



Rasa Certification Workshop

Mady Mantha, Juste Petraityte, Karen White

The Rasa Team



Mady Mantha
Sr. Technical Evangelist



Juste Petraityte
Head of Developer Relations



Karen White
Developer Marketing
Manager

Setting up the environment

Setup and Installation checklist:

Before you begin, you'll need:

- IDE or Text editor
 - Python 3.6, 3.7, or 3.8
- ```
python --version
```

If you're a Windows user, we recommend also installing:

- Anaconda
- Microsoft C++ Build tools

Setting up the project:

- Create a new project folder
- Start a new virtual environment
- Financial Demo bot
  - Clone a local copy in the project folder
  - Install dependencies (including Rasa Open Source)
- Run the bot!
- Install ngrok

## Installing Rasa, by operating system

The process to install Rasa is a little different, depending on which operating system you use. Choose the appropriate option:

**Mac?**

**Keep watching - we'll demo installation on Mac during the workshop**

**Windows?**

**Pause the video and follow the instructions on [YouTube](#) to create your virtual environment**

**Linux?**

**Pause the video and follow the setup instructions in the Rasa docs**

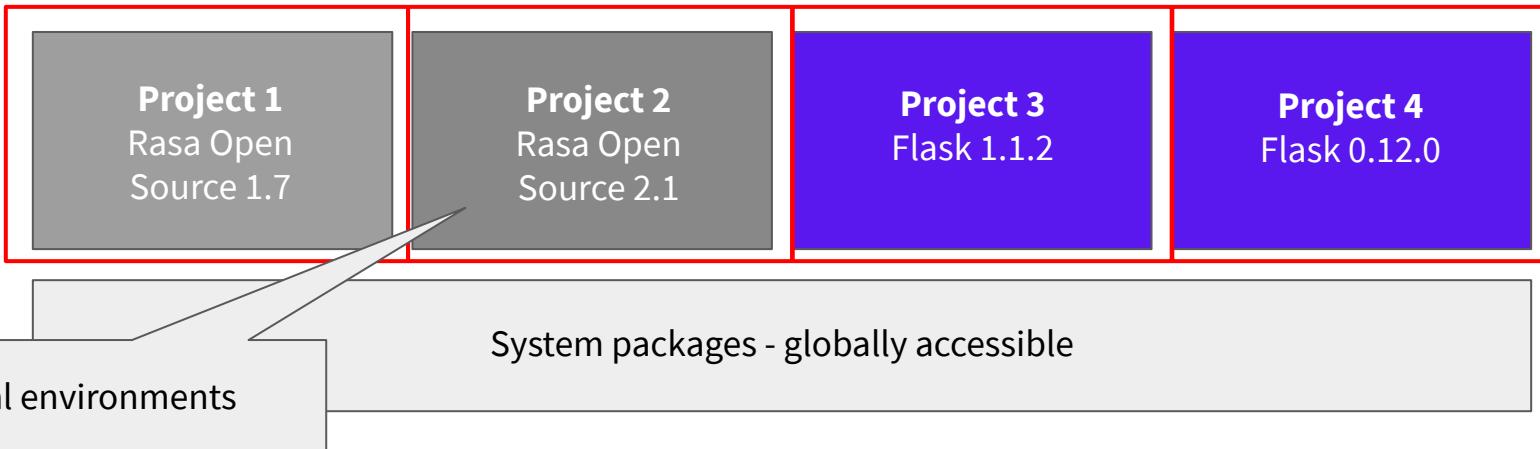
Follow the Windows YouTube video until the **2:30** second mark

## Virtual Environments

Using Windows?  
Follow the installation video for  
Windows, using Anaconda

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. For Windows, we recommend Anaconda.



## Creating the virtual environment - Mac

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`
4. Deactivate your virtual environment:  
`deactivate`

## Clone 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. Clone the repository:

git clone

<https://github.com/RasaHQ/financial-demo.git>

action\_listen (1.00)

How much did I spend at Amazon last week?

search\_transactions["vendor\_name":"Amazon","time":"2020-11-02T00:00:00.000-08:00"] (0.99)

slot["time":"2020-11-02T00:00:00.000-08:00"]

slot["vendor\_name":"Amazon"]

transact\_search\_form (1.00)



Searching transactions with Amazon  
between Monday Nov 02, 2020 and Monday  
Nov 09, 2020...



I found 5 transactions with Amazon totalling  
\$152.54.

form{"name":"transact\_search\_form"}

## Install project dependencies

Install the dependencies in your virtual environment:

```
cd financial-demo
```

```
pip install -r requirements.txt
```

3 lines (3 sloc) | 71 Bytes

```
1 -r actions/requirements-actions.txt
2 rasa[spacy]~=2.1.0
3 rasa-sdk~=2.1.1
```

## Installing pre-trained models

Certain types of entities require A LOT of training data for your model to recognize. For example:

- Names of people
- Names of places
- Numbers
- Email addresses
- Amounts of money
- Time periods

Instead of giving our model hundreds of training examples so it can learn to extract these types of entities on its own, we can use a pre-trained model that's already been trained for the task.

## 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:

```
python3 -m spacy link en_core_web_md en
```

## What's spaCy?

SpaCy is a pre-trained language model! Our bot uses it to extract names of people.

## Running Duckling

We've deployed Duckling to a server for you already, where it's accessible at this address:

`http://duckling.rasa.com:8000`

We'll point to this address instead of localhost in our `config.yml` file

```
- name: DucklingHTTPExtractor
 url: http://duckling.rasa.com:8000
 dimensions:
 - amount-of-money
 - time
 - number
```

## What's duckling?

Duckling is a pre-trained model we'll use to extract amounts of money, numbers, and dates.

It runs as a separate service, in its own container or server.

## Start the assistant



1. Train the model

```
rasa train
```

2. Start the Rasa Open Source server

```
rasa shell
```

3. Start the action server (in a new terminal window)

```
rasa run actions --port 5056
```

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?”

## Installing ngrok

### OSX

```
brew cask install ngrok
```

Or download ngrok from  
<https://ngrok.com/download>

### Windows/Linux

download ngrok from  
<https://ngrok.com/download>



If you installed using Homebrew, ngrok is accessible globally. If you manually downloaded the file, you'll need to either add ngrok to your \$PATH or supply the file path to the place you unzipped it every time you start ngrok

