Rasa Certification Workshop

March 2020

Mady Mantha, Juste Petraityte, Karen White

Agenda

Day 1: Deep dive into NLU and dialogue management with Rasa Open Source, livecoding and testing

Day 2: Deep dive into DIET (Dual Intent and Entity Transformer) and TED (Transformer Embedding Policy) and create an MVP assistant

Day 3: Adding custom actions and implementing forms

Day 4: Deploying and improving your assistant using Rasa X

Day 5: Recap and certification

*More info blablabla

The Rasa Team

Mady Mantha
Sr. Technical Evangelist

Juste Petraityte
Head of Developer Relations

Karen White
Developer Marketing Manager

How to get help

- Please ask your questions in the **#workshop-help** Slack channel rather than the Zoom chat. Slack is the place the Rasa team will be monitoring most closely.
    - Karen, Mady, and Juste will be in Slack answering questions, as well as Arjaan, Melinda, and Ella from our Customer Success Engineering team
- Monday - Friday, the Rasa team will be dedicating time to answering your questions in Slack from 4 pm - 6 pm CEST (Central European Summer Time)
- Feel free to ask questions outside of these hours, but responses may be a little slower
- A note on time zones:
    - The Rasa team is based across the US and in Berlin. We'll do our best to answer questions within team members' working hours, but please keep in mind, some discussions may need to take place async rather than in real time.

Setting up the environment

Setup

Pre-workshop checklist:

Before you begin, you'll need:

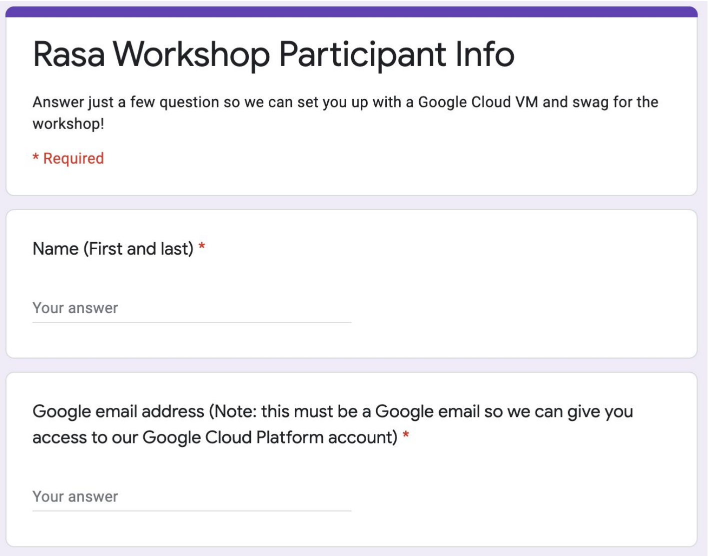
- ❏ GitHub account
- ❏ IDE or Text editor
- ❏ Python 3.6 or 3.7

Setting up the project:

- ❏ Virtual environment
- ❏ Financial Demo bot
  - ❏ Fork the repo
  - ❏ Clone a local copy
  - ❏ Install dependencies (including Rasa Open Source)
- ❏ Run the bot!
- ❏ Telegram account

Reminder: Fill out the form to provide your Google email address

Later in the workshop, we'll be using Google Cloud Platform to deploy Rasa X and your assistant. To set up a VM for you, we need your Google email address. If you haven't done it yet, please fill in the Rasa Workshop Participant Info survey (sent by email and linked in the Workshop Setup handout)



Rasa Workshop Participant Info

Answer just a few question so we can set you up with a Google Cloud VM and swag for the workshop!

* Required

Name (First and last) *

Your answer

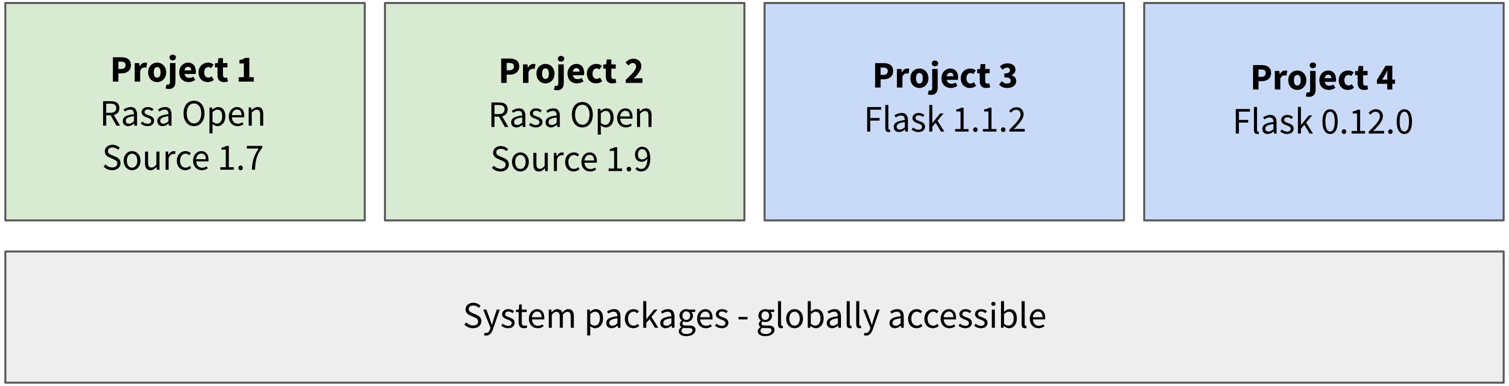
Google email address (Note: this must be a Google email so we can give you access to our Google Cloud Platform account) *

Your answer

Virtual Environments

Virtual environments let you scope packages to a specific project directory, instead of installing the package globally on your system. This allows you to use different versions of the same package for different projects.

We're using **venv**, which is built into Python. Feel free to use alternatives like Anaconda or virtualenv if you prefer those tools.



Setup

Creating the virtual environment

1. Create a project directory
   mkdir rasa-workshop
   cd rasa-workshop

2. To create a virtual Python environment run:
   python3 -m venv ./venv

3. Active the environment (Ubuntu/Mac):
   source ./venv/bin/activate

   Activate the environment (Windows):
   .\venv\Scripts\activate

4. Deactivate your virtual environment:
   deactivate

2. Installing Rasa 1.9.3

For this workshop we are going to use the latest Rasa Open Source release which is 1.9.3:

1. Install Rasa 1.9.3:
   ```
   pip3 install rasa==1.9.3
   ```

2. Check that the correct Rasa version has been installed:
   ```
   rasa --version
   ```

Fork the financial bot repository

We’ll be using an open source AI assistant for this workshop - Financial Demo bot.

This bot can:

- Answer questions about account balance
- Transfer funds to another account
- Check spending/earning history

1. Fork the repository:
   https://github.com/RasaHQ/financial-demo

2. Clone the repo to create a local copy on your computer:
   git clone <url to your repository>

A quick note about GitHub

We'll be using GitHub during this workshop to manage different versions of the assistant, and to port the assistant's code into Rasa X when it's time to deploy to a server on day 4. Here's a cheat sheet of handy commands

git remote -v	Show remote repositories
git remote add upstream https://github.com/RasaHQ/financial-demo.git	Set upstream repo so we can fetch changes from the repo we forked from
git fetch <remote name>	Download new branches from remote repo
git branch -a	List all branches (local and remote)
git checkout <existing branch name>	Switch to a different branch
git status	Track uncommitted changes
git stash	Temporarily shelve uncommitted changes (so you can switch to a different branch)

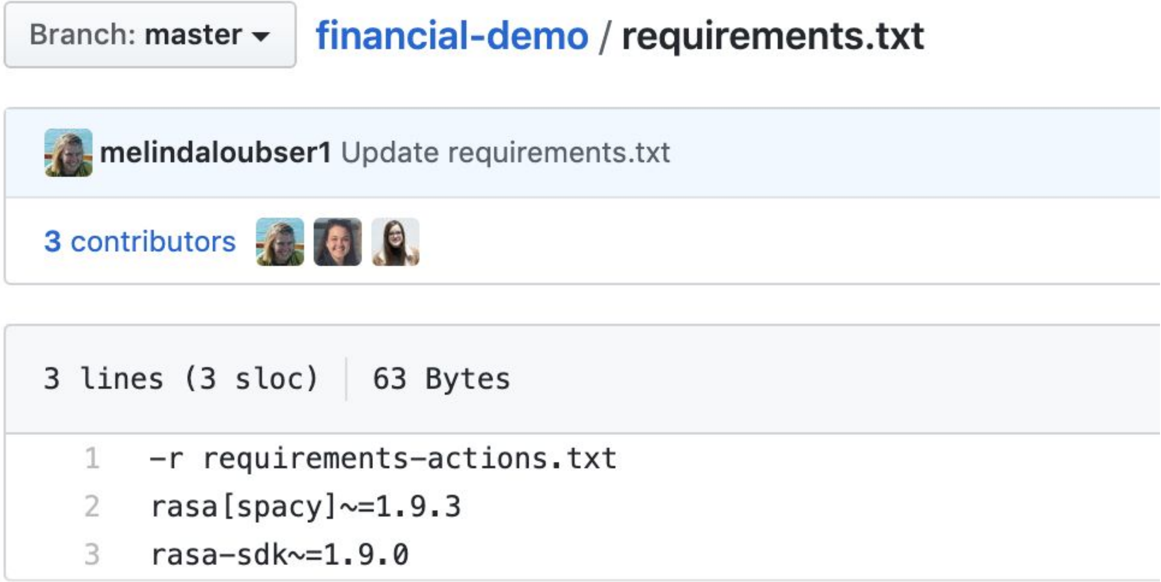
Setup

Install project dependencies (including Rasa Open Source)

Install the dependencies:

cd financial-demo

pip install -r requirements.txt



4. Downloading SpaCy language model

Download the SpaCy English language model.

1. Download the model:
   python3 -m spacy download en_core_web_md

2. Link the model:
   python -m spacy link en_core_web_md en

Optional: Download Docker

The Financial Services demo bot uses Docker to run Duckling locally. Duckling is an entity extractor for dates (but more on that later).

If you want to follow along, you can download Docker using the instructions here.

If you run into trouble, don't worry - we'll be providing a live Duckling server later on that you can use instead of running Docker.

Start the assistant

1. Train the model
   rasa train

2. Start the duckling server
   docker run -p 8000:8000 rasa/duckling

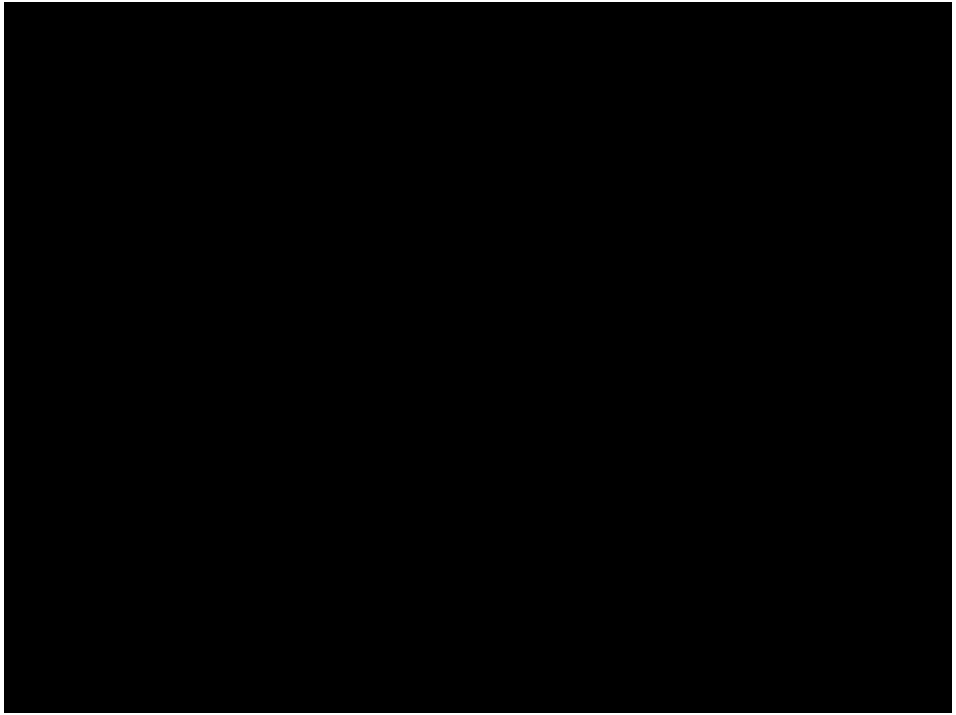
3. Start the action server and a new shell session
   rasa run actions & rasa shell

You can now talk to the assistant on the command line! Try asking “what’s my account balance” or “how much did I spend at Target?”

