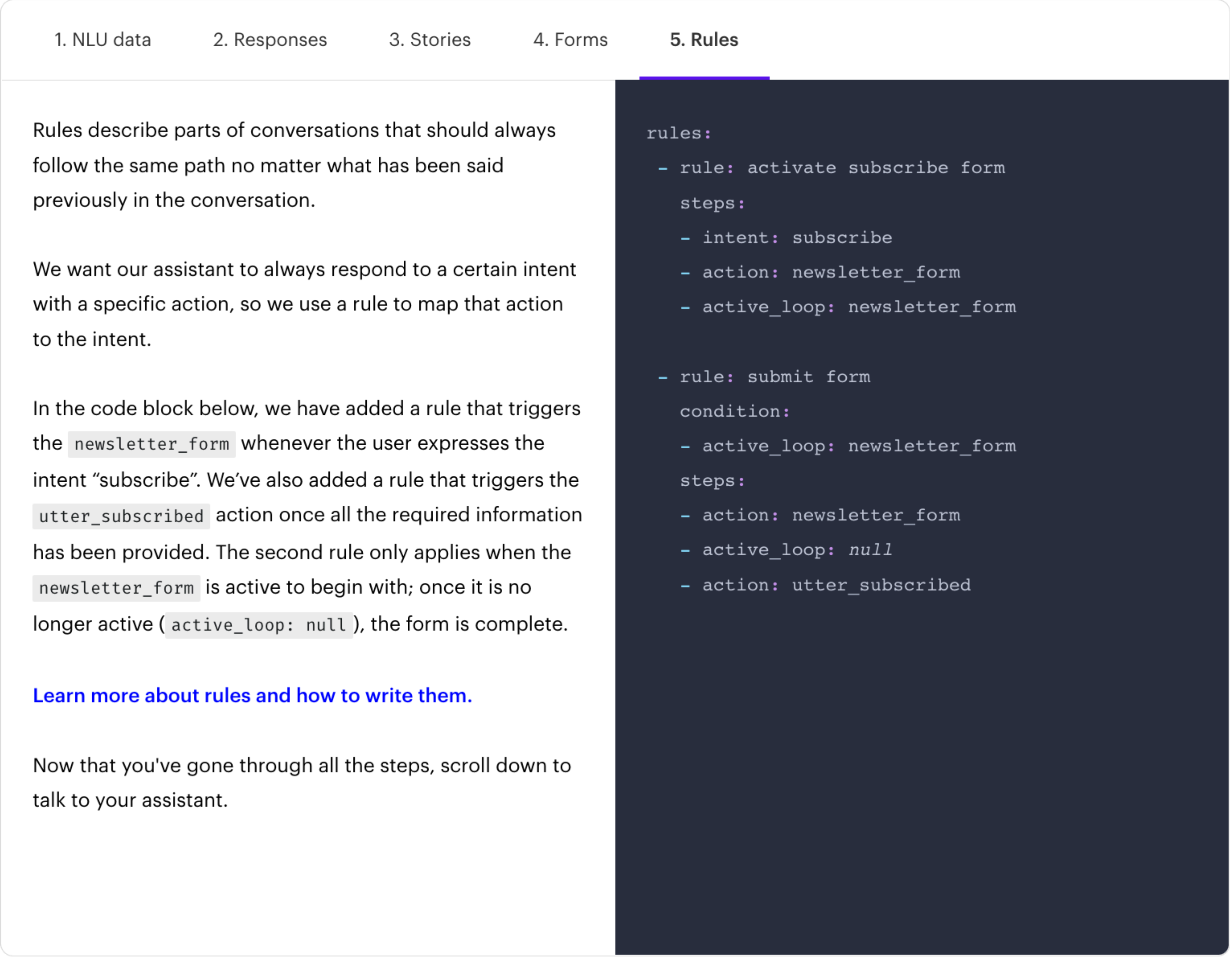
There are many situations where an assistant needs to collect information from the user. For example, when a user wants to subscribe to a newsletter, the assistant must ask for their email address.

You can do this in Rasa using a form. In the code block below, we added the `newsletter_form` and used it to collect an email address from the user.

Learn more about forms here

```
slots:
  email:
    type: text
    mappings:
    - type: from_text
      conditions:
      - active_loop: newsletter_form
        requested_slot: email
forms:
  newsletter_form:
    required_slots:
    - email
```

Building Virtual Assistants - Basics



Rules describe parts of conversations that should always follow the same path no matter what has been said previously in the conversation.

We want our assistant to always respond to a certain intent with a specific action, so we use a rule to map that action to the intent.