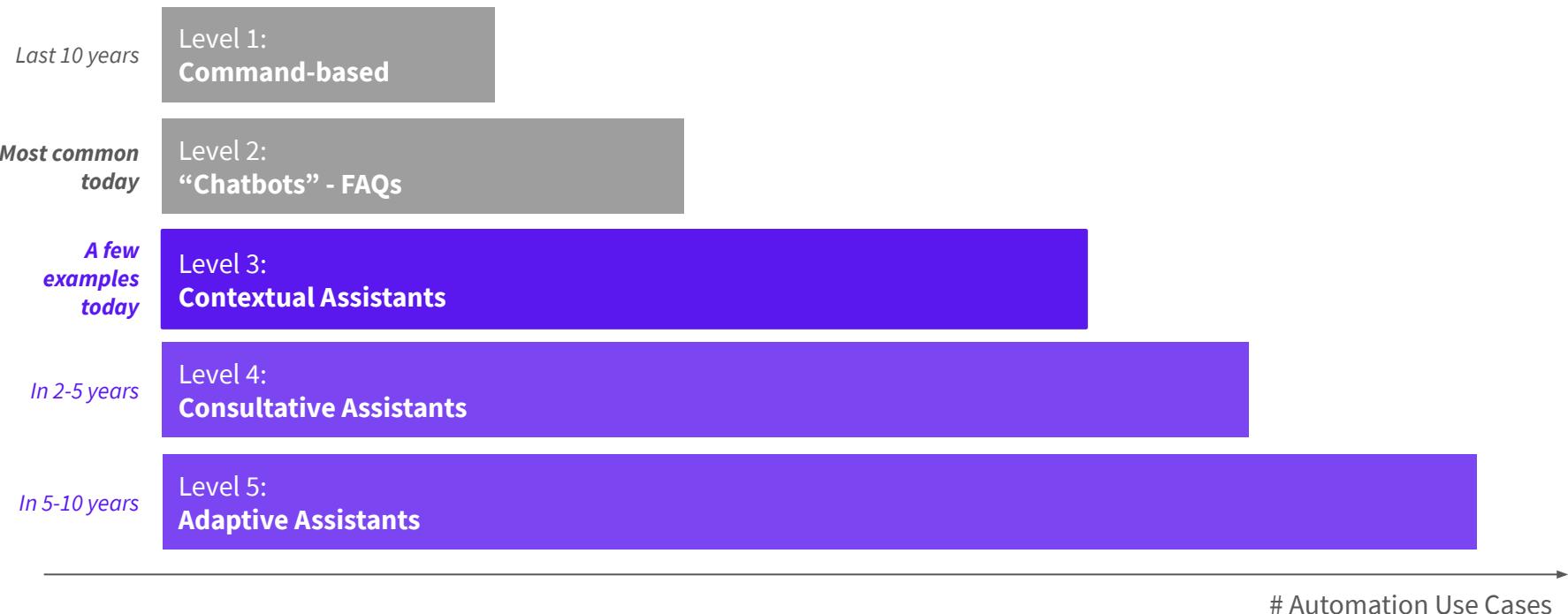
# Intro to Rasa

## Day 1 Roadmap

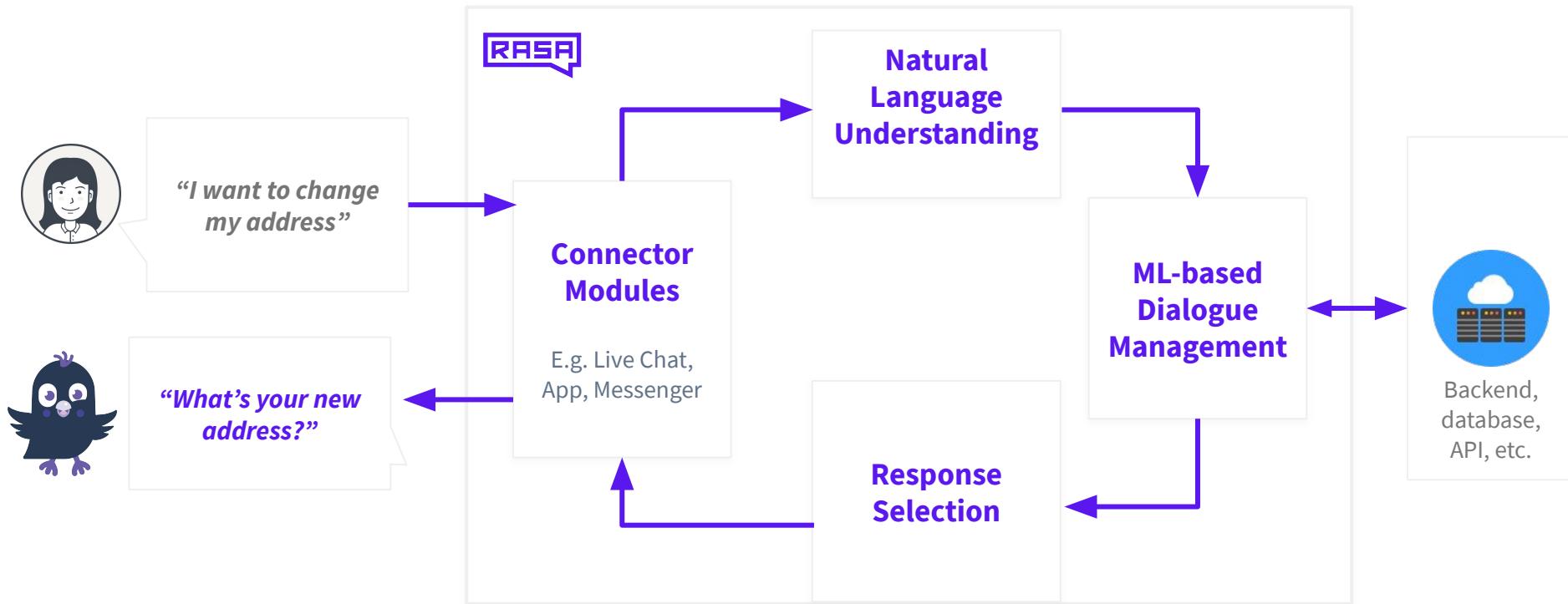
- First half: talk
- Second half: we'll code together
- Recap: talk, with some time for questions

## 5 LEVELS OF CONVERSATIONAL AI MATURITY MODEL

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

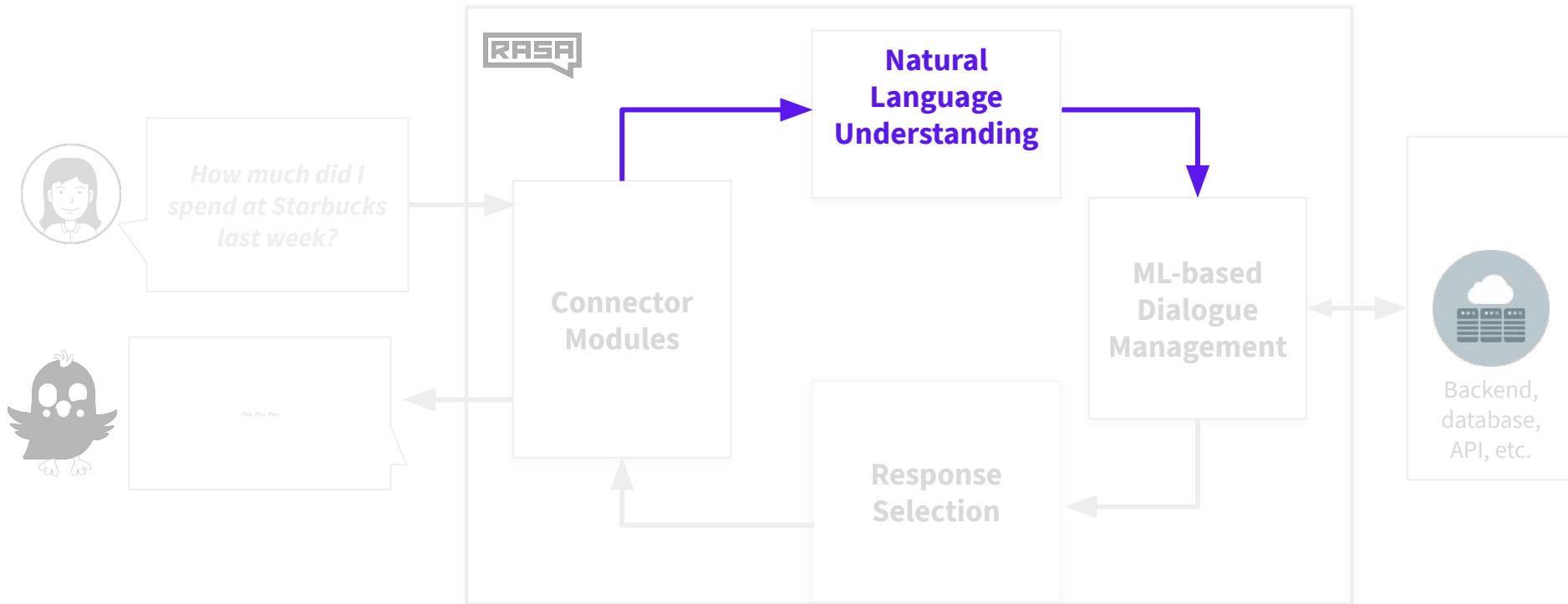


# Rasa Open Source



# NLU & Dialogue Management

# Natural Language Understanding (NLU)



# Natural Language Understanding (NLU)

Goal: Extract structured information from messages

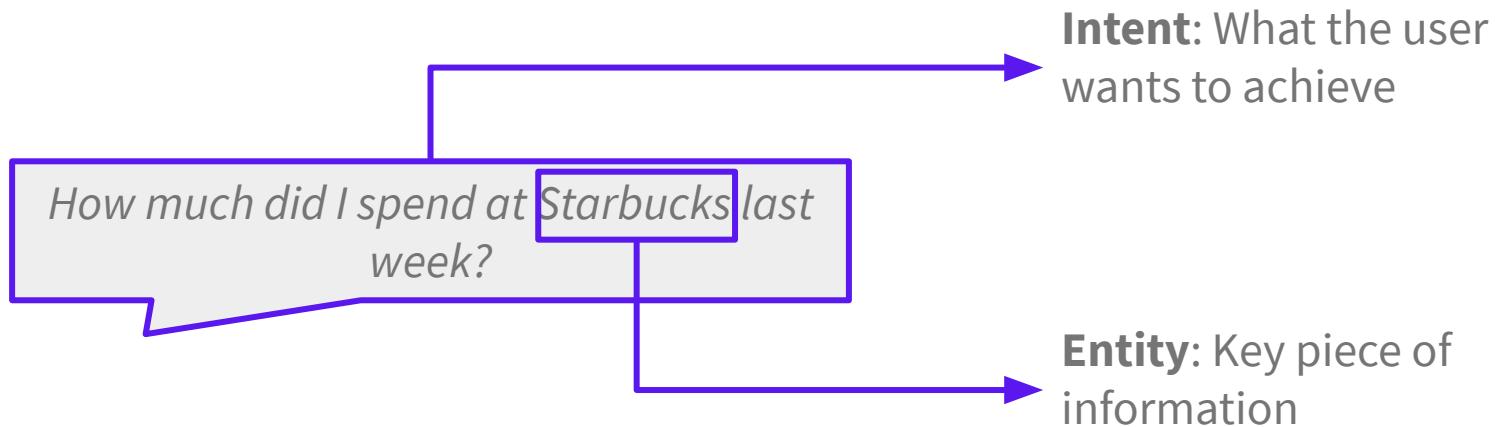
*How much did I spend  
at Starbucks last  
week?*



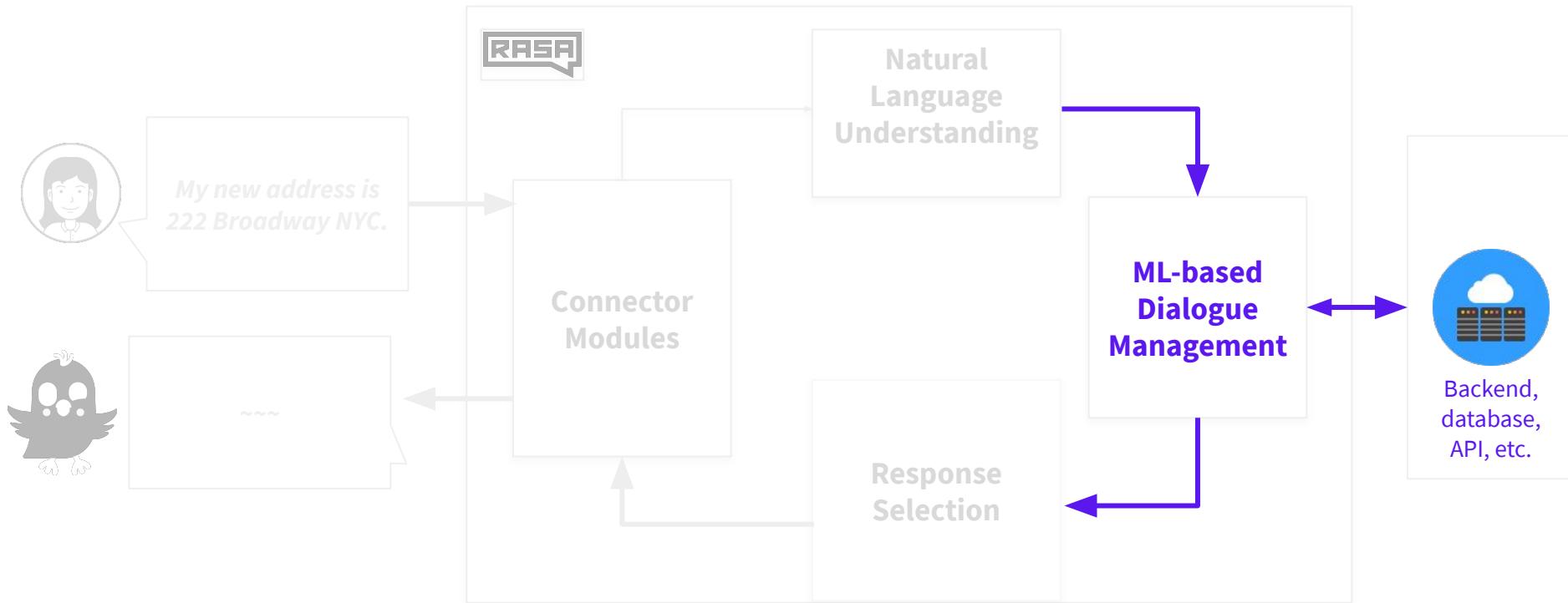
```
{...
 "intent": {
 "name": "search_transactions",
 "confidence": 0.96
 },
 "entities": [
 {
 "entity": "vendor_name",
 "value": "Starbucks",
 ...
 }
]
}
```