5. Creating your Telegram account

At the later steps of our workshop we will connect our assistant to the outside world using Telegram. For that you will need a Telegram account (video tutorial included):

1. Download Telegram on your computer (or phone)
2. Setup your Telegram account
3. Login and be ready to connect your Rasa Assistant to it.



Office Hours

Need one-on-one help getting set up? We're here to help.

Keep an eye out for an email invitation to the Slack workspace we've set up for the workshop. We'll also email you the date/time for an Office Hours session we're holding before the workshop. We'll have experts from Rasa monitoring Slack during Office Hours to answer any questions and help you get set up.

See you then!

7. Setting up the IDE

To follow this workshop you can use an IDE or a Text Editor of your choice. The instructors of this workshop will be using Visual Studio Code. If you prefer using the same IDE you can download it from here.

4. Connecting to your Google Cloud instance:

In the later stages of the workshop we will be using Rasa X to improve our assistant. We will deploy Rasa X on an Google Cloud VM.

1. Connect to your Google Cloud VM instance :
   Link

2. Once connected to the instance, install Rasa X:
   curl -s get-rasa-x.rasa.com | sudo bash

3. Open your Rasa X instance in the browser by pointing to the provided IP address

Intro to Rasa

Agenda

Day 1: Deep dive into NLU and dialogue management with Rasa Open Source, livecoding and testing

Day 2: Deep dive into DIET (Dual Intent and Entity Transformer) and TED (Transformer Embedding Policy) and create an MVP assistant

Day 3: Adding custom actions and implementing forms

Day 4: Deploying and improving your assistant using Rasa X

Day 5: Recap and certification

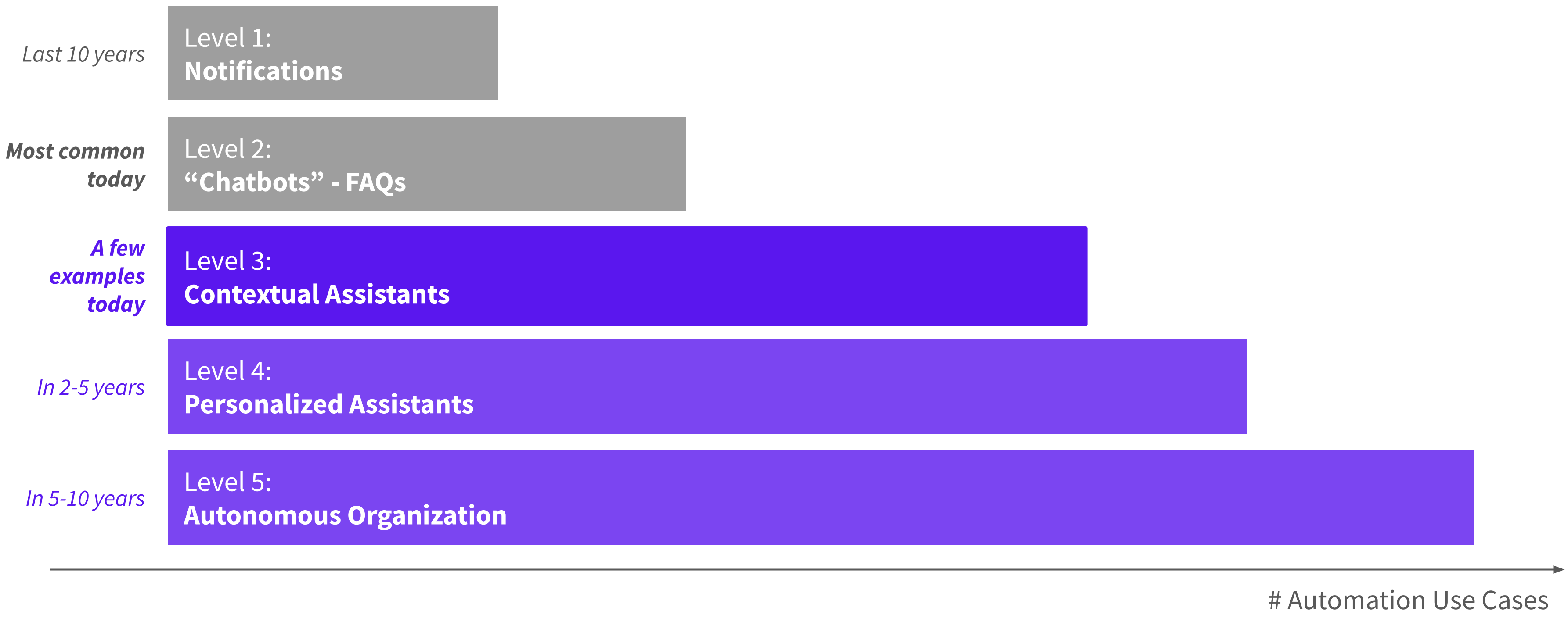
*More info blablabla

Day 1 Roadmap

- First half: talk, with some time for questions
- Second half: we'll code together
- Recap: talk, with some time for questions

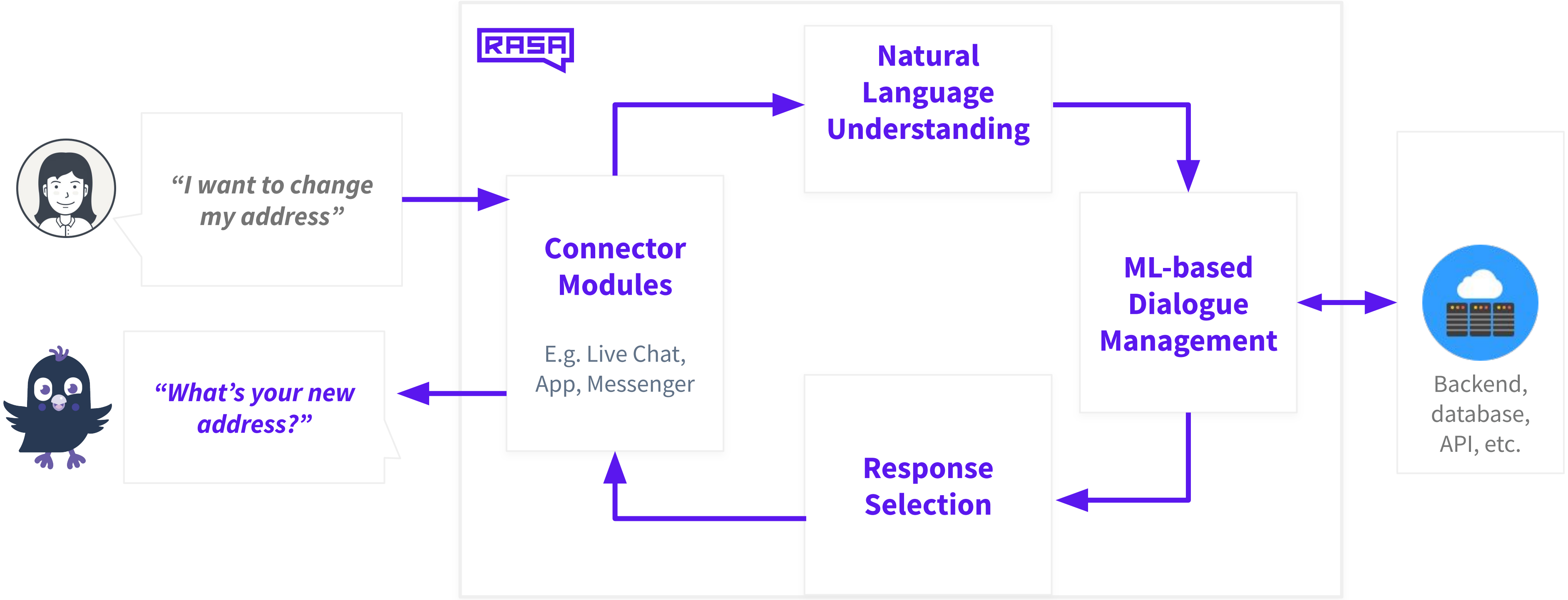
5 LEVELS OF CONVERSATIONAL AI MATURITY MODEL

Contextual assistants are an important step on the journey to autonomous organizations



RASA OPEN SOURCE

Rasa Open Source

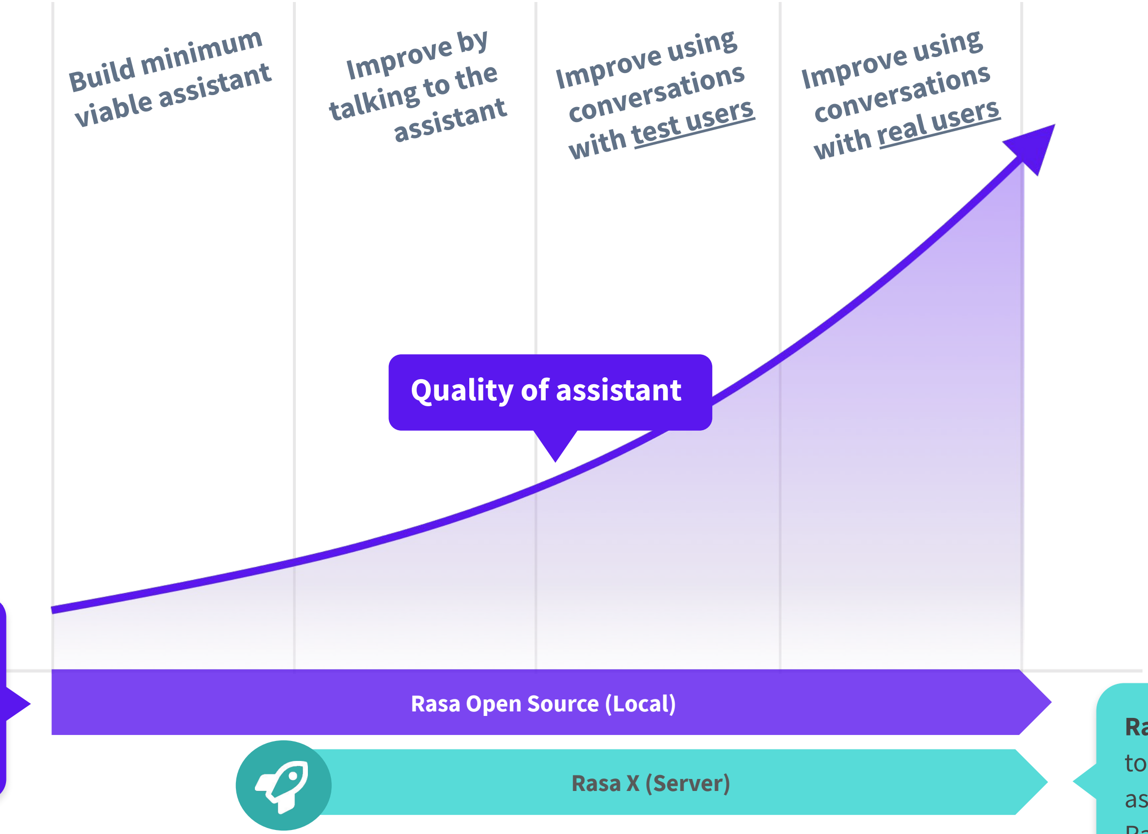


Conversational AI is not easy

Real conversations don’t follow the happy path

THE APPROACH

Build a minimum viable assistant with Rasa Open Source + improve it using Rasa X



Rasa Open Source is an open source framework for natural language understanding, dialogue management, and integrations.