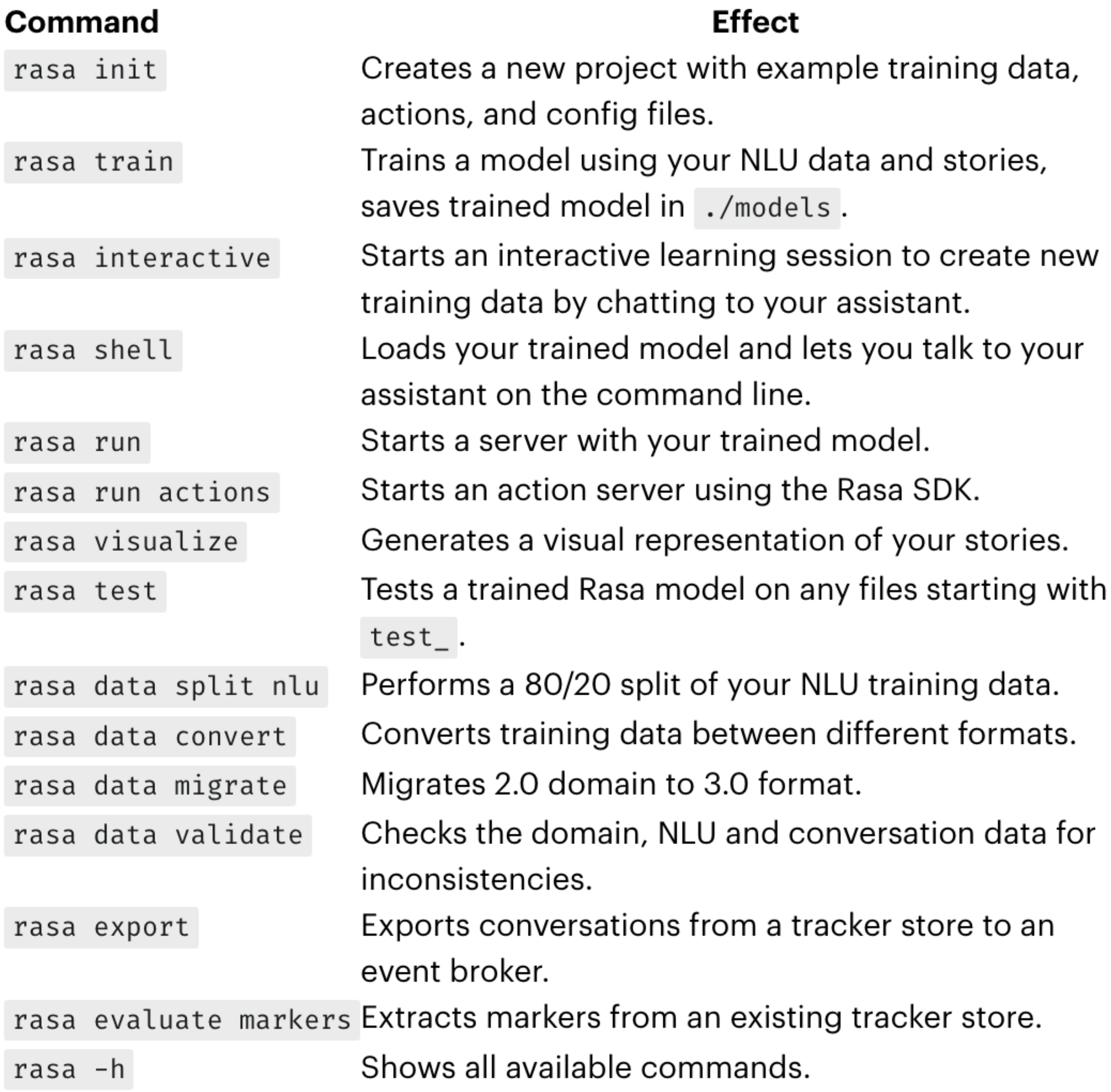
In the code block below, we have added a rule that triggers the `newsletter_form` whenever the user expresses the intent “subscribe”. We’ve also added a rule that triggers the `utter_subscribed` action once all the required information has been provided. The second rule only applies when the `newsletter_form` is active to begin with; once it is no longer active (`active_loop: null`), the form is complete.

Learn more about rules and how to write them.

Now that you've gone through all the steps, scroll down to talk to your assistant.

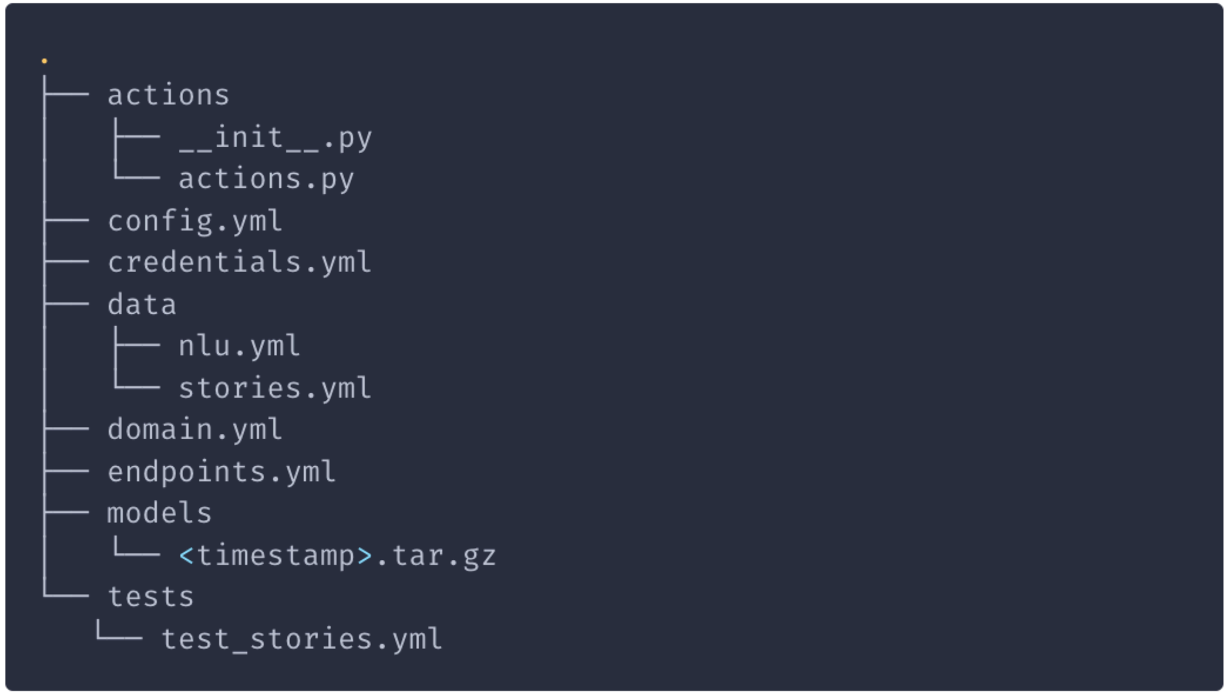
```
rules:
 - rule: activate subscribe form
   steps:
   - intent: subscribe
   - action: newsletter_form
   - active_loop: newsletter_form

 - rule: submit form
   condition:
   - active_loop: newsletter_form
   steps:
   - action: newsletter_form
   - active_loop: null
   - action: utter_subscribed
```

Cheat Sheet



Command	Effect
`rasa init`	Creates a new project with example training data, actions, and config files.
`rasa train`	Trains a model using your NLU data and stories, saves trained model in `./models`.
`rasa interactive`	Starts an interactive learning session to create new training data by chatting to your assistant.
`rasa shell`	Loads your trained model and lets you talk to your assistant on the command line.
`rasa run`	Starts a server with your trained model.
`rasa run actions`	Starts an action server using the Rasa SDK.
`rasa visualize`	Generates a visual representation of your stories.
`rasa test`	Tests a trained Rasa model on any files starting with `test_`.
`rasa data split nlu`	Performs a 80/20 split of your NLU training data.
`rasa data convert`	Converts training data between different formats.
`rasa data migrate`	Migrates 2.0 domain to 3.0 format.
`rasa data validate`	Checks the domain, NLU and conversation data for inconsistencies.
`rasa export`	Exports conversations from a tracker store to an event broker.
`rasa evaluate markers`	Extracts markers from an existing tracker store.
`rasa -h`	Shows all available commands.