## Intents and Entities

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

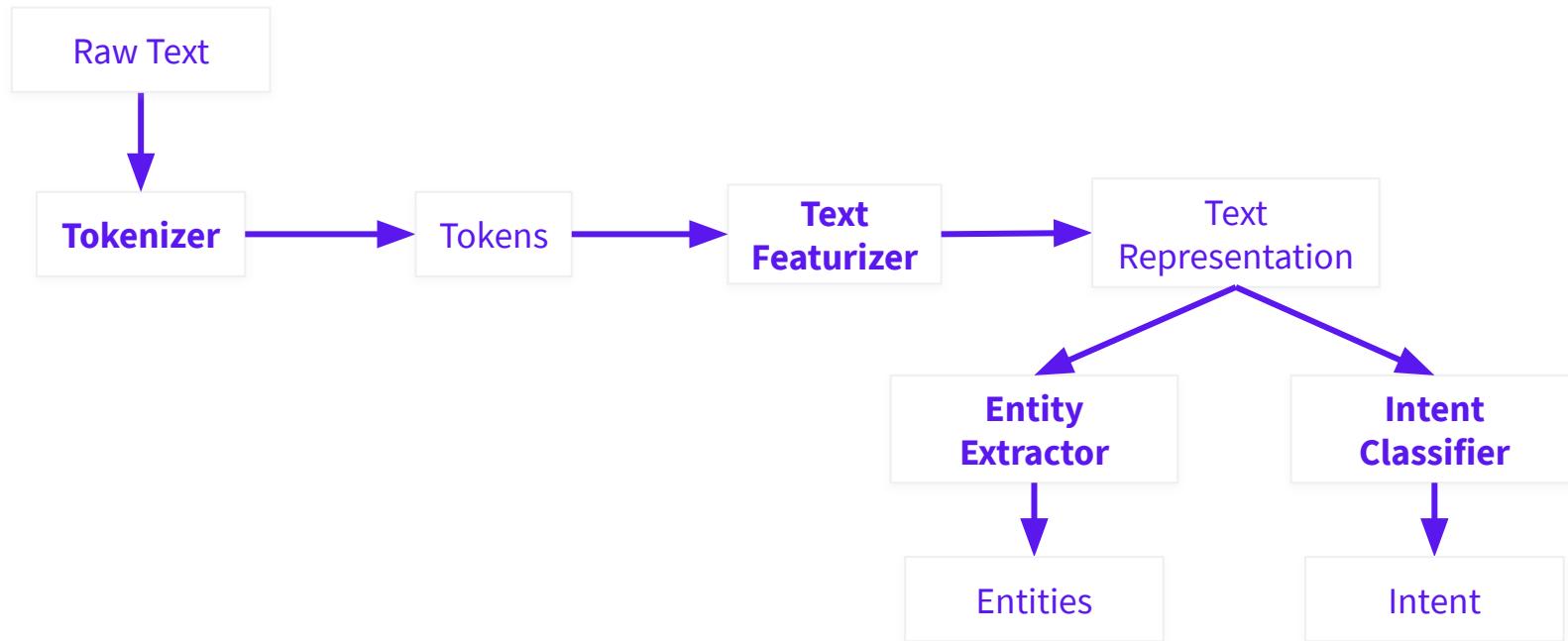


# Dialogue Management

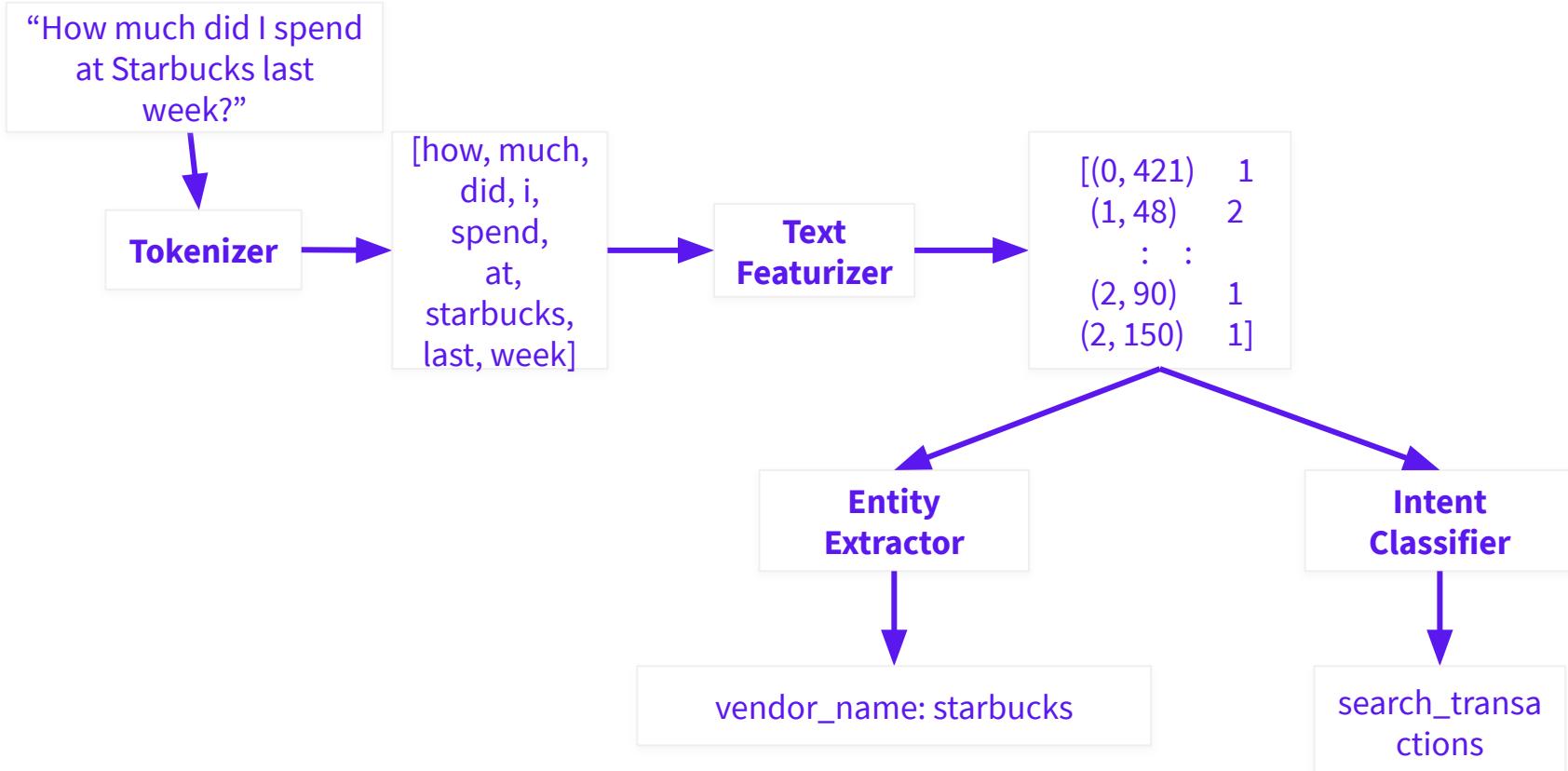


## NLU Pipeline: Input and Output

Many components take the output of earlier components as input

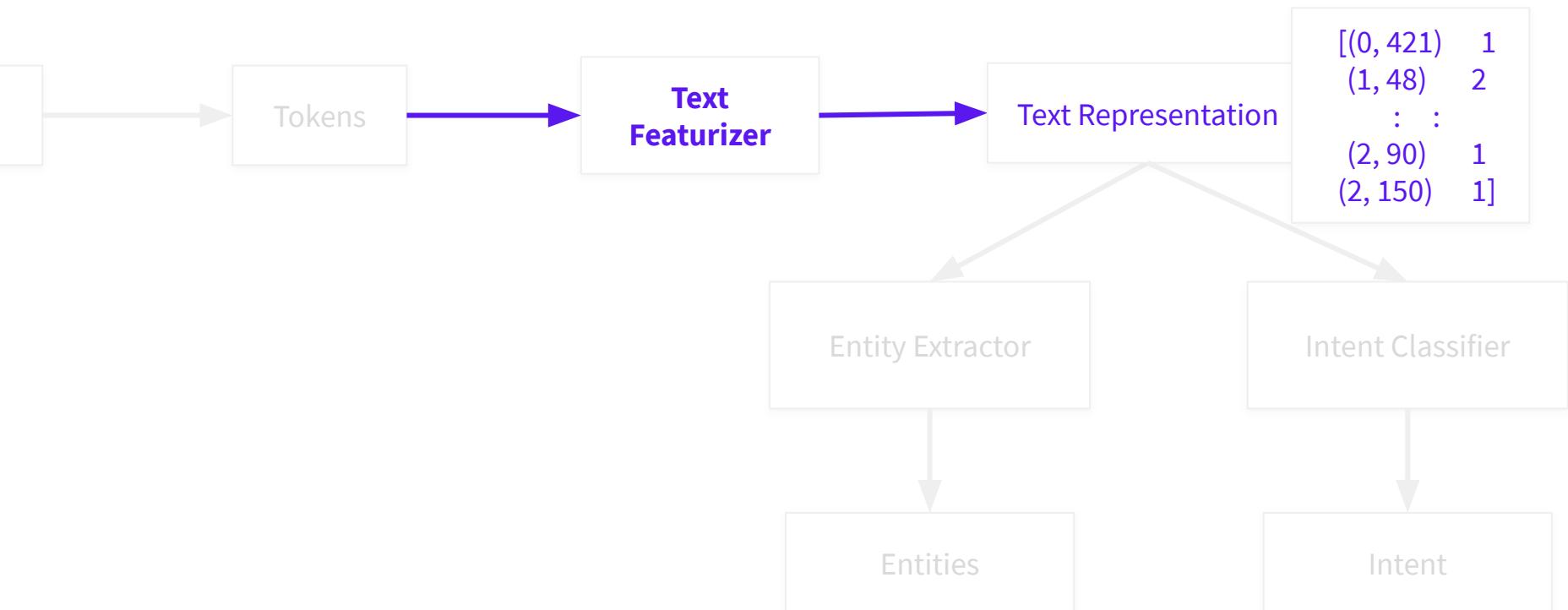


## NLU Pipeline: Example



## “Text Representation”

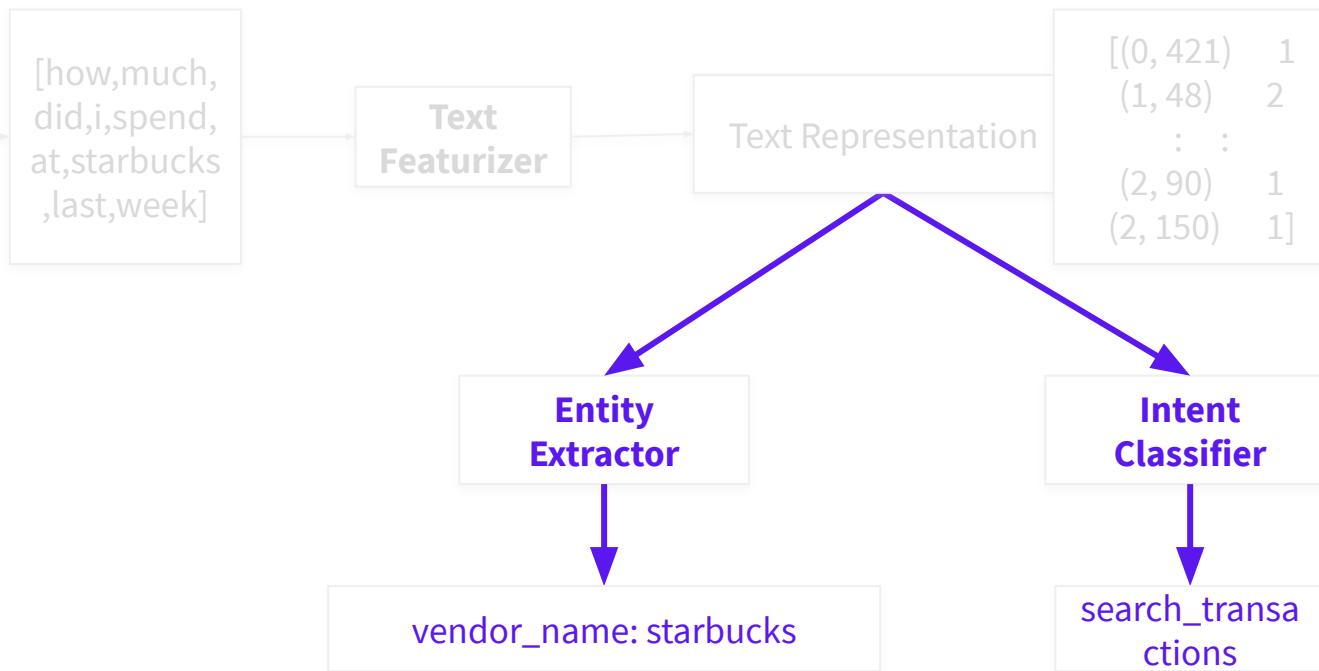
How do words turn into numbers?



# Intent Classification & Entity Extraction

did I  