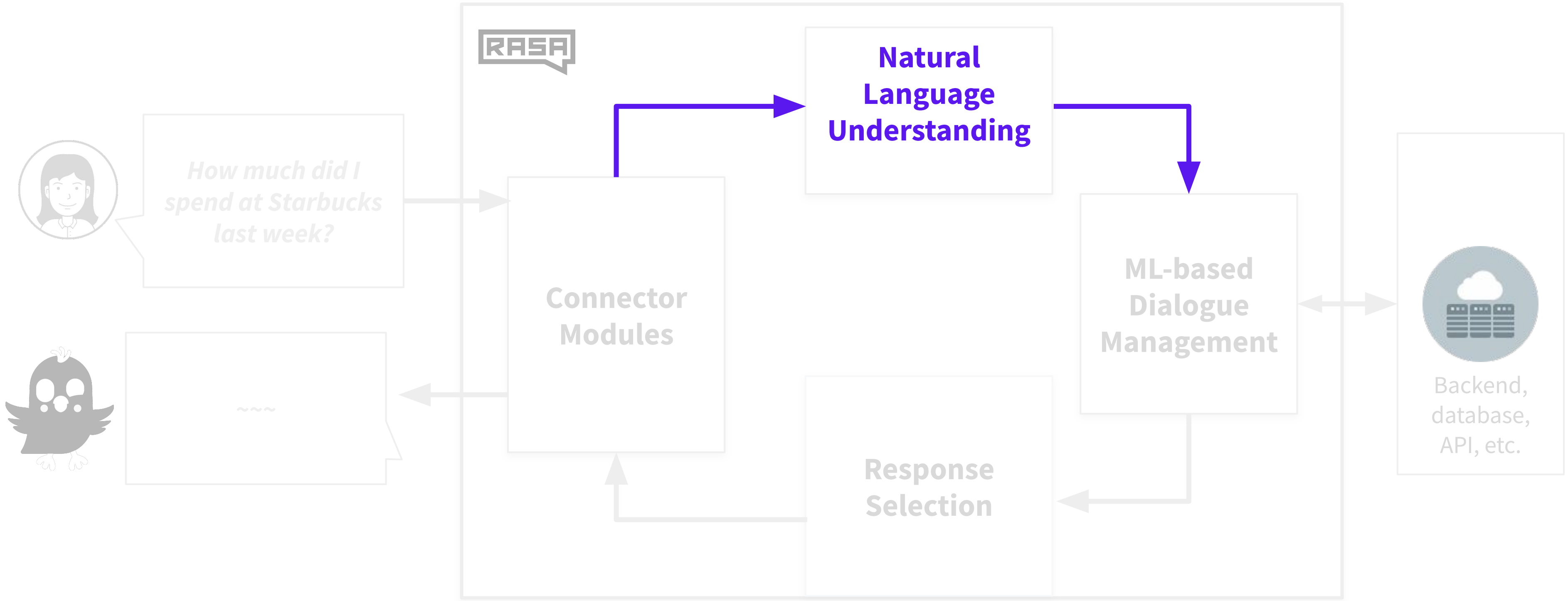
Rasa X is a toolset used to improve a contextual assistant built using Rasa Open Source.

RASA

NLU & Dialogue Management

NLU Deep Dive

Natural Language Understanding (NLU)



Natural Language Understanding (NLU)

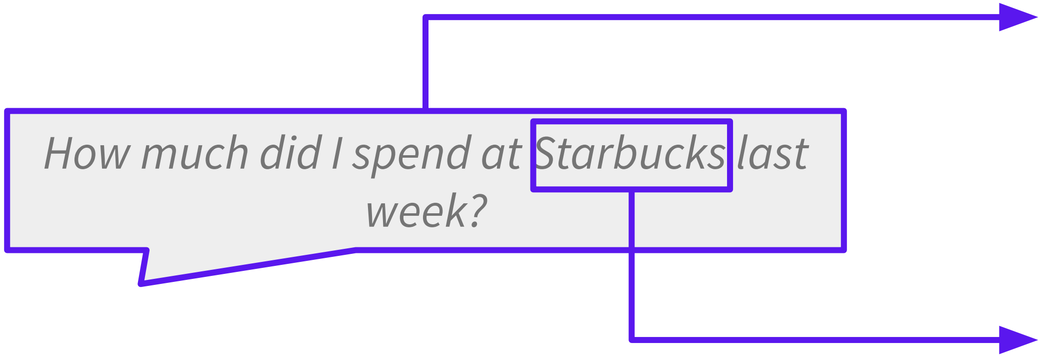
Goal: Extract structured information from messages

How much did I spend at Starbucks last week?

```
{...
    "intent": {
        "name": "search_transactions",
        "confidence": 0.96
    },
    "entities": [
        {
            "entity": "vendor_name",
            "value": "Starbucks",
             ...
        }
    ]
}
```

Intents and Entities

Two of the most common and necessary types of information to extract from a message