Basic Folder Structures of Scripts



```
.
├── actions
│   ├── __init__.py
│   └── actions.py
├── config.yml
├── credentials.yml
├── data
│   ├── nlu.yml
│   └── stories.yml
├── domain.yml
├── endpoints.yml
├── models
│   └── <timestamp>.tar.gz
└── tests
   └── test_stories.yml
```

Related GitHub Repos

https://github.com/RasaHQ/rasa
https://github.com/RasaHQ/rasa-sdk
https://github.com/RasaHQ/rasa-demo
https://github.com/RasaHQ/helm-charts
https://github.com/RasaHQ/rasa-x-helm

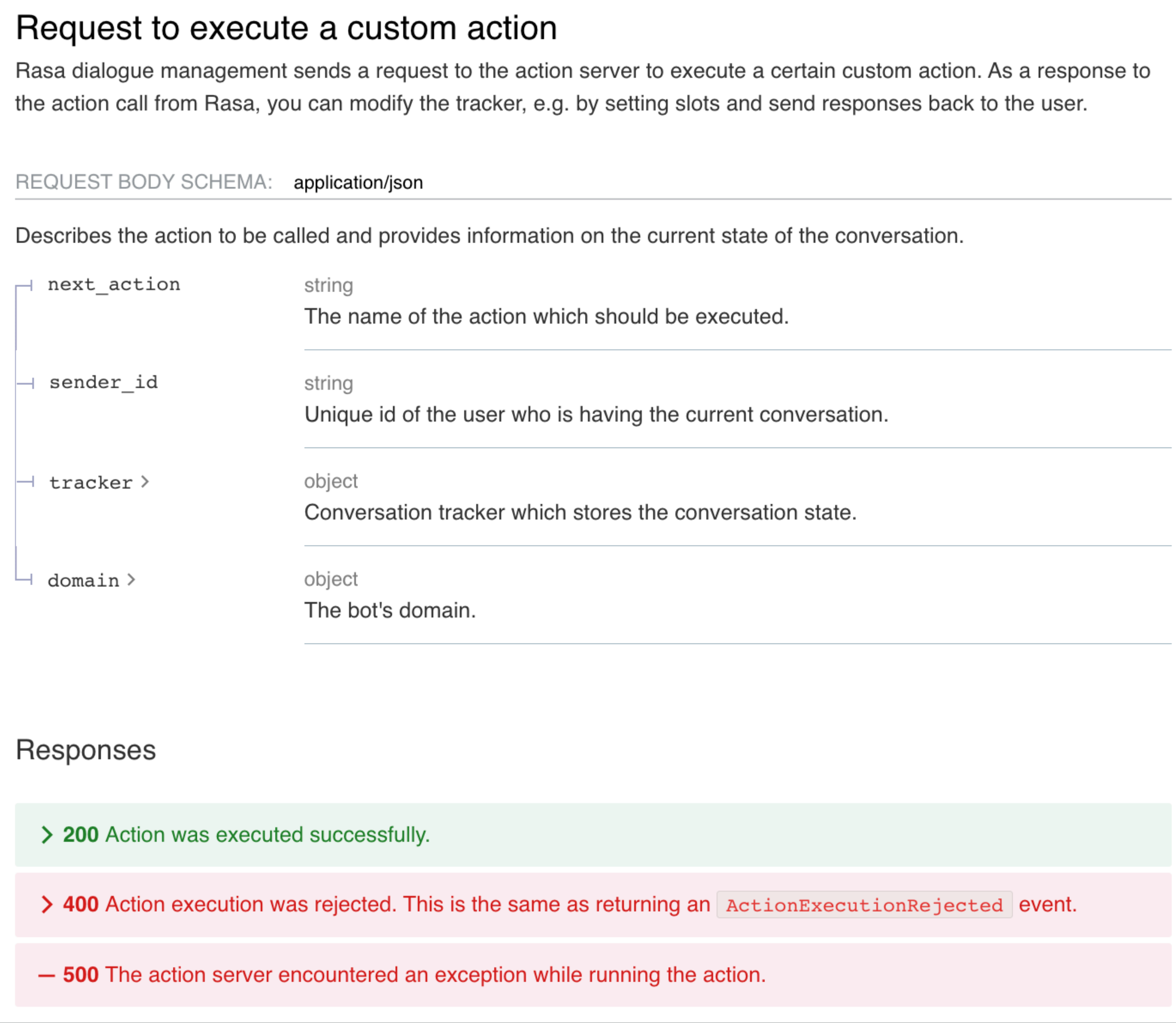
Rasa Action Server

A Rasa action server runs custom actions (e.g. API calls, database queries, etc.) for a Rasa Open Source conversational assistant.

How it works

- When your assistant predicts a custom action, the Rasa server sends a POST request to the action server with a json payload including the name of the predicted action, the conversation ID, the contents of the tracker and the contents of the domain.
- When the action server finishes running a custom action, it returns a json payload of responses and events.

https://rasa.com/docs/action-server/



Request to execute a custom action

Rasa dialogue management sends a request to the action server to execute a certain custom action. As a response to the action call from Rasa, you can modify the tracker, e.g. by setting slots and send responses back to the user.

REQUEST BODY SCHEMA: application/json

Describes the action to be called and provides information on the current state of the conversation.

`next_action`	string	The name of the action which should be executed.
`sender_id`	string	Unique id of the user who is having the current conversation.
`tracker` ›	object	Conversation tracker which stores the conversation state.
`domain` ›	object	The bot's domain.

Responses

› **200** Action was executed successfully.

› **400** Action execution was rejected. This is the same as returning an `ActionExecutionRejected` event.

— **500** The action server encountered an exception while running the action.