bucks  
?”

er



## Text Representation: Word Vectors

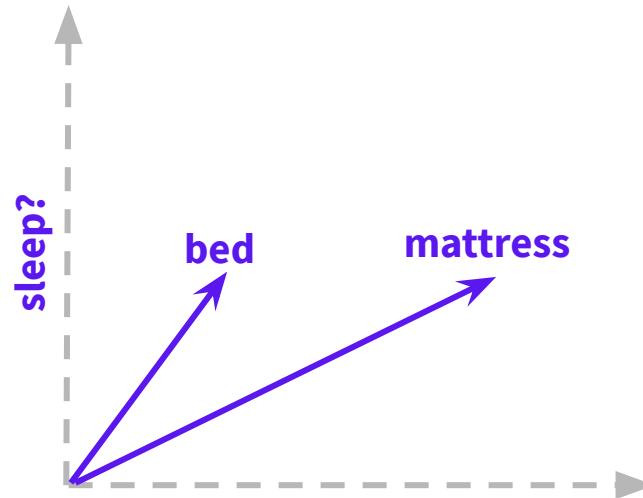
“Judge a Word by the Company it Keeps” — John R. Firth

“... 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:

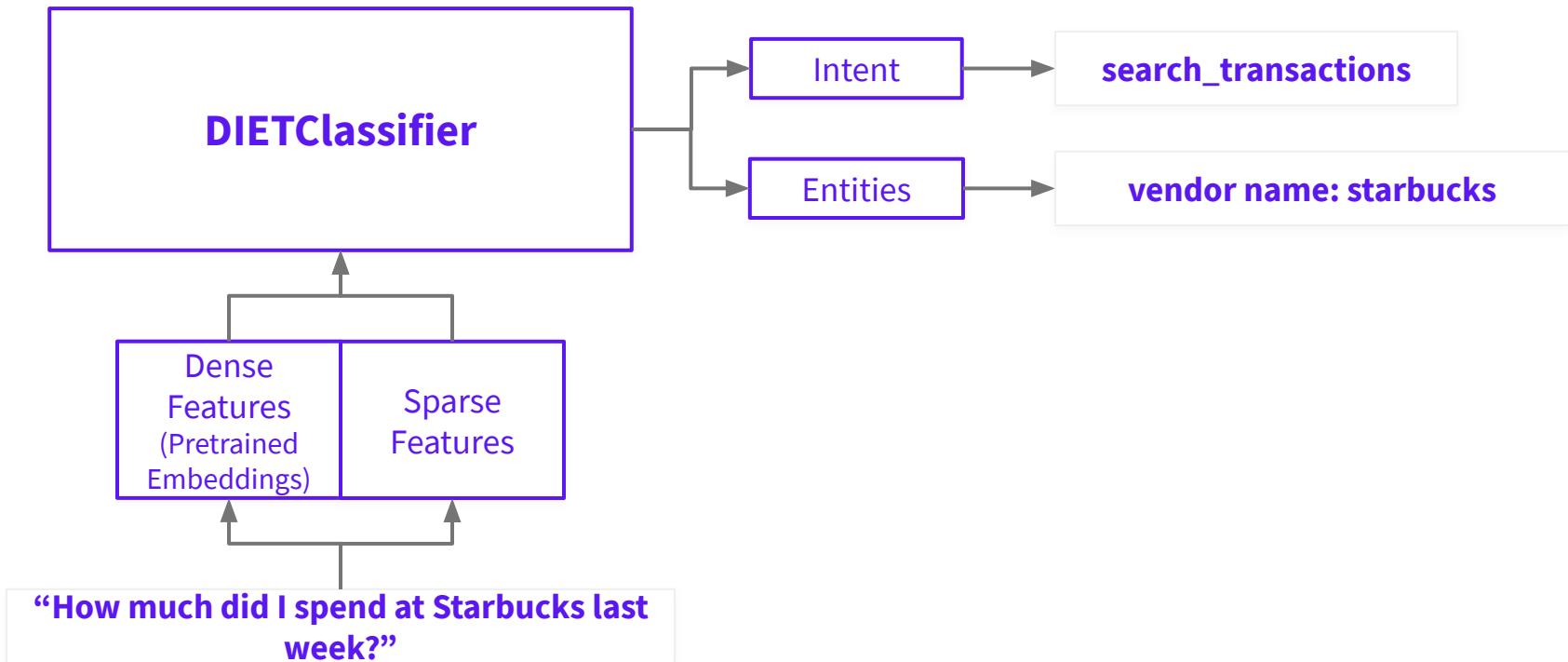
- One feature vector per input (whole message)
- Order of words not captured

### Sequence model:

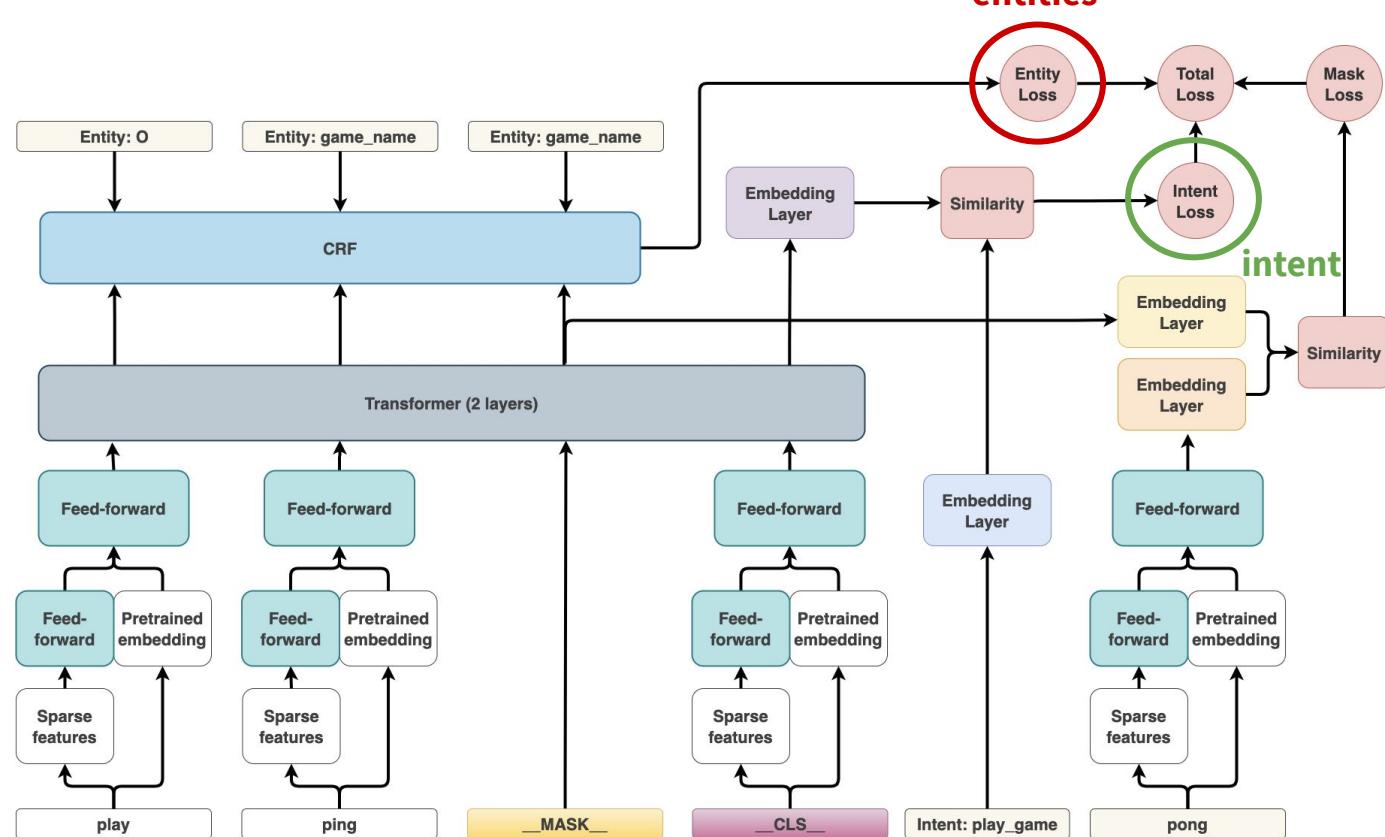
- One feature vector per token (word) & feature vector for whole message
- Word order captured