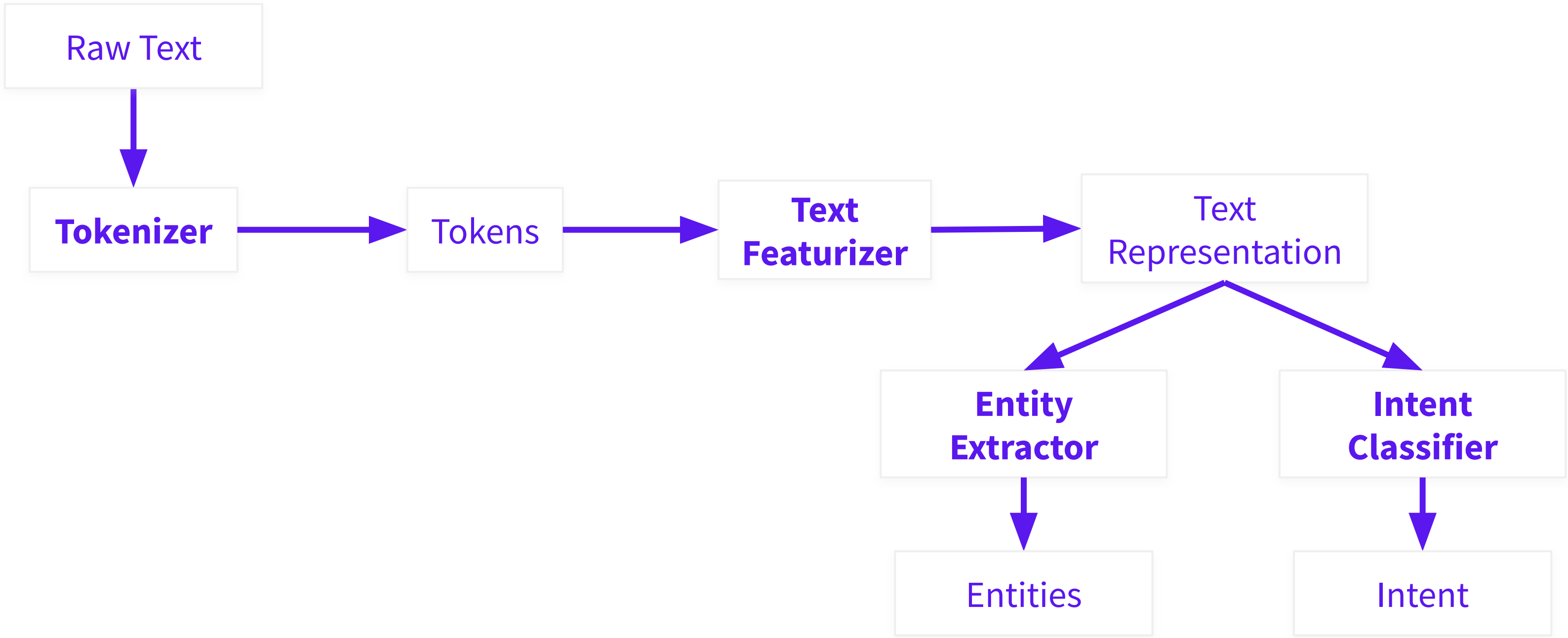


Intent: What the user wants to achieve

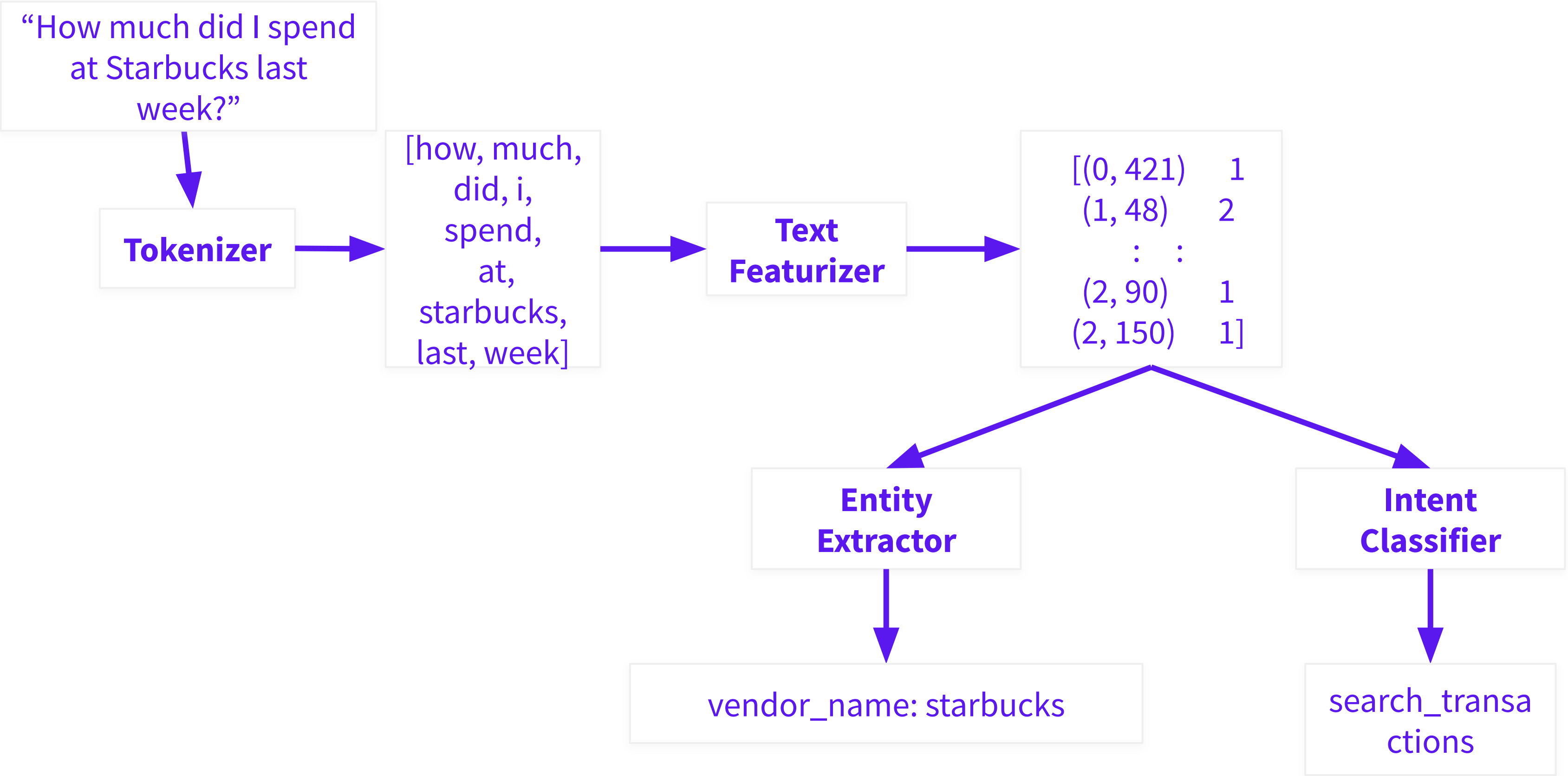
Entity: Key piece of information

NLU Pipeline: Input and Output

Many components take the output of earlier components as input

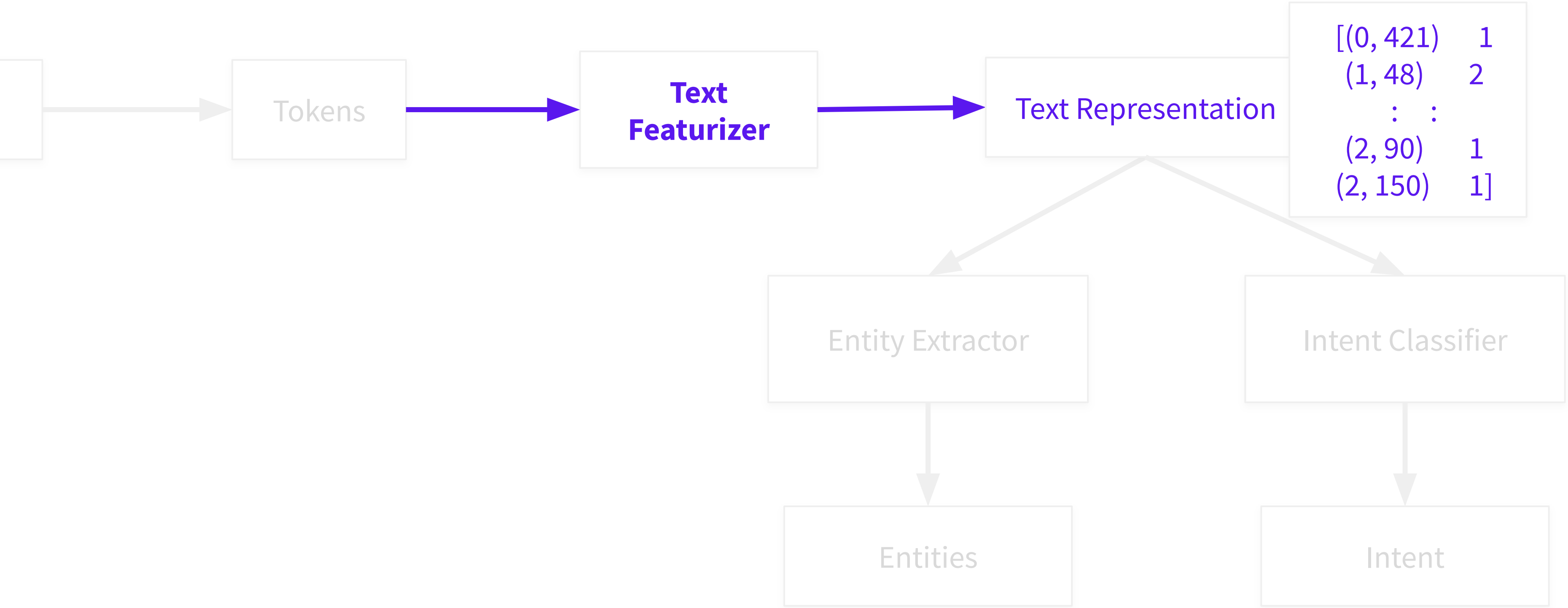


NLU Pipeline: Example

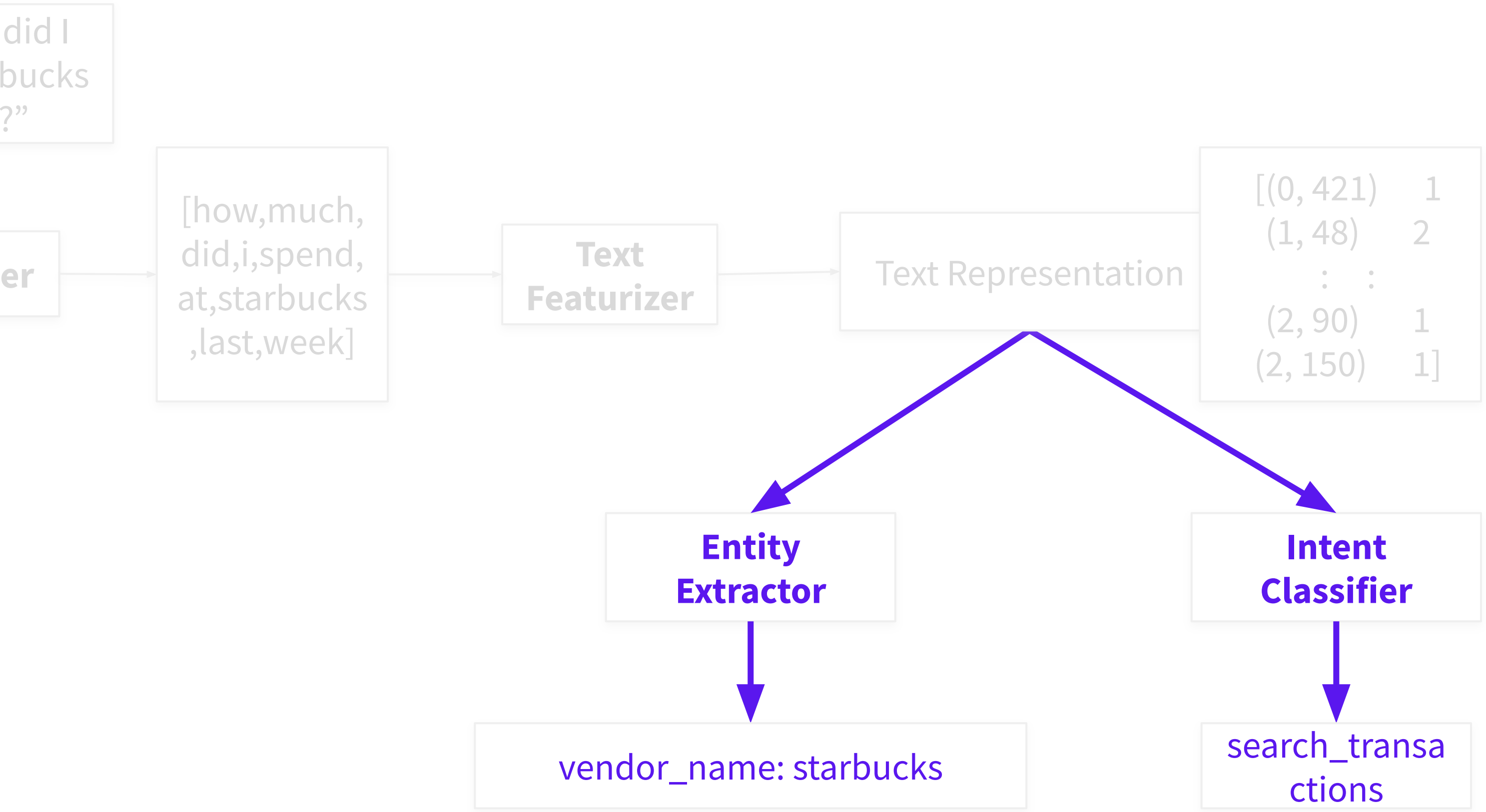


"Text Representation"

How do words turn into numbers?



Intent Classification & Entity Extraction



Text Representation: Word Vectors

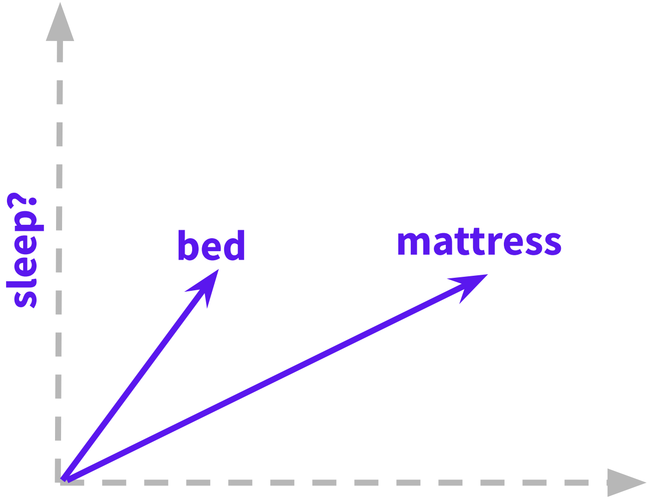
"Judge a Word by the Company it Keeps" — John R. Firth

"… *sleep* on a **bed**…"

"…*sleep* in a **bed**?"

"…*sleep* on a **mattress**…"

"…**mattress** to *sleep* on…"



These would be two of **many** dimensions!

Text Representation: How models use word vectors

Non-sequence model:

- One feature vector per input (whole message)
- Order of words not captured

Sequence model:

- One feature vector per token (word) & feature vector for whole message
- Word order captured

Configuration Files

Let’s start coding!