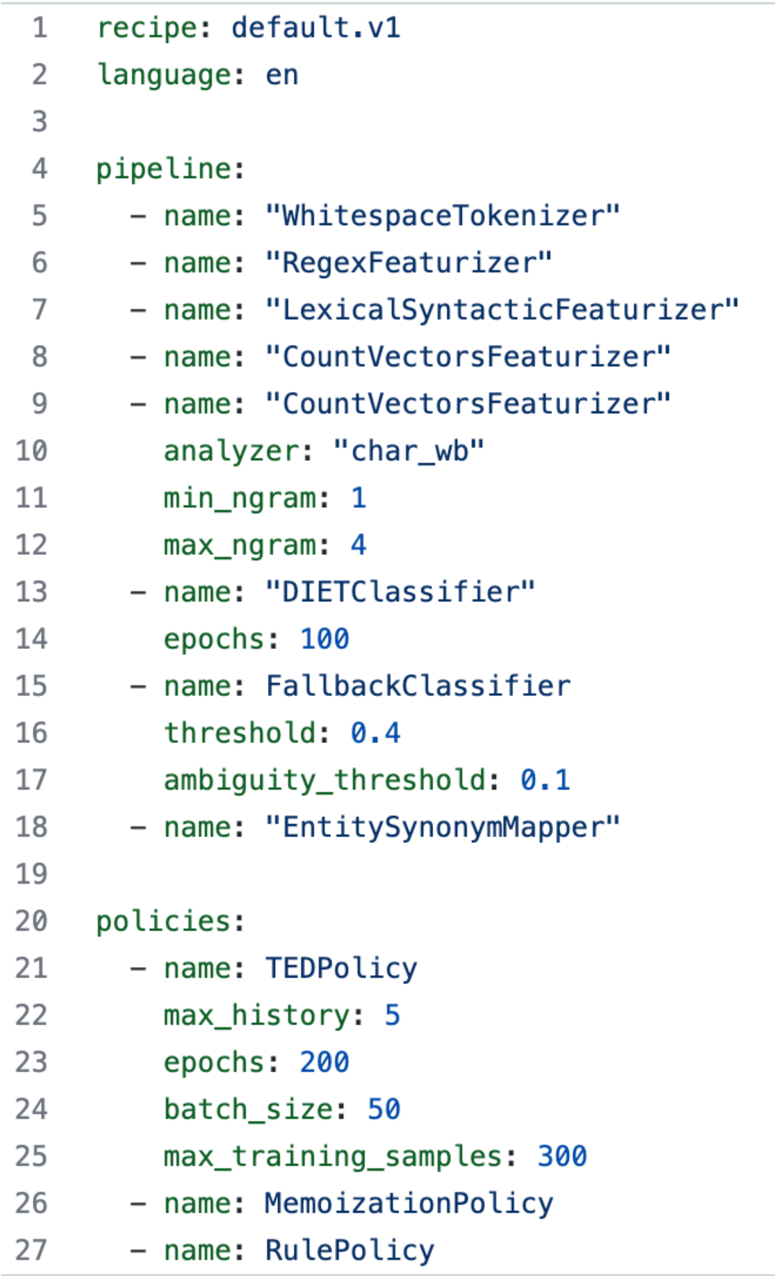
Model Configuration – NLU Pipeline

Supported components for building NLU model pipelines:

- Language Models
  - MitieNLP
  - SpacyNLP
- Tokenizers
  - WhitespaceTokenizer
  - JiebaTokenizer
  - MitieTokenizer
  - SpacyTokenizer
- Featurizers
  - MitieFeaturizer
  - SpacyFeaturizer
  - ConveRTFeaturizer
  - LanguageModelFeaturizer
  - RegexFeaturizer
  - CountVectorsFeaturizer
  - LexicalSyntacticFeaturizer
- Intent Classifiers
  - MitieIntentClassifier
  - LogisticRegressionClassifier
  - SklearnIntentClassifier
  - KeywordIntentClassifier
  - FallbackClassifier
- Entity Extractors
  - MitieEntityExtractor
  - SpacyEntityExtractor
  - CRFEntityExtractor
  - DucklingEntityExtractor
  - RegexEntityExtractor
  - EntitySynonymMapper
- Combined Intent Classifiers and Entity Extractors
  - DIETClassifier (https://arxiv.org/pdf/2004.09936)
- Selectors
  - ResponseSelector

https://rasa.com/docs/rasa/components

config.yml example:



```
recipe: default.v1
language: en

pipeline:
  - name: "WhitespaceTokenizer"
  - name: "RegexFeaturizer"
  - name: "LexicalSyntacticFeaturizer"
  - name: "CountVectorsFeaturizer"
  - name: "CountVectorsFeaturizer"
    analyzer: "char_wb"
    min_ngram: 1
    max_ngram: 4
  - name: "DIETClassifier"
    epochs: 100
  - name: FallbackClassifier
    threshold: 0.4
    ambiguity_threshold: 0.1
  - name: "EntitySynonymMapper"

policies:
  - name: TEDPolicy
    max_history: 5
    epochs: 200
    batch_size: 50
    max_training_samples: 300
  - name: MemoizationPolicy
  - name: RulePolicy
```