# DIETClassifier: Intent Classification & Entity Extraction

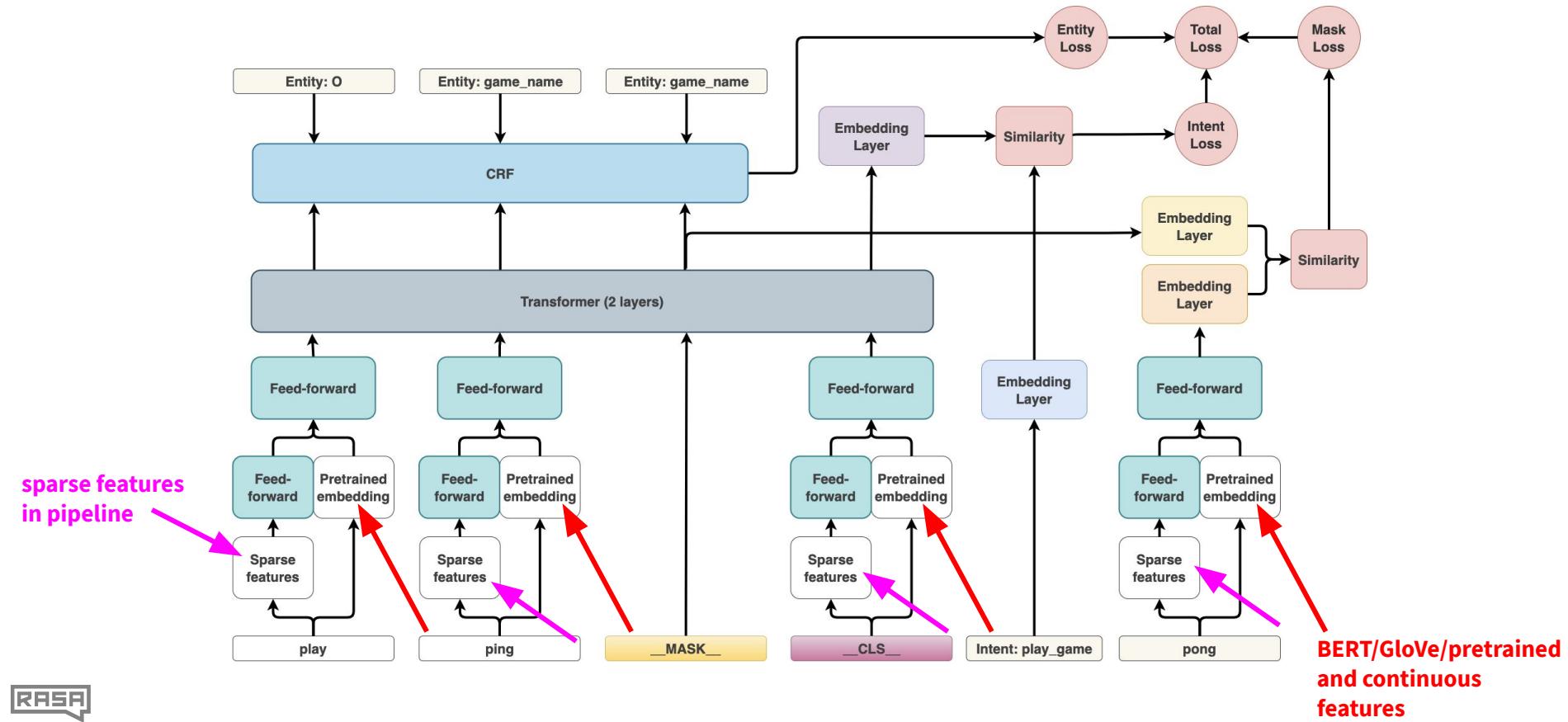
The Short Story



# DIET: Intent Classification & Entity Extraction



# Feature: Customize What Features Go In!



## How to use DIET in your Rasa project

Here's an example config.yml

Before the DIET model, you can specify any featurizer.

In our experiments, we use:

- Sparse features (aka no pre-trained model)
- GloVe (word vectors)
- BERT (large language model)
- ConveRT (pre-trained encoder for conversations)

```
language: en
pipeline:
- name: ConveRTTokenizer
- name: ConveRTFeaturizer
- name: DIETClassifier
 hidden_layers_sizes:
 text: [256, 128]
 label: []
 intent_classification: True
 entity_recognition: False
 use_masked_language_model: False
 BILOU_flag: False
 number_of_transformer_layers: 0
```

## Experiments on the NLU-benchmark dataset

- Repo is [on github](#)
- Domain: human-robot interaction (smart home setting)
- 64 different intents
- 54 different entity types
- ~26k labelled examples

Previous state of the art:

- [HERMIT NLU](#) (Vanzo, Bastianelli, and Lemon @ SIGdial 2019)
- uses ELMo embeddings

## Result 1: DIET outperforms SotA even without any pretrained embeddings

Previous state of the art: intent: 87.55 entities: 84.74

| sparse | dense   | mask loss | Intent            | Entities          |
|--------|---------|-----------|-------------------|-------------------|
| ✓      | ✗       | ✗         | 87.10±0.75        | 83.88±0.98        |
| ✓      | ✗       | ✓         | 88.19±0.84        | 85.12±0.85        |
| ✗      | GloVe   | ✗         | 89.20±0.90        | 84.34±1.03        |
| ✓      | GloVe   | ✗         | 89.38±0.71        | 84.89±0.91        |
| ✗      | GloVe   | ✓         | 88.78±0.70        | 85.06±0.84        |
| ✓      | GloVe   | ✓         | 89.13±0.77        | 86.04±1.09        |
| ✗      | BERT    | ✗         | 87.44±0.92        | 84.20±0.91        |
| ✓      | BERT    | ✗         | 88.46±0.88        | 85.26±1.01        |
| ✗      | BERT    | ✓         | 86.92±1.09        | 83.96±1.33        |
| ✓      | BERT    | ✓         | 87.45±0.67        | 84.64±1.31        |
| ✗      | ConveRT | ✗         | 89.76±0.98        | 86.06±1.38        |
| ✓      | ConveRT | ✗         | 89.89±0.43        | <b>87.38±0.64</b> |
| ✗      | ConveRT | ✓         | <b>90.15±0.68</b> | 85.76±0.80        |
| ✓      | ConveRT | ✓         | 89.47±0.74        | 86.04±1.29        |

## Result 2: GloVe embeddings perform better than BERT

| sparse | dense   | mask loss | Intent            | Entities          |
|--------|---------|-----------|-------------------|-------------------|
| ✓      | ✗       | ✗         | 87.10±0.75        | 83.88±0.98        |
| ✓      | ✗       | ✓         | 88.19±0.84        | 85.12±0.85        |
| ✗      | GloVe   | ✗         | 89.20±0.90        | 84.34±1.03        |
| ✓      | GloVe   | ✗         | 89.38±0.71        | 84.89±0.91        |
| ✗      | GloVe   | ✓         | 88.78±0.70        | 85.06±0.84        |
| ✓      | GloVe   | ✓         | 89.13±0.77        | 86.04±1.09        |
| ✗      | BERT    | ✗         | 87.44±0.92        | 84.20±0.91        |
| ✓      | BERT    | ✗         | 88.46±0.88        | 85.26±1.01        |
| ✗      | BERT    | ✓         | 86.92±1.09        | 83.96±1.33        |
| ✓      | BERT    | ✓         | 87.45±0.67        | 84.64±1.31        |
| ✗      | ConveRT | ✗         | 89.76±0.98        | 86.06±1.38        |
| ✓      | ConveRT | ✗         | 89.89±0.43        | <b>87.38±0.64</b> |
| ✗      | ConveRT | ✓         | <b>90.15±0.68</b> | 85.76±0.80        |
| ✓      | ConveRT | ✓         | 89.47±0.74        | 86.04±1.29        |

## Result 3: ConveRT embeddings perform best on the NLU-benchmark dataset

| sparse | dense   | mask loss | Intent            | Entities          |
|--------|---------|-----------|-------------------|-------------------|
| ✓      | X       | X         | 87.10±0.75        | 83.88±0.98        |
| ✓      | X       | ✓         | 88.19±0.84        | 85.12±0.85        |
| X      | GloVe   | X         | 89.20±0.90        | 84.34±1.03        |
| ✓      | GloVe   | X         | 89.38±0.71        | 84.89±0.91        |
| X      | GloVe   | ✓         | 88.78±0.70        | 85.06±0.84        |
| ✓      | GloVe   | ✓         | 89.13±0.77        | 86.04±1.09        |
| X      | BERT    | X         | 87.44±0.92        | 84.20±0.91        |
| ✓      | BERT    | X         | 88.46±0.88        | 85.26±1.01        |
| X      | BERT    | ✓         | 86.92±1.09        | 83.96±1.33        |
| ✓      | BERT    | ✓         | 87.45±0.67        | 84.64±1.31        |
| X      | ConveRT | X         | 89.76±0.98        | 86.06±1.38        |
| ✓      | ConveRT | X         | 89.89±0.43        | <b>87.38±0.64</b> |
| X      | ConveRT | ✓         | <b>90.15±0.68</b> | 85.76±0.80        |
| ✓      | ConveRT | ✓         | 89.47±0.74        | 86.04±1.29        |