1. Let’s use a sandbox branch instead of the master branch you cloned yesterday. Make sure you are inside the financial-demo folder:
   cd financial-demo

2. Fetch the remote branches:
   git fetch origin

3. Checkout the sandbox branch named “workshop_day_2_sandbox”:
   git checkout -b workshop_day_2_sandbox origin/workshop_day_2_sandbox

Core Deep Dive

Domain

Defines the “world” of your assistant - what it knows, can understand, and can do

```
intents:
…
entities:
…
slots:
…
actions:
…
responses:
…
session_config:
…
```

What your bot can understand

What your bot can remember

What your bot can do (and say)

What your bot can say (and display)

What your bot considers a new conversation

Project setup: Files

__init__.py	an empty file that helps python find your actions
actions.py	code for your custom actions
config.yml	configuration of your NLU and dialogue models
credentials.yml	details for connecting to other services
data/**nlu.md**	your NLU training data
data/**stories.md**	your stories
domain.yml	your assistant's domain
endpoints.yml	details for connecting to channels like fb messenger
models/<timestamp>.tar.gz	your initial model

NLU Pipeline

Defines how a user message is processed & what information is extracted

```
language: "en"

pipeline:
 -   name: "Whitespacetokenizer"
 -   name: "ConveRTFeaturizer"
 -   name: "RegexFeaturizer"
 -   name: "LexicalSyntacticFeaturizer"
 -   name: "CountVectorsFeaturizer"
```

Components

NLU Pipeline: Components

Built-in component types:

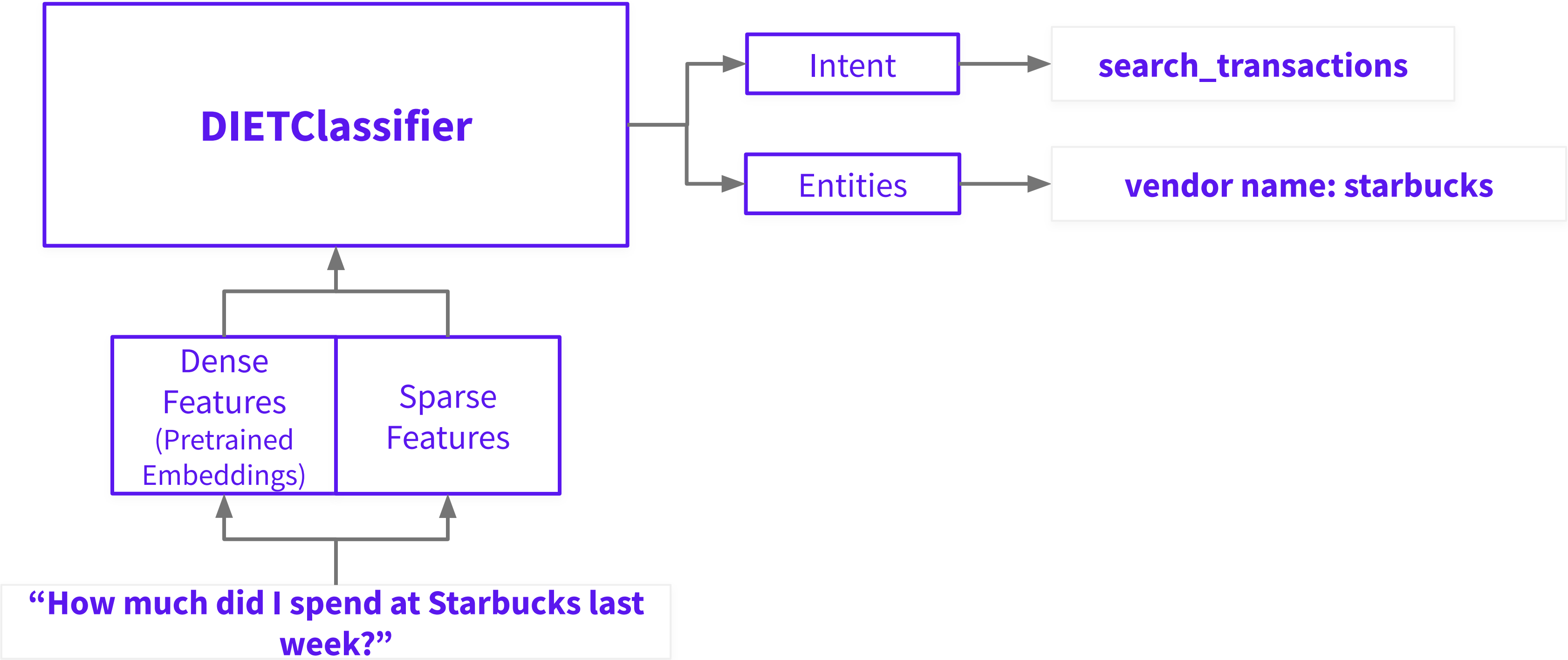
- Tokenizers
- Word Vector Sources
- Text Featurizers
- Entity Extractors
- Intent Classifiers
- Response Selectors

You can also add custom components for e.g.

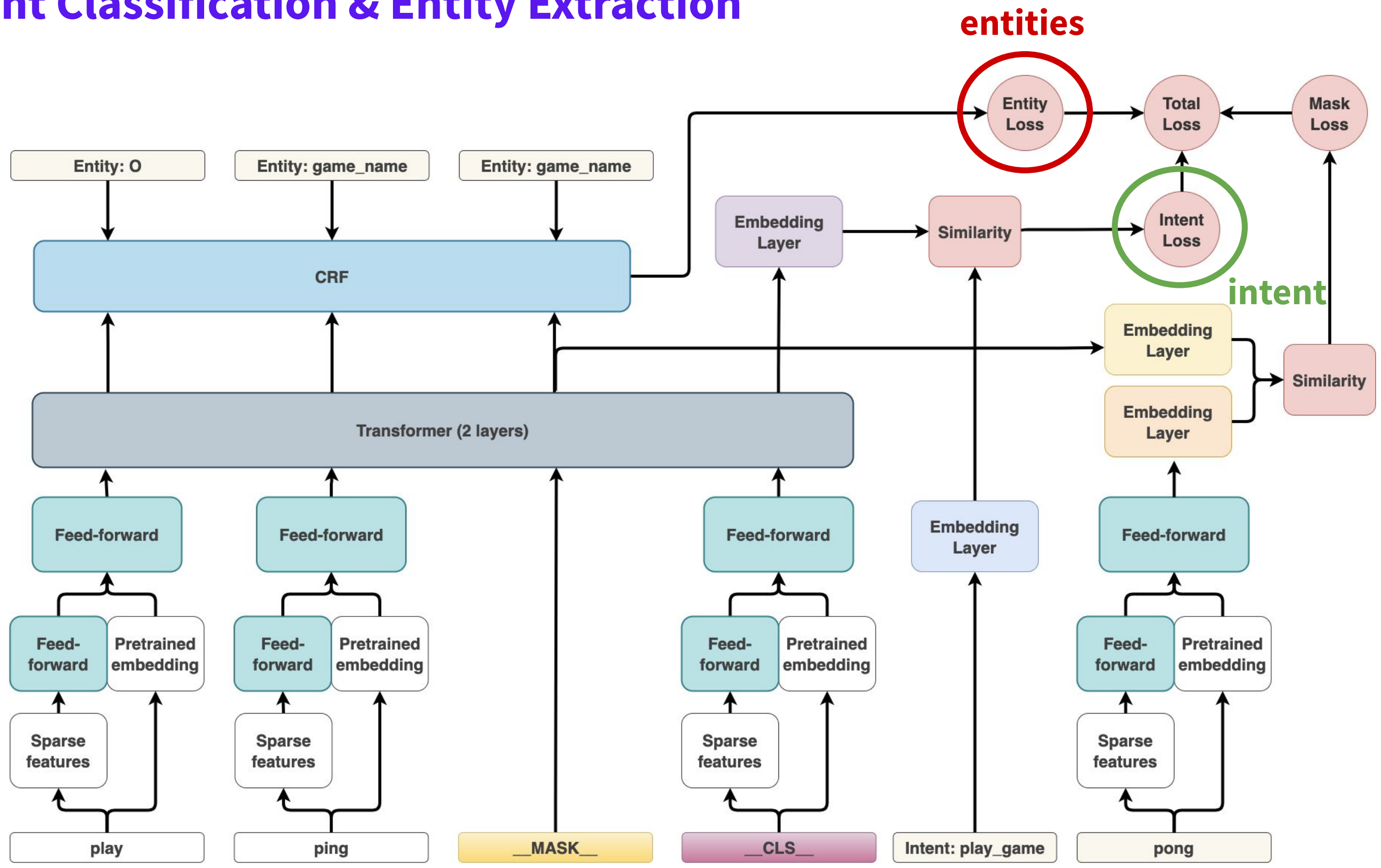
- Sentiment analysis
- Spell checking

DIETClassifier: Intent Classification & Entity Extraction

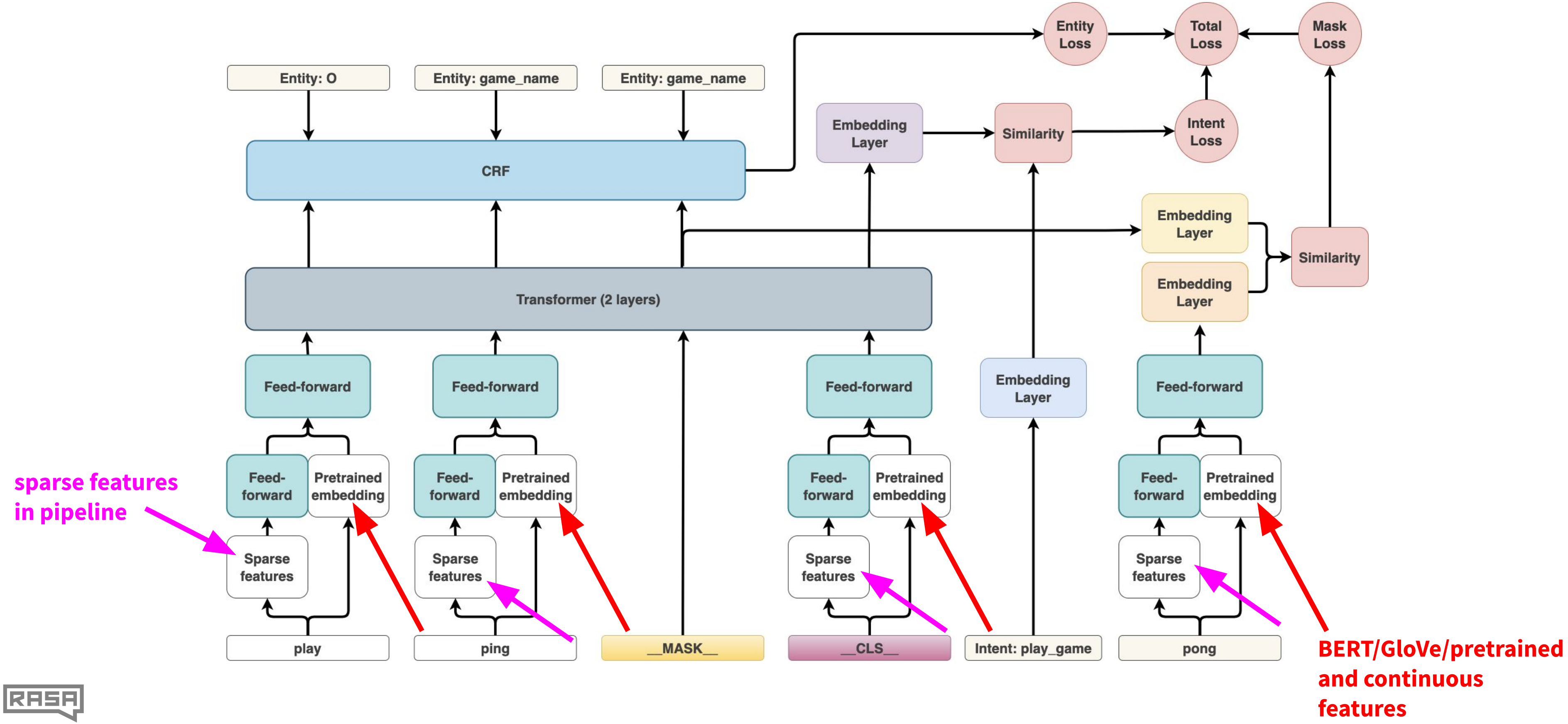
The Short Story



DIET: Intent Classification & Entity Extraction



Feature: Customize What Features Go In!