Dual Intent and Entity Transformer (DIET) Classifier

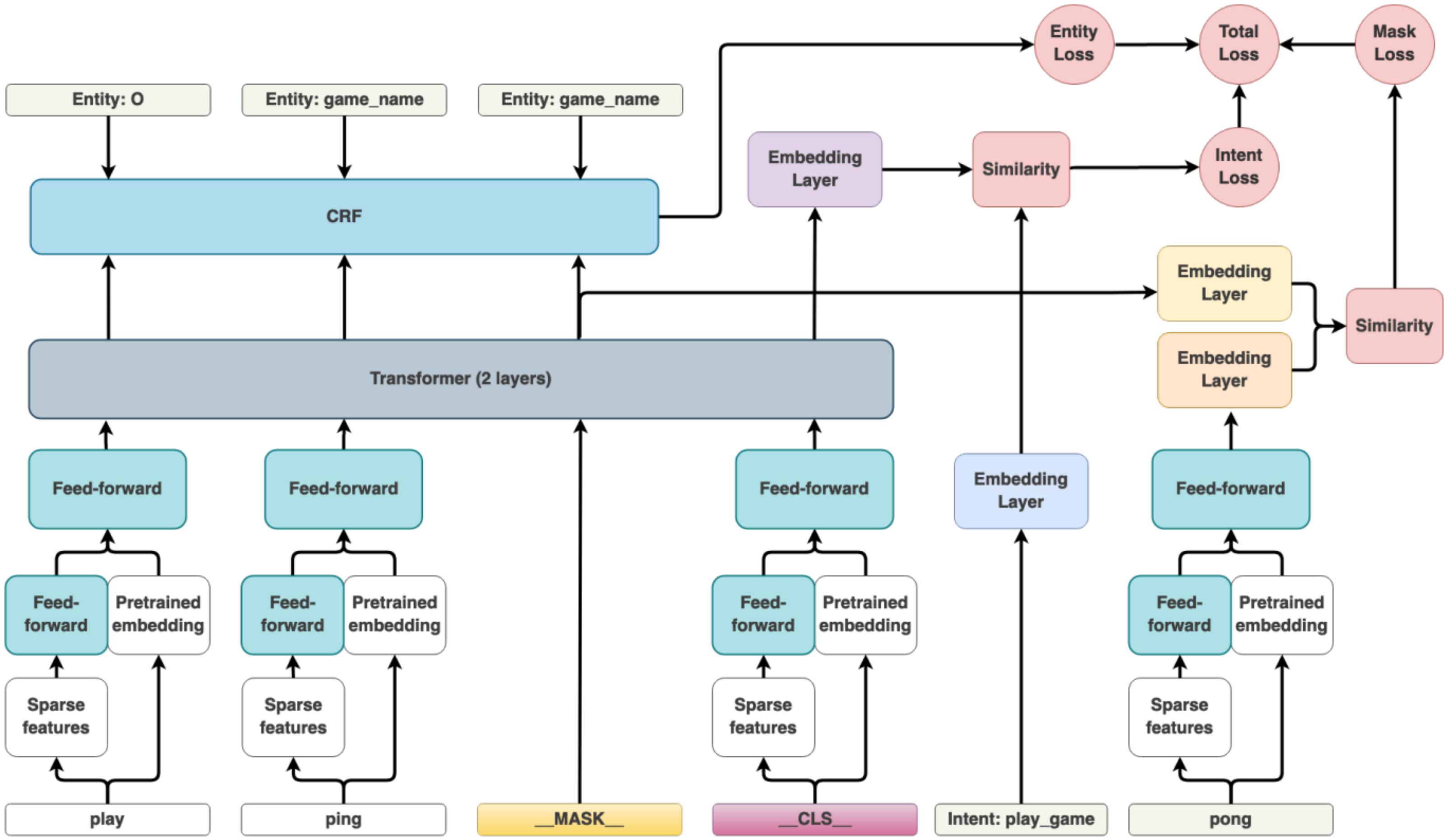


Figure 1: A schematic representation of the DIET architecture. The phrase ”play ping pong” has the intent play_game and entity game_name with value ”ping pong”. Weights of the feed-forward layers are shared across tokens.

https://arxiv.org/pdf/2004.09936

https://www.youtube.com/watch?v=vWStcJDuOUk&list=PL75e0qA87dlG-za8eLI6t0_Pbxafk-cxb&index=2

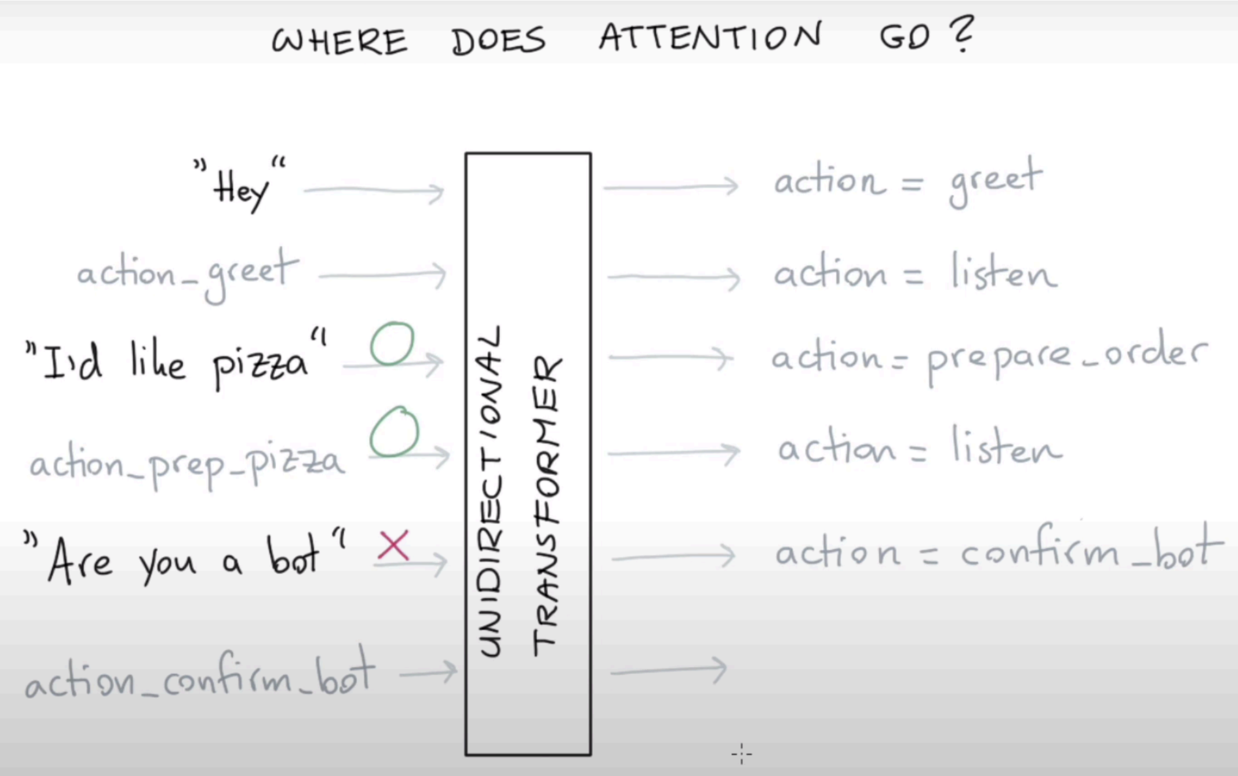
Model Configuration – Dialogue Policies

Your assistant uses policies to decide which action to take at each step in a conversation. There are machine-learning and rule-based policies that your assistant can use in tandem.