## Result 4: DIET outperforms fine-tuning BERT

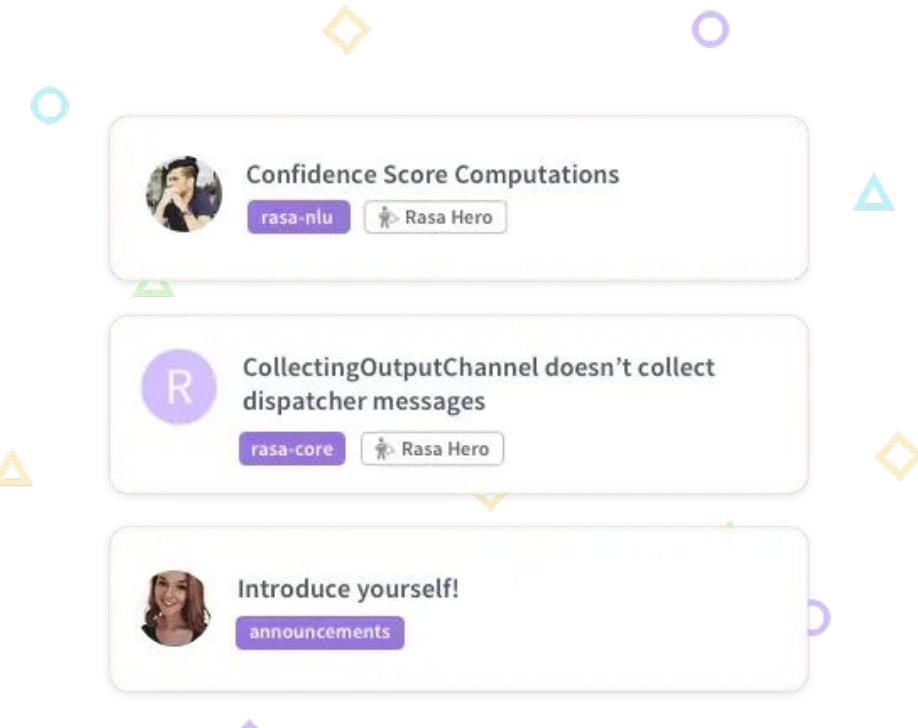
|                                  |    | Intent            | Entities          |
|----------------------------------|----|-------------------|-------------------|
| Fine-tuned<br>BERT               | F1 | <b>89.67±0.48</b> | <b>85.73±0.91</b> |
|                                  | R  | <b>89.67±0.48</b> | <b>84.71±1.28</b> |
|                                  | P  | <b>89.67±0.48</b> | <b>86.78±1.02</b> |
| sparse +<br>ConveRT <sup>†</sup> | F1 | <b>89.89±0.43</b> | <b>87.38±0.64</b> |
|                                  | R  | <b>89.89±0.43</b> | <b>87.15±0.97</b> |
|                                  | P  | <b>89.89±0.43</b> | <b>87.62±0.94</b> |

## Which featurizer is best depends on your dataset, so try different ones!

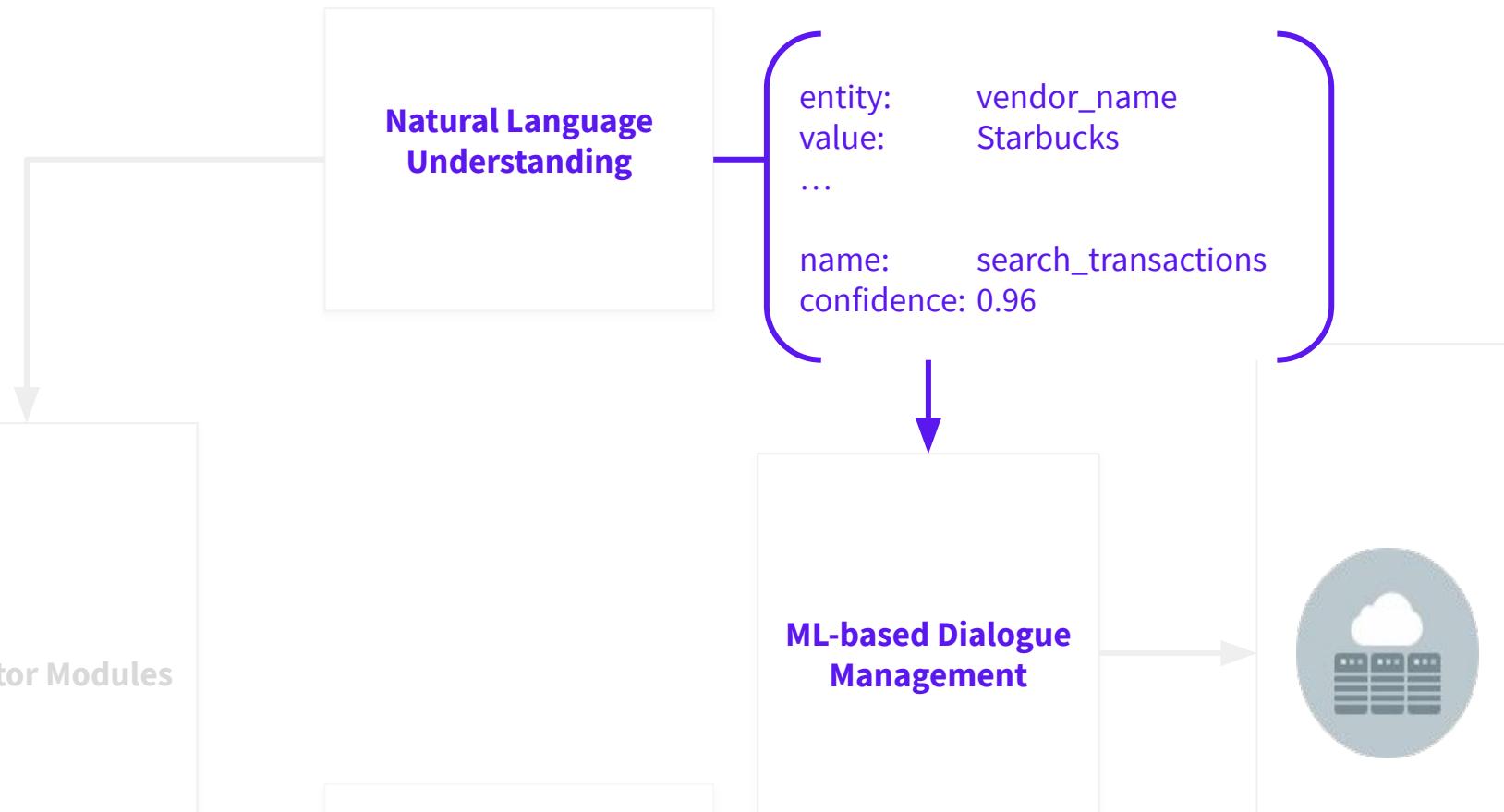
We don't believe in "one size fits all" machine learning

- We aim to provide sensible defaults and suggestions
- BUT even more important that these models are easy to customize

Share your results and compare notes with 8000+ developers at [forum.rasa.com](https://forum.rasa.com)

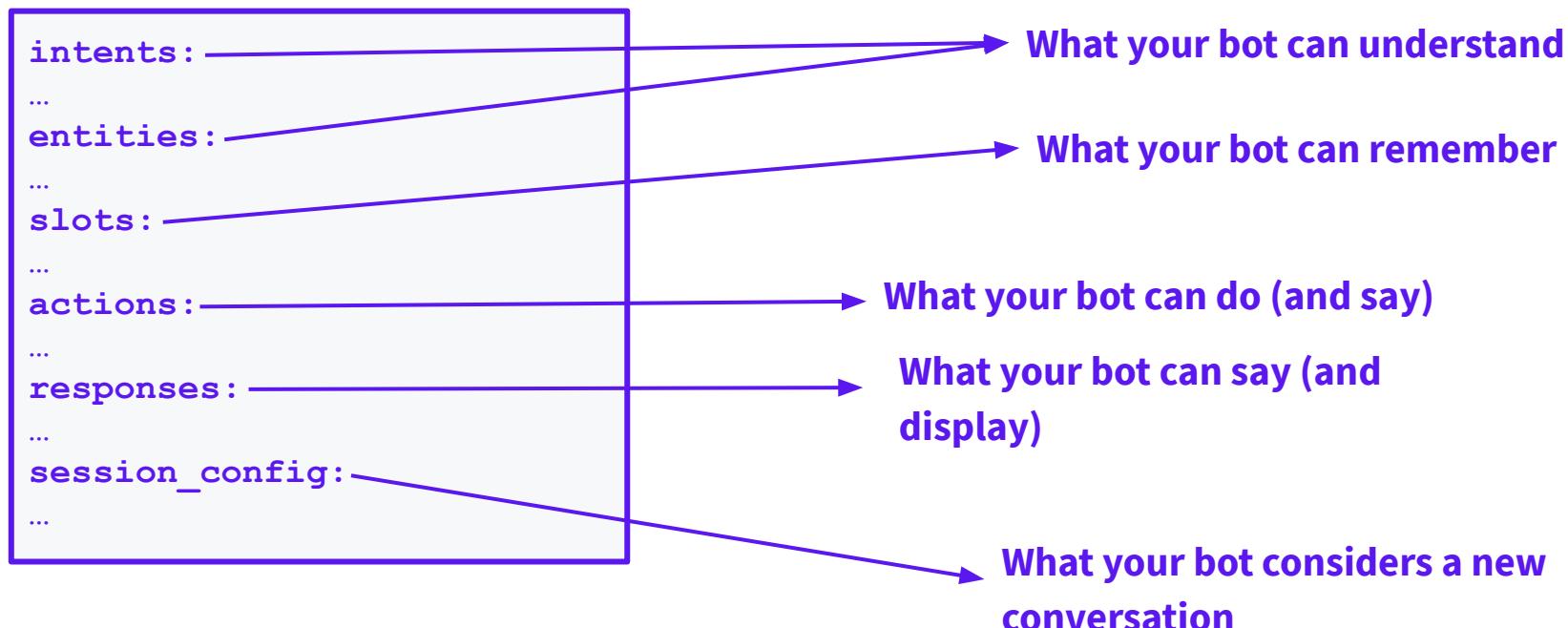


## NLU: Output to Dialogue Management



## Domain

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



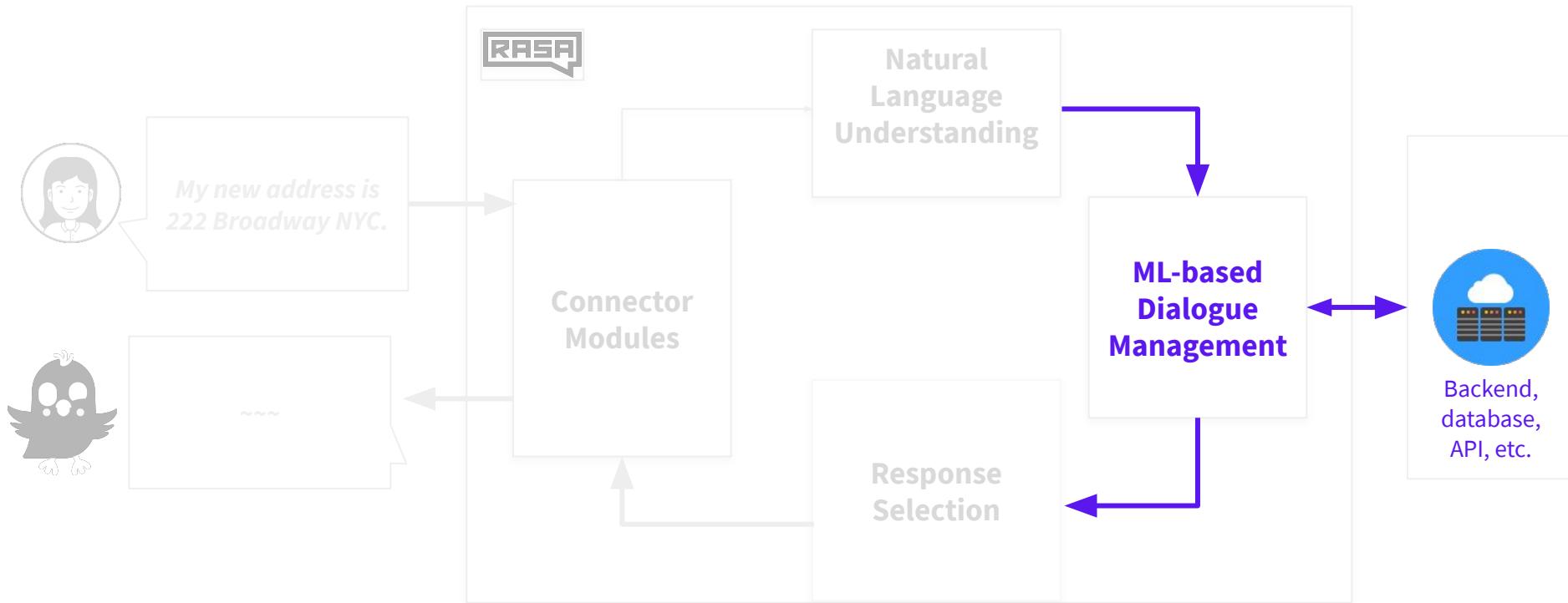
## Project setup: Files

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

Livecoding

**Let's create some intents and entities!**

# Dialogue Management



# Conversational AI is not easy

## Real conversations don't follow the happy path



## Conversational AI requires NLU and Dialogue management



Your total is \$15.50 - shall I charge the card you used last time?