Entity Extraction: Other options

Possible Approaches:

1. Rule-based entity extraction
2. Direct structured prediction of entity based on sentence context

`DIETClassifier`
`MitieEntityExtractor`

`SpacyEntityExtractor`
`DucklingHTTPExtractor`

`EntitySynonymMapper`

Pretrained

Predefined, rule-based

Trained on your data

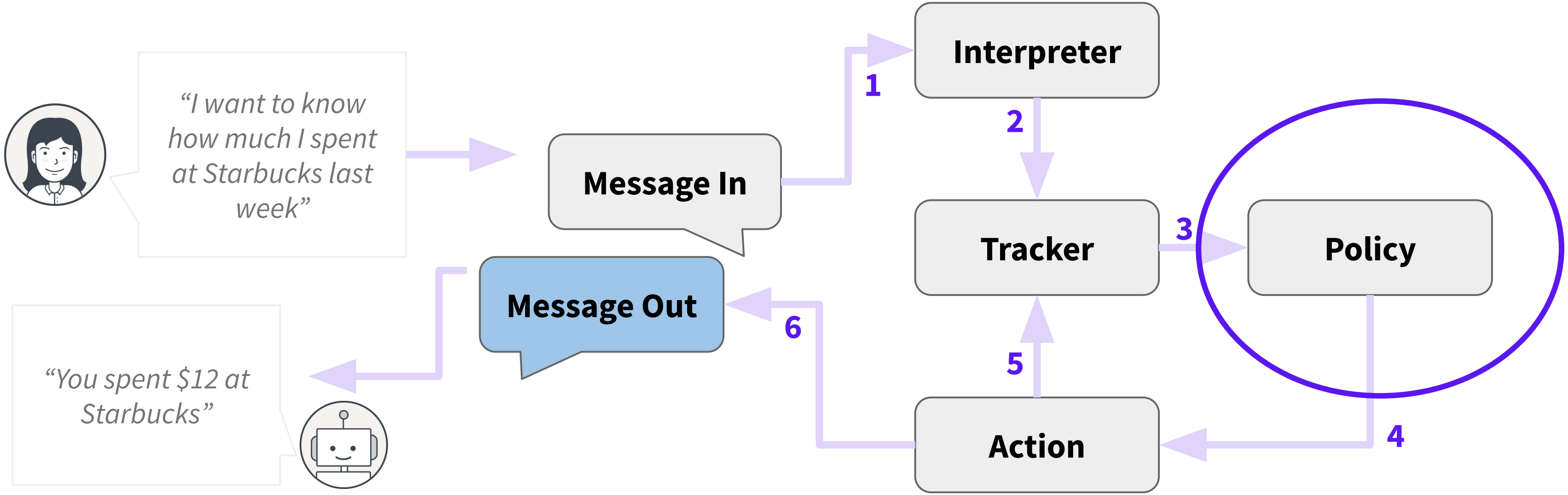
Modifies extracted entities

Livecoding

Let's create some intents and entities!

Policies

- Decide which action to take at every step in the conversation
- Each policy predicts an **action** with some **probability.** This is called **core confidence**.



Rule Based Policies

- **MemoizationPolicy:** Memorizes your stories
  - Makes predictions based on your max_history value
- **AugmentedMemoizationPolicy:** Memorizes your stories with a twist
  - As above, but if no match is found, it will forget certain events until a match is found
- **FormPolicy:** Fills required slots
  - Asks for information until all required slots are filled
- **MappingPolicy:** Intents trigger actions
  - Will execute the specified action regardless of context unless superseded by another policy
- **FallbackPolicy**: Failing gracefully
  - If model confidence is below a certain value, it will send the user a “fallback message”, e.g. “Sorry, I didn’t understand you”
- **TwoStageFallbackPolicy**: Failing (more) gracefully
  - Attempts to disambiguate the user’s intent before sending a fallback message

Machine Learning Based Policies

These policies should be used in conjunction with rule-based policies

- **KerasPolicy:** Uses a standard LSTM to predict the next action
  - Learns the patterns of your stories
  - Good for handling stories that don’t exactly match your training data
- **TED Policy:** Uses Attention to Handle Uncooperative Dialogue
  - Requires fewer story examples of uncooperative user dialogue
    - e.g. users who go off on tangents instead of providing the requested information
  - Effectively “ignores” irrelevant parts of the dialogue

Multiple Policies

- The policy with the **highest confidence** wins.
- If the confidence of two policies is equal, the policy with the **highest priority** wins.

Rule-based policies have higher priority than ML-based policies

```
Policy priorities
(higher numbers = higher priority)

5. FormPolicy
4. FallbackPolicy, TwoStageFallbackPolicy
3. MemoizationPolicy, AugmentedMemoizationPolicy
2. MappingPolicy
1. EmbeddingPolicy, KerasPolicy, TED
```

Conversation Design

Initially: Conversations as simple question / answer pairs

What can I help you with today?

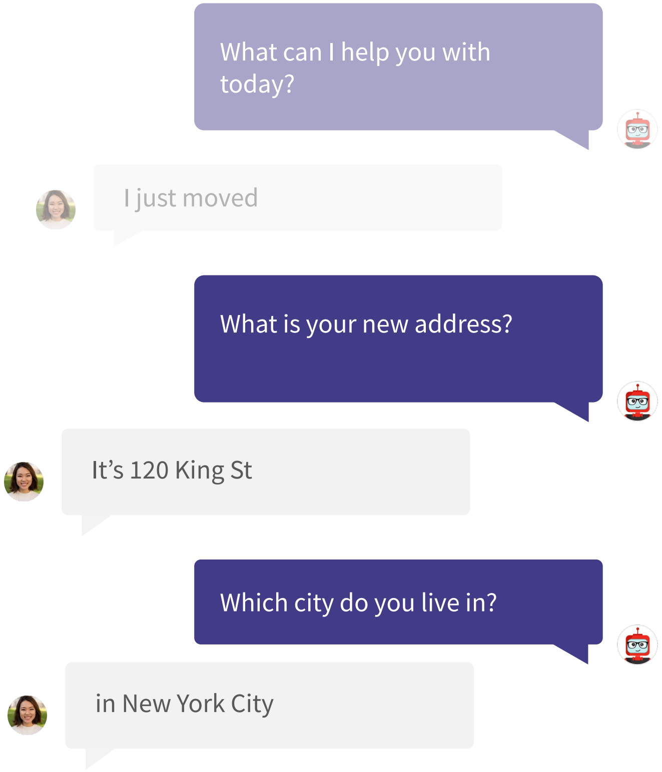
I just moved

What is your new address?

It’s 120 King St in New York City.