Supported policies:

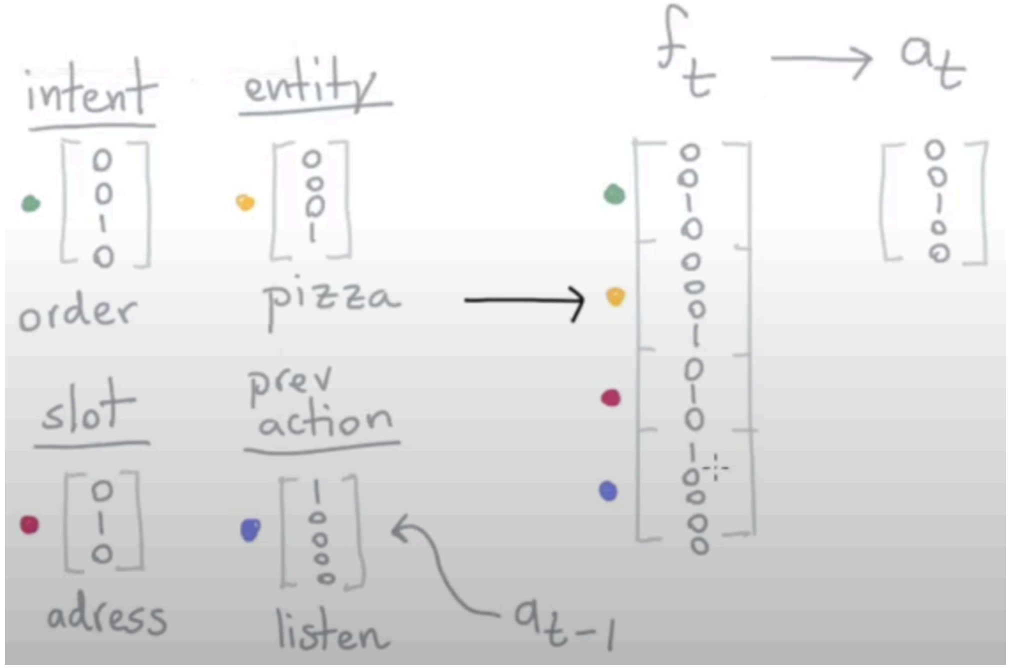
- Action Selection
  - Policy Priority
- Machine Learning Policies
  - TED Policy (https://arxiv.org/abs/1910.00486)
  - UnexpecTED Intent Policy
  - Memoization Policy
  - Augmented Memoization Policy
- Rule-based Policies
  - Rule Policy
- Configuring Policies
  - Max History
  - Data Augmentation
  - Featurizers
- Custom Policies

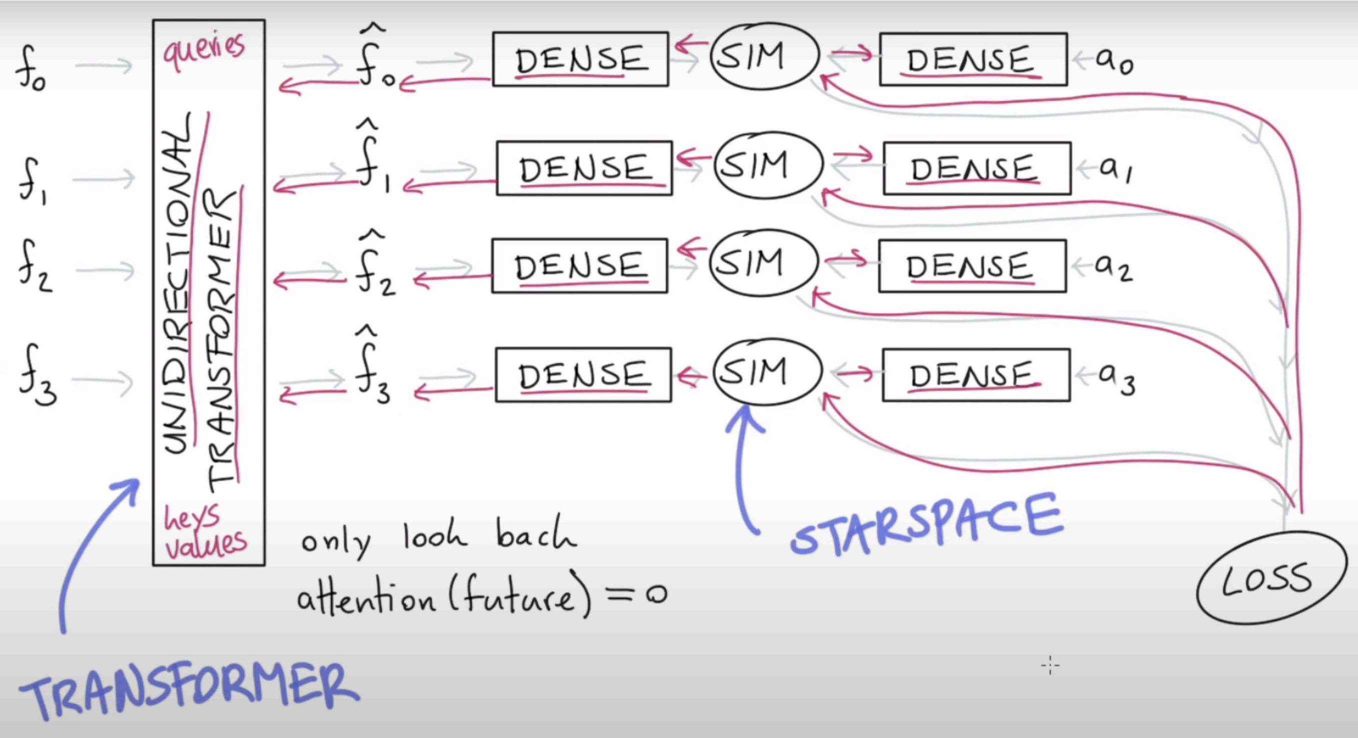
Transformer Embedding Dialogue (TED) Policy