Yes.



intent: affirm, 99%



Done. You should have your items tomorrow.



action: confirm, 87%

## Happy paths are already solved

---

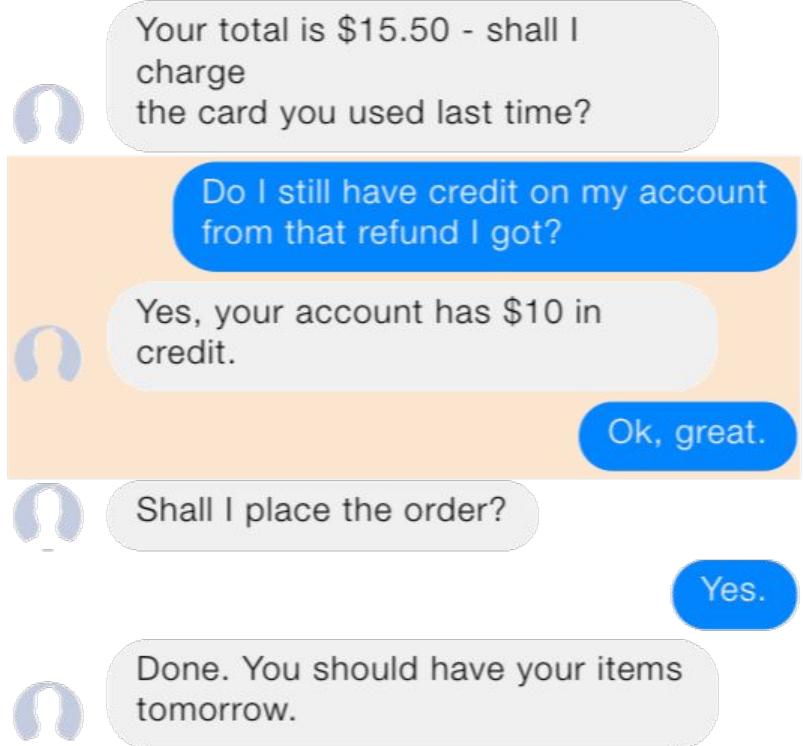
```
● ● ●

class CheckoutForm(FormAction):
 """Perform steps for checkout"""

 def name(self):
 return "checkout_form"

 @staticmethod
 def required_slots(tracker):
 return ["address", "card_number", "shipping_method"]
```

## You can't predict user behavior



## Users will interject and loop back to earlier topics

Your total is \$15.50 - shall I charge the card you used last time?

Do I still have credit on my account from that refund I got?

Yes, your account has \$10 in credit.

Ok, great.

Shall I place the order?

Yes.

Done. You should have your items tomorrow.

so did that use my credit?

## Real conversations don't follow the happy path

Your total is \$15.50 - shall I charge the card you used last time?

Do I still have credit on my account from that refund I got?

Yes, your account has \$10 in credit.

Wait, are you a bot?

Why yes I am! I may not be human, but I can help with lots of useful tasks 😊

Ok, great.

Shall I place the order?

Take the last thing I added out of my order. I changed my mind.

Okay, removing Reusable water bottle - Color blue

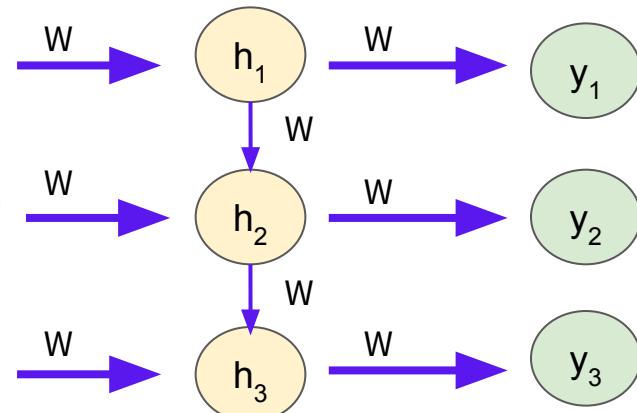
Cool, place the order now.

## People typically use a recurrent neural net (RNN) to model dialogue

Your total is \$15.50 - shall I charge the card you used last time?

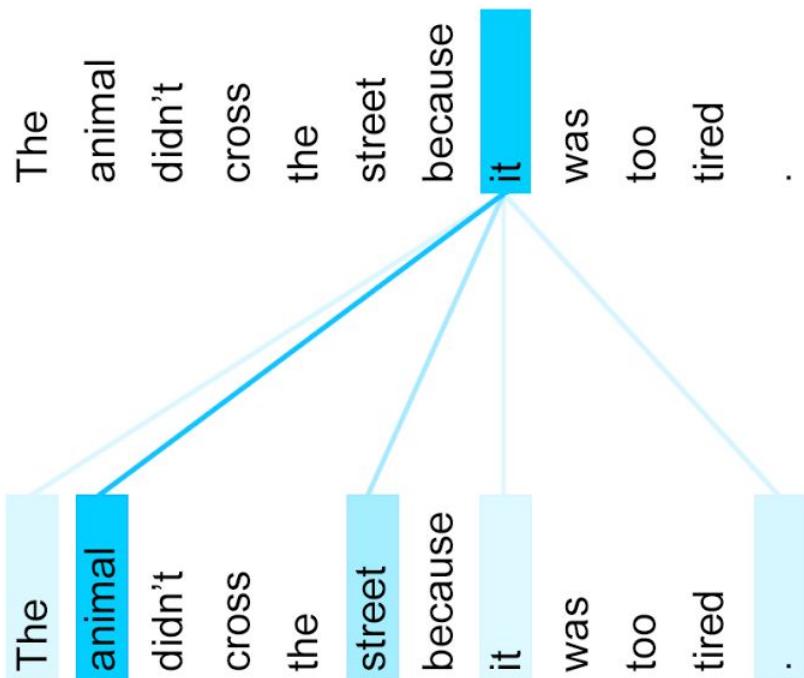
Yes.

Done. You should have your items tomorrow.

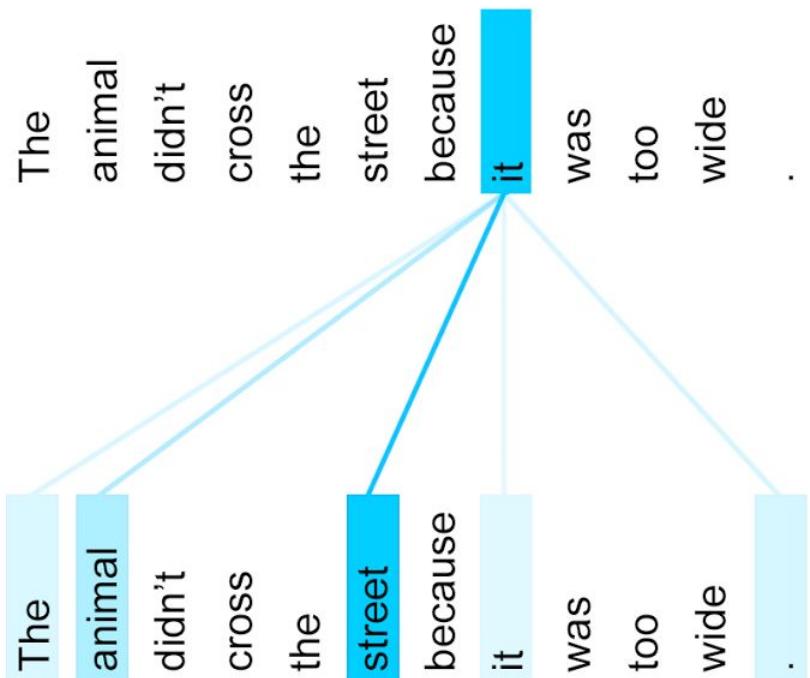


## But not all input should be treated equally

The animal didn't cross the street because it was too tired .



The animal didn't cross the street because it was too wide .