Core Deep Dive

Real conversations are more complex: Rely on context



Scope Conversation

How to get started with conversation design

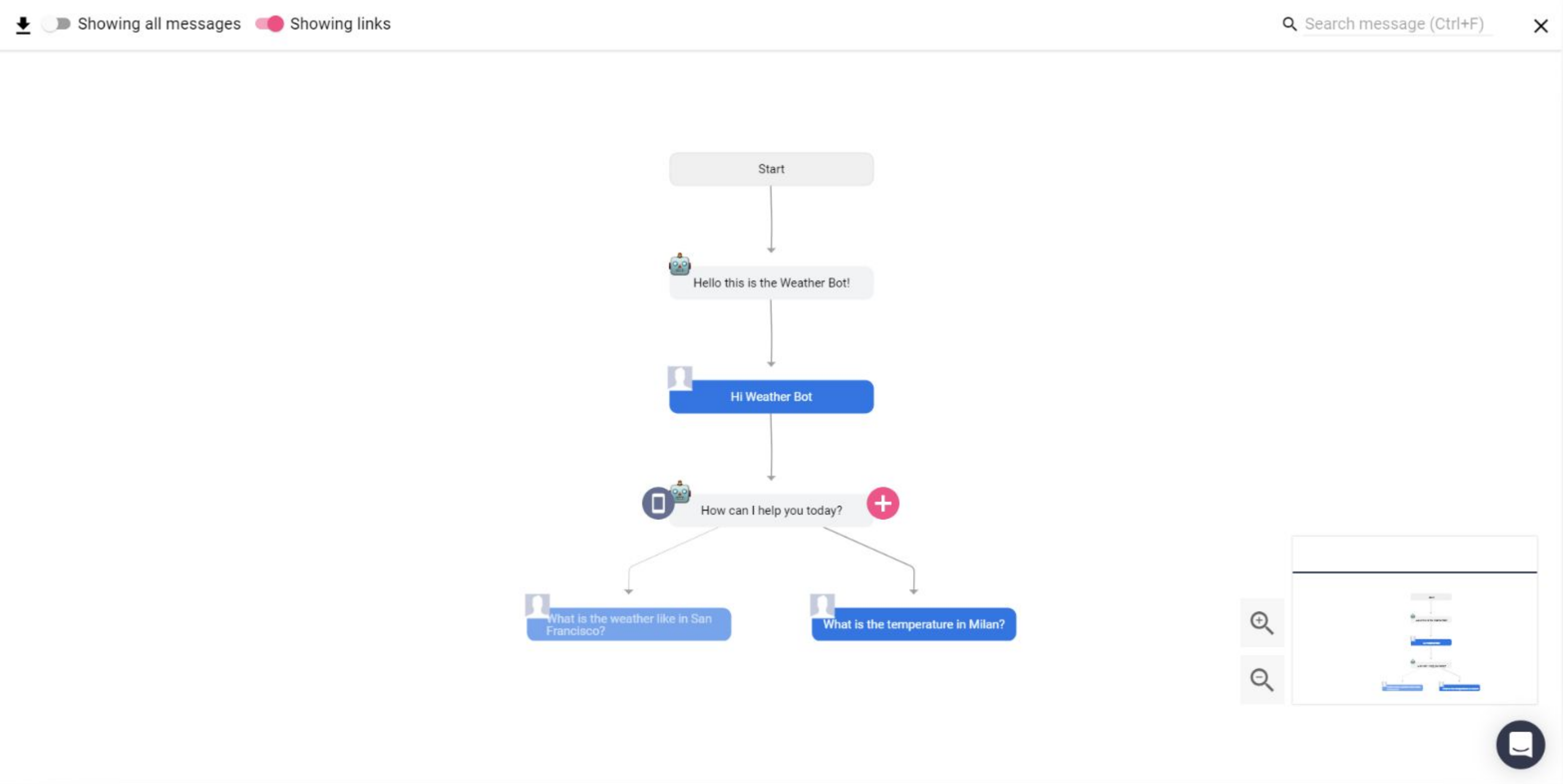
The assistant’s purpose

Leverage the knowledge of domain experts

Common search queries

FAQs and wikis

Scope Conversation



Source: Botsociety

Livecoding

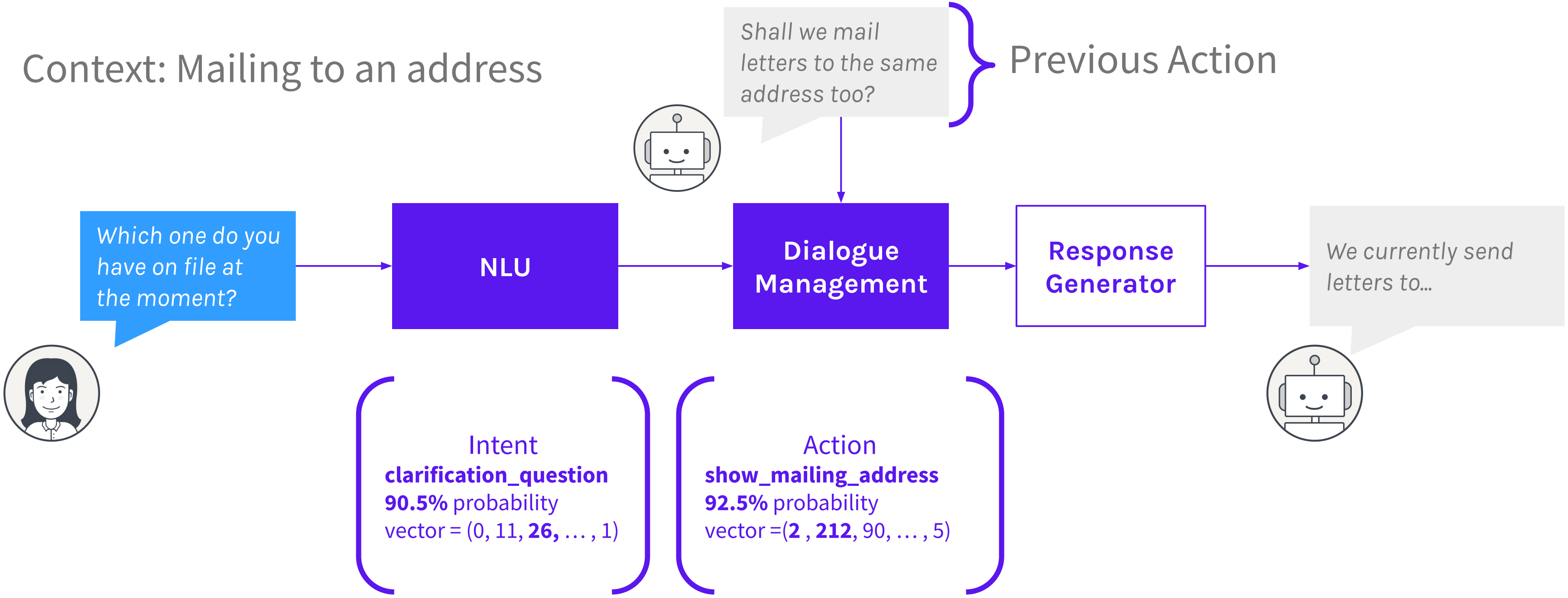
Let's create some stories together!