https://rasa.com/docs/rasa/policies

Transformer Embedding Dialogue (TED) Policy





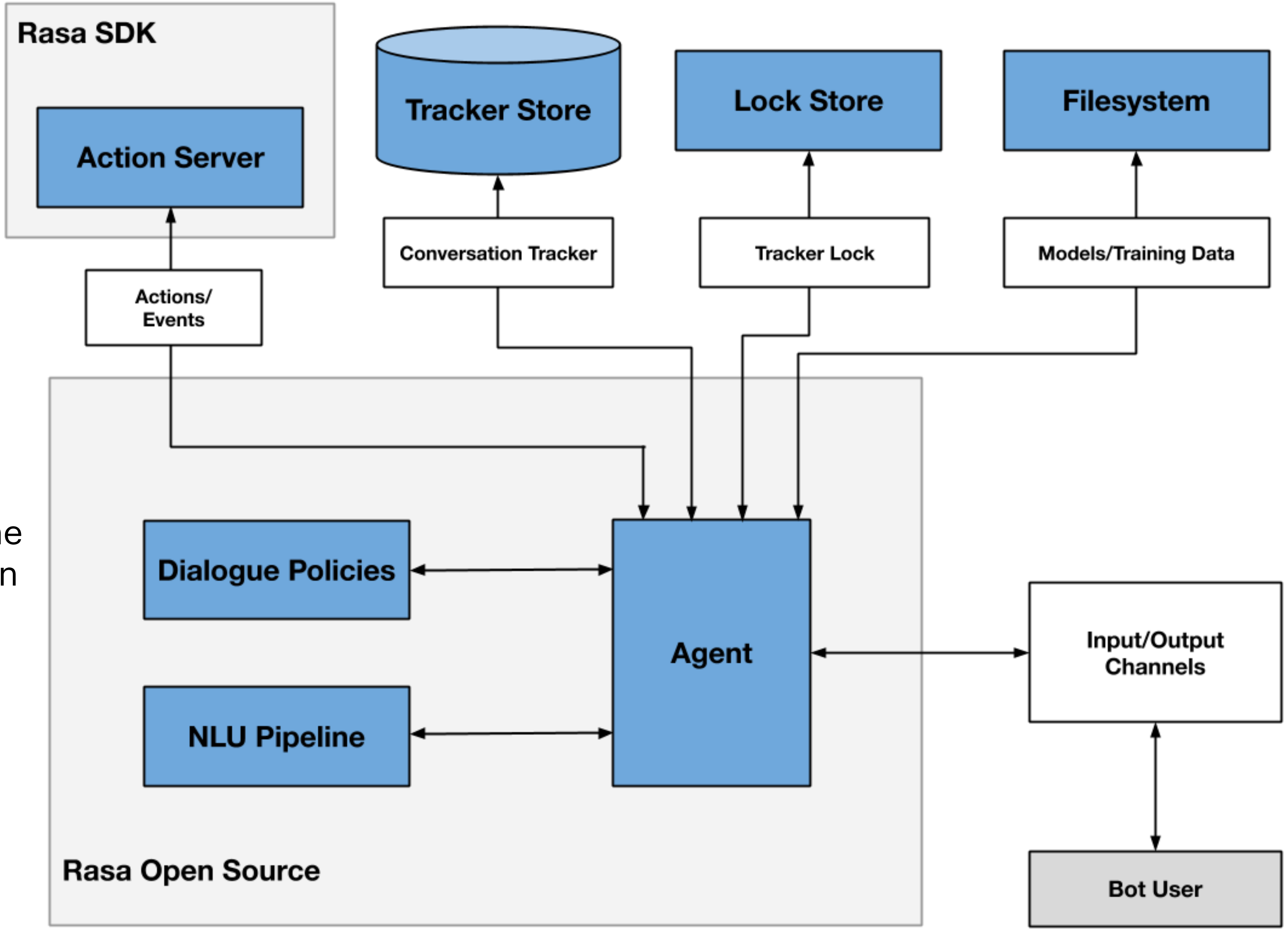
https://www.youtube.com/watch?v=j90NvurJI4I&list=PL75e0qA87dlG-za8eLI6t0_Pbxafk-cxb&index=14

Rasa Open Source Architecture

Action Server runs custom actions (e.g. API calls, database queries, etc.) for a Rasa Open Source conversational assistant.

Dialogue Policies decides the next action in a conversation based on the context.

NLU Pipeline handles intent classification, entity extraction, and response retrieval.



Supported channel connectors:

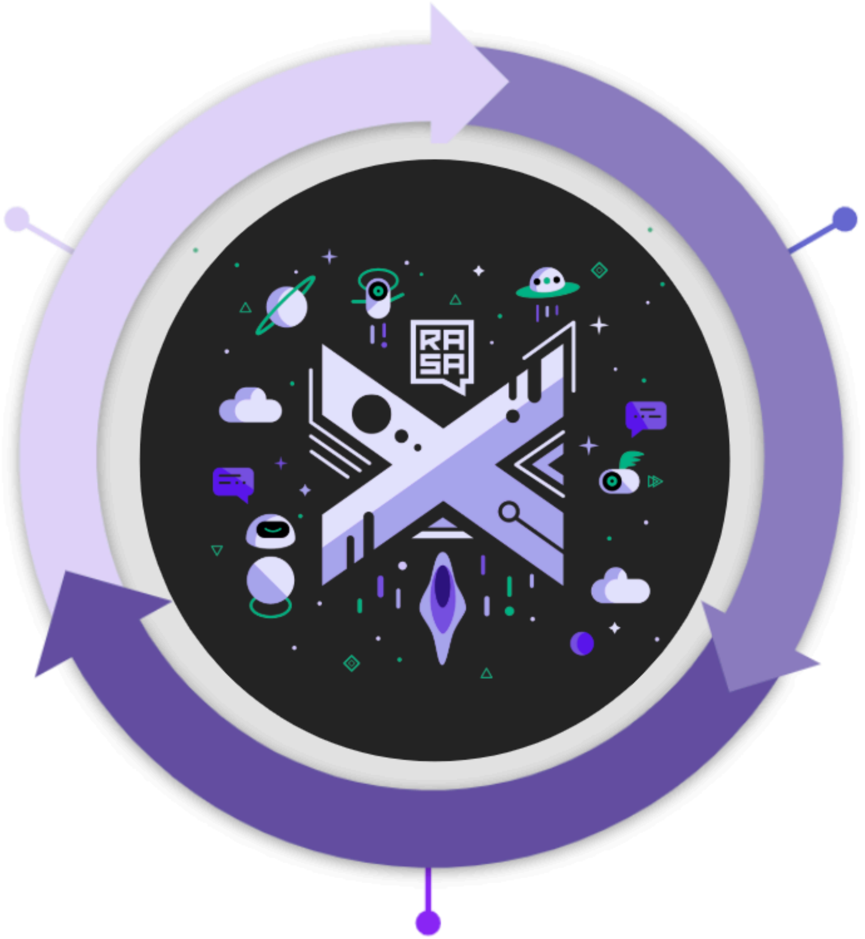
- REST Channels
- Websocket Channel
- Facebook Messenger, Google Hangouts Chat, Microsoft Bot Framework, Slack, Telegram, etc.

https://rasa.com/docs/rasa/arch-overview

Conversation-Driven Development (CDD) with Rasa X

Rasa X is a tool for Conversation-Driven Development (CDD), the process of listening to your users and using those insights to improve your AI assistant.

Continually improve your assistant using Rasa X



Ensure your new assistant passes tests using **continuous integration (CI)** and redeploy it to users using **continuous deployment (CD)**

Collect conversations between users and your assistant

Review conversations and **improve your assistant** based on what you learn

Rasa X:

- layers on top of Rasa Open Source and helps you build a better assistant
- is a free, closed source tool available to all developers
- can be deployed anywhere, so your training data stays secure and proprietary

Rasa X:

- Share your assistant with users
- Review conversations on a regular basis
- Annotate messages and use them as NLU training data
- Test that your assistant always behaves as you expect
- Track when your assistant fails and measure its improvement
- Fix how your assistant handles unsuccessful conversations

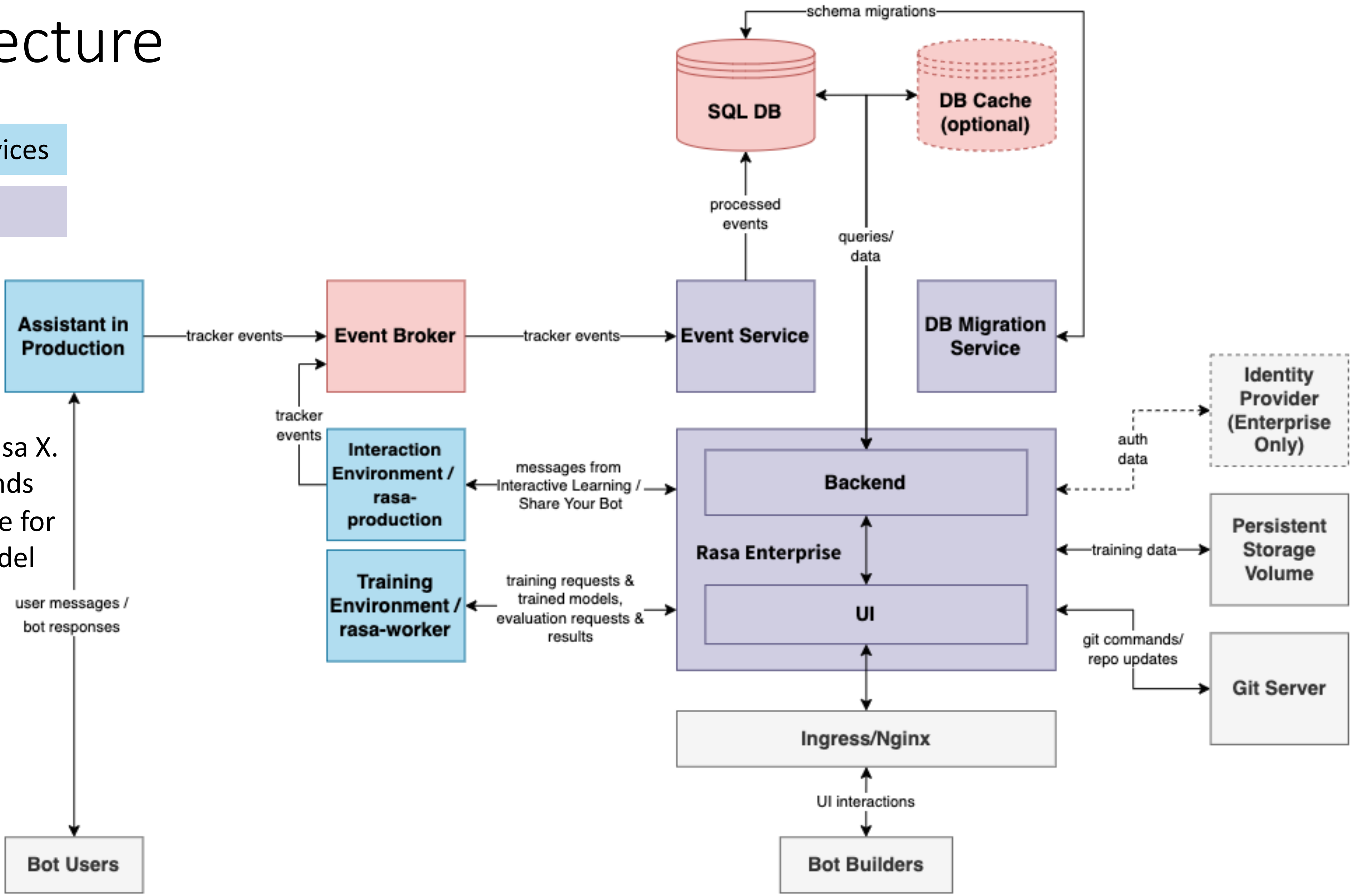
https://rasa.com/docs/rasa-enterprise/1.0.x

Rasa X Architecture

Blue: Rasa Open Source services

Purple: Rasa X services

Rasa Open Source can run completely independently of Rasa X. Rasa X on the other hand depends on the Rasa Open Source service for handling conversation data, model training and running.


schema migrations
SQL DB
DB Cache (optional)
processed events
queries/ data
Assistant in Production
tracker events
Event Broker
tracker events
Event Service
DB Migration Service
tracker events
Identity Provider (Enterprise Only)
auth data
Interaction Environment / rasa-production
messages from Interactive Learning / Share Your Bot
Backend
Rasa Enterprise
training data
Persistent Storage Volume
Training Environment / rasa-worker
training requests & trained models, evaluation requests & results
UI
git commands/ repo updates
Git Server
user messages / bot responses
Ingress/Nginx
UI interactions
Bot Users
Bot Builders

https://rasa.com/docs/rasa-enterprise/1.0.x/api/architecture