- Check account balance
- Search transactions

Core Deep Dive

The importance of context

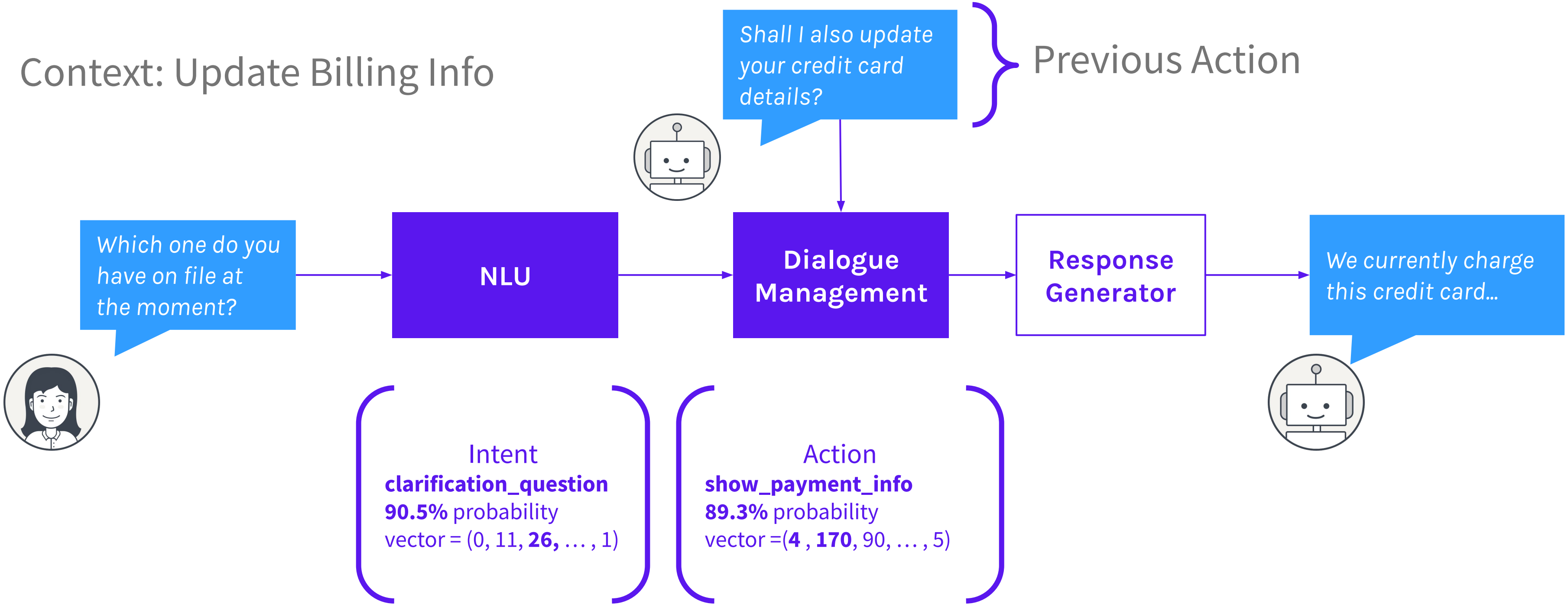
Context: Mailing to an address



Core Deep Dive

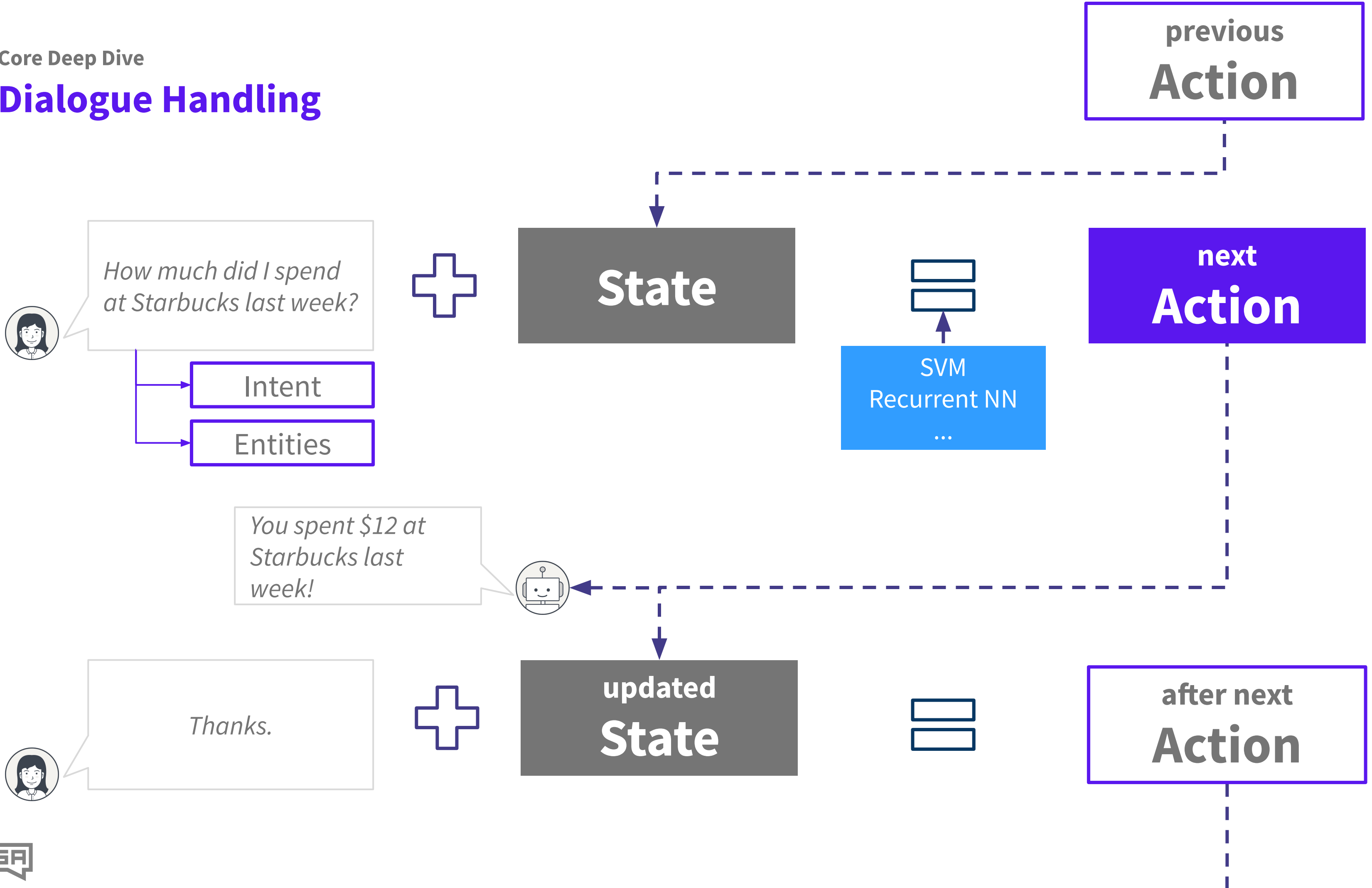
The importance of context

Context: Update Billing Info

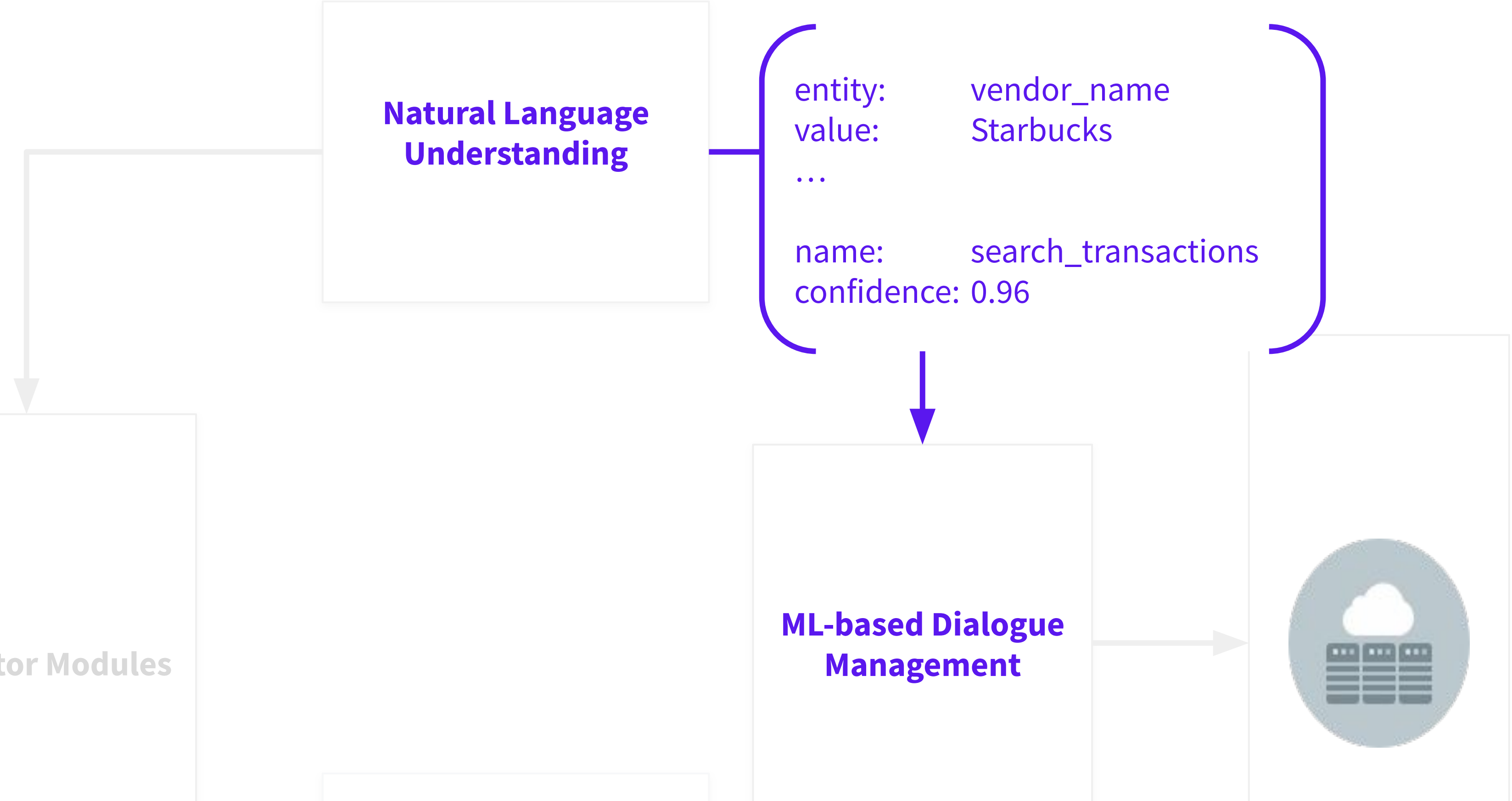


Core Deep Dive

Dialogue Handling

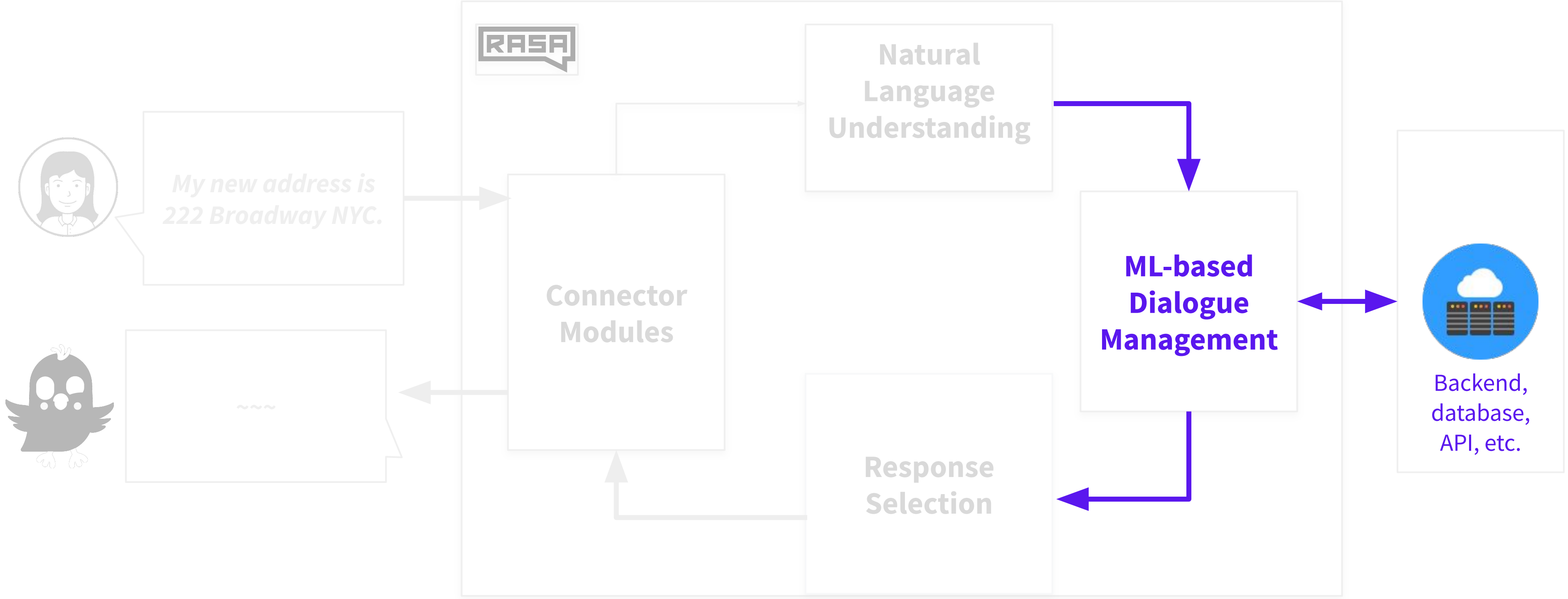


NLU: Output to Dialogue Management



Core Deep Dive

Dialogue Management (Core)



Training Data

Machine learning models require training data that the models can generalize from

NLU needs data in the form of examples for intents

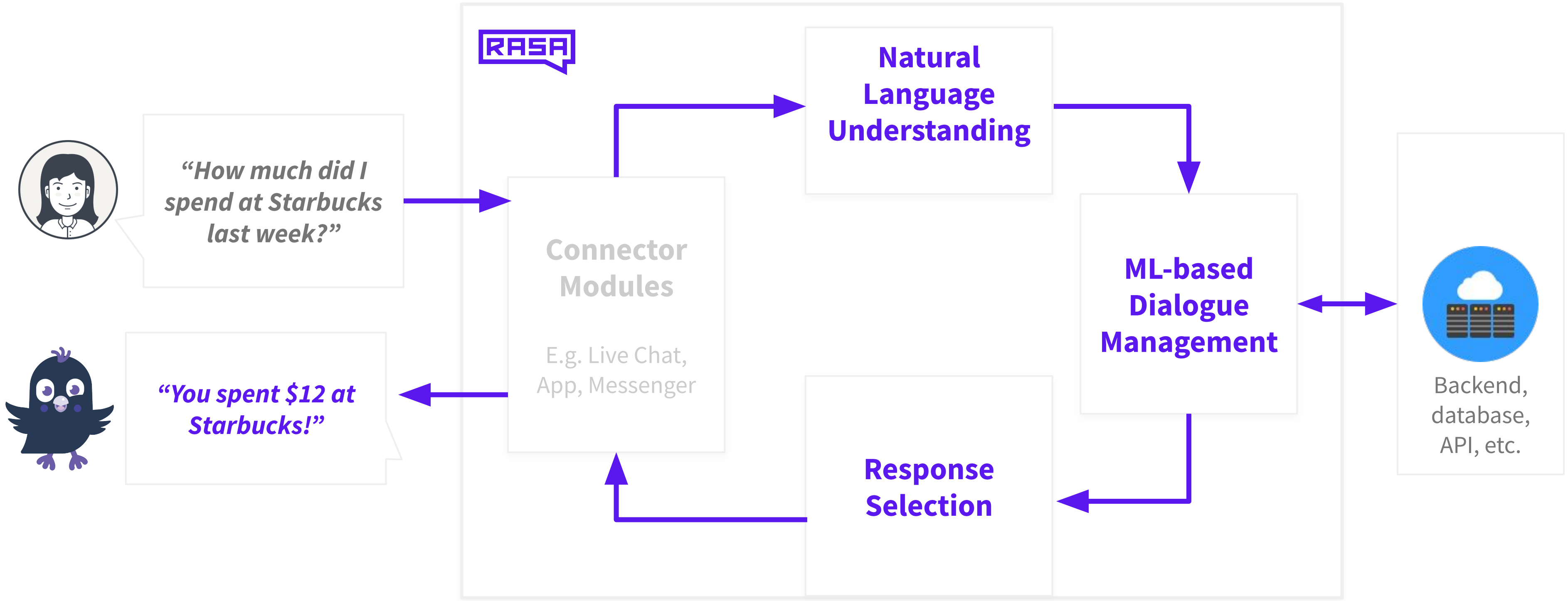
```
## intent:bot_challenge
- are you a bot?
- are you a human?
- am I talking to a bot?
- am I talking to a human?

## intent:i_like_food
- I like [apples](food)
- my friend likes [oranges](food)
- I'm a fan of [pears](food)
- do you like [coffee](food)
```

Dialogue management model needs data in the form of stories

```
## new to rasa at start
* how_to_get_started{"user_type":
"new"}
    - action_set_onboarding
    - slot{"onboarding": true}
    - utter_getstarted_new
    - utter_built_bot_before
* deny
    - utter_explain_rasa_components
    - utter_rasa_components_details
    - utter_ask_explain_nlucorex
```

RASA

Rasa Open Source Reviewed



Conversation Design

Testing

Conversation Design

Recap

Testing

Use the Rasa CLI to test your assistant

End to End Evaluation

Run through test conversations to make sure that both NLU and Core make correct predictions.

```
$ rasa test
```

NLU Evaluation

Split data into a test set or estimate how well your model generalizes using cross-validation.

```
$ rasa test nlu -u data/nlu.md --config config.yml --cross-validation
```

Core Evaluation

Evaluate your trained model on a set of test stories and generate a confusion matrix.

```
$ rasa test core --stories test_stories.md --out results
```

Interactive Learning: Talk to your bot yourself

- **Correct** your bot’s predictions as you go

- Save conversations as **training stories** or **E2E stories**

Run `rasa test` locally when building a minimum viable assistant

- Test your model after training to make development more productive and reliable

```
#### This file contains tests to evaluate that your bot behaves as expected.
#### If you want to learn more, please see the docs: https://rasa.com/docs/ra

## happy path 1
* greet: hello there!
  - utter_greet
* mood_great: amazing
  - utter_happy

## happy path 2
* greet: hello there!
  - utter_greet
* mood_great: amazing
  - utter_happy
* goodbye: bye-bye!
  - utter_goodbye

## sad path 1
* greet: hello
  - utter_greet
* mood_unhappy: not good
  - utter_cheer_up
  - utter_did_that_help
* affirm: yes
  - utter_happy
```

```
Your Rasa model is trained and saved at '/Users/ty/Documents/product_mgmt/docs/rasa/tem
(rasa_env) BERMB00017:temp ty$ rasa test
Processed Story Blocks: 100%|
2020-03-24 14:24:31 INFO     rasa.core.test  - Evaluating 7 stories
Progress:
100%|
2020-03-24 14:24:32 INFO     rasa.core.test  - Finished collecting predictions.
2020-03-24 14:24:32 INFO     rasa.core.test  - Evaluation Results on END-TO-END level:
2020-03-24 14:24:32 INFO     rasa.core.test  -  Correct:          7 / 7
2020-03-24 14:24:32 INFO     rasa.core.test  -  F1-Score:         1.000
2020-03-24 14:24:32 INFO     rasa.core.test  -  Precision:        1.000
2020-03-24 14:24:32 INFO     rasa.core.test  -  Accuracy:         1.000
2020-03-24 14:24:32 INFO     rasa.core.test  -  In-data fraction: 0.943
2020-03-24 14:24:32 INFO     rasa.core.test  - Evaluation Results on ACTION level:
2020-03-24 14:24:32 INFO     rasa.core.test  -  Correct:          35 / 35
2020-03-24 14:24:32 INFO     rasa.core.test  -  F1-Score:         1.000
2020-03-24 14:24:32 INFO     rasa.core.test  -  Precision:        1.000
2020-03-24 14:24:32 INFO     rasa.core.test  -  Accuracy:         1.000
2020-03-24 14:24:32 INFO     rasa.core.test  -  In-data fraction: 0.943
2020-03-24 14:24:32 INFO     rasa.core.test  -  Classification report:
                     precision    recall  f1-score   support
```