# Transformers (AKA self-attention) are now state of the art for many tasks

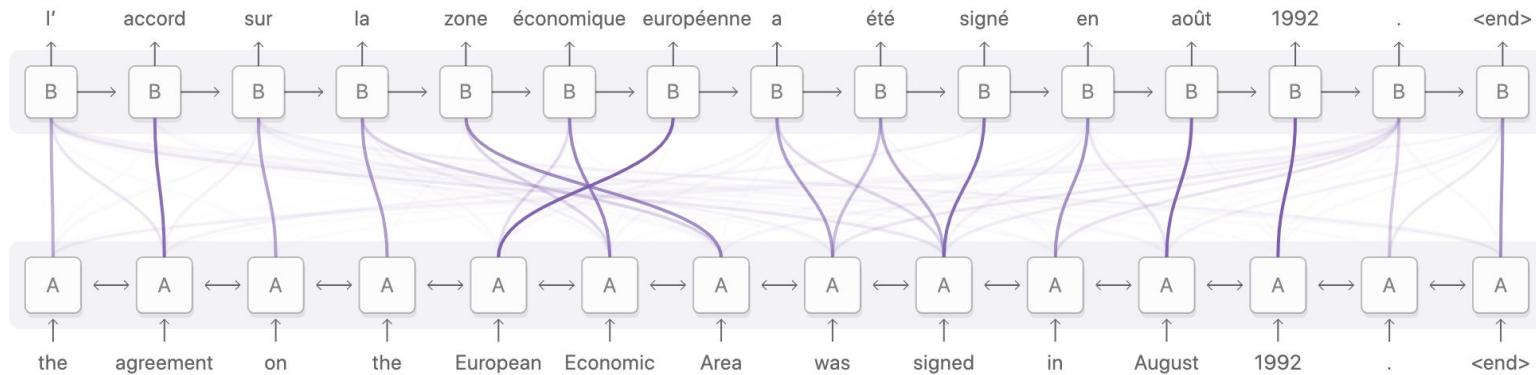
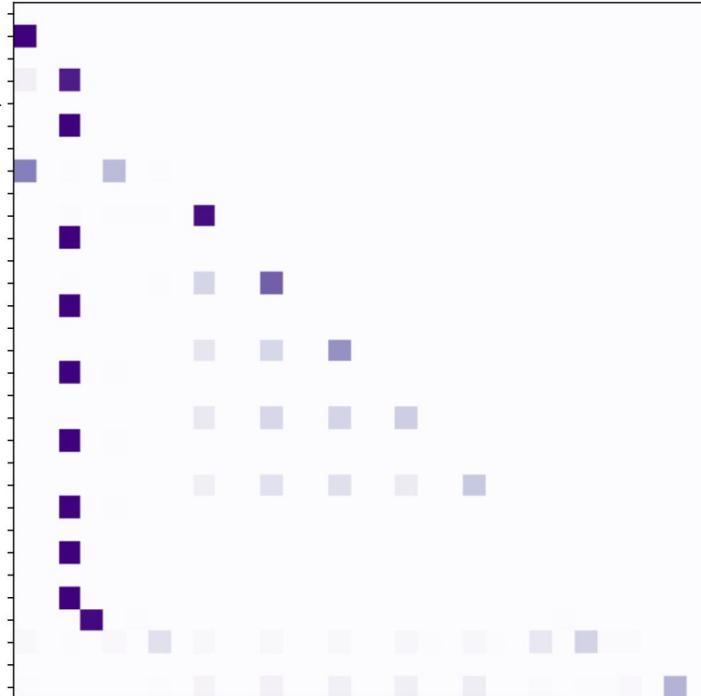


Diagram derived from Fig. 3 of Bahdanau, et al. 2014

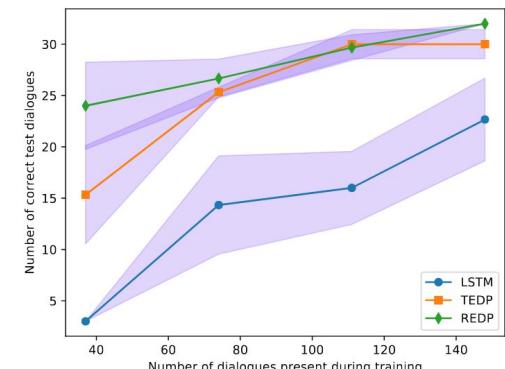
<https://distill.pub/2016/augmented-rnns/>

# We found out that the Transformer Embedding Dialogue policy can untangle sub-dialogues

```
* request_hotel
 - utter_ask_details
* inform{"enddate": "May 26th"}
 - utter_ask_startdate
* inform{"startdate": "next week"}
 - utter_ask_location
* inform{"location": "rome"}
 - utter_ask_price
* chitchat
 - utter_chitchat
 - utter_ask_price
* inform{"price": "expensive"}
 - utter_ask_people
* inform{"people": "4"}
 - utter_filled_slots
 - action_search_hotel
 - utter_suggest_hotel
* affirm
 - utter_happy
```



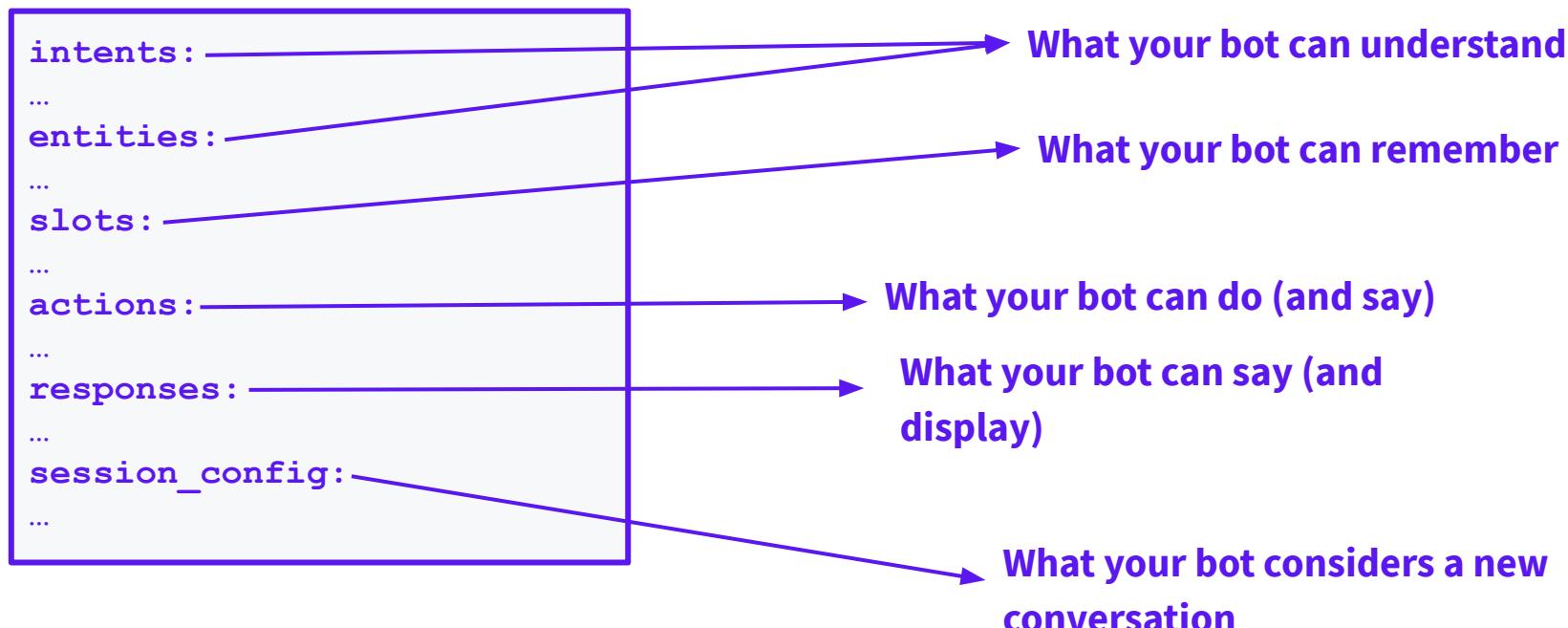
paper



# Configuration Files

## Domain

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



## Project setup: Files

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

## Conversation Driven Development (CDD)

### The challenge

When developing assistants, it's impossible to anticipate all of the things your users might say.

### The approach

A user-centric process: listening to your users and using those insights to improve your AI assistant.

Blog post: [Conversation-Driven Development](#)

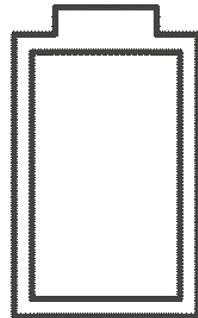
# Conversation Driven Development



Rasa X

**Time for a short break (10 minutes)**

# HUMANS RECHARGING



# Conversation Design

## Scope Conversation

How to get started with conversation design

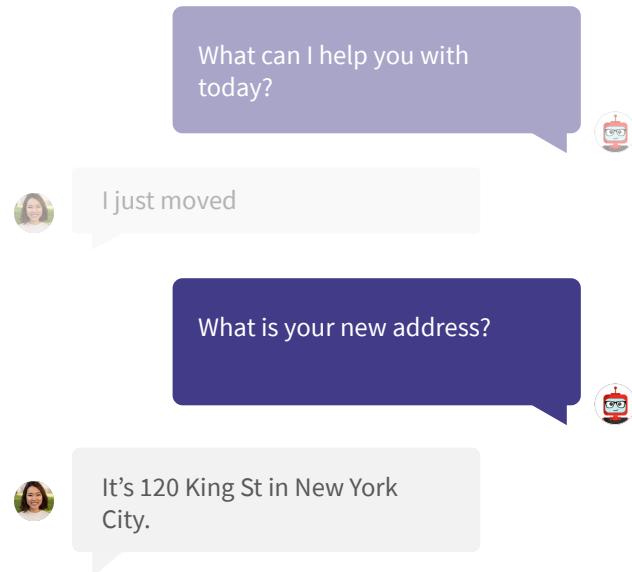
The assistant's purpose

Leverage the knowledge of domain experts

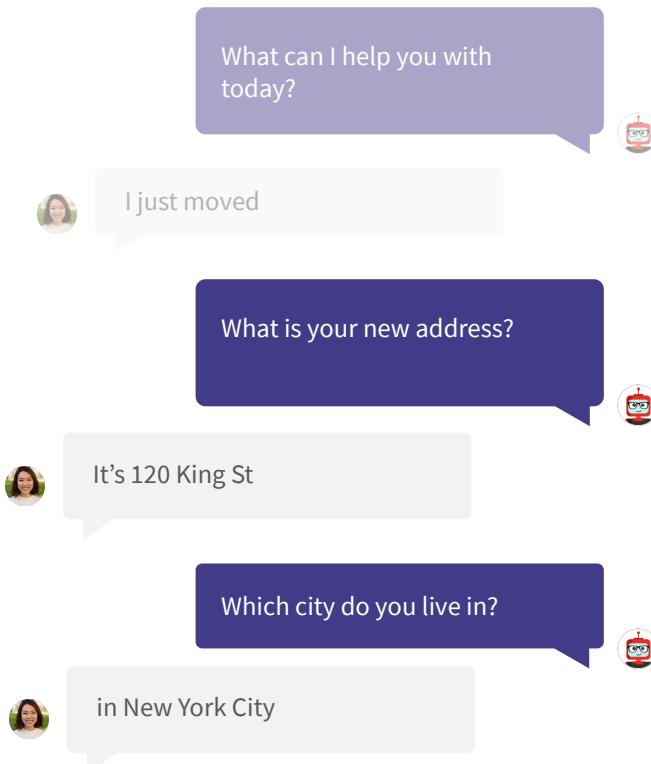
Common search queries

FAQs and wikis

## Initially: Conversations as simple question / answer pairs



## Real conversations are more complex: Rely on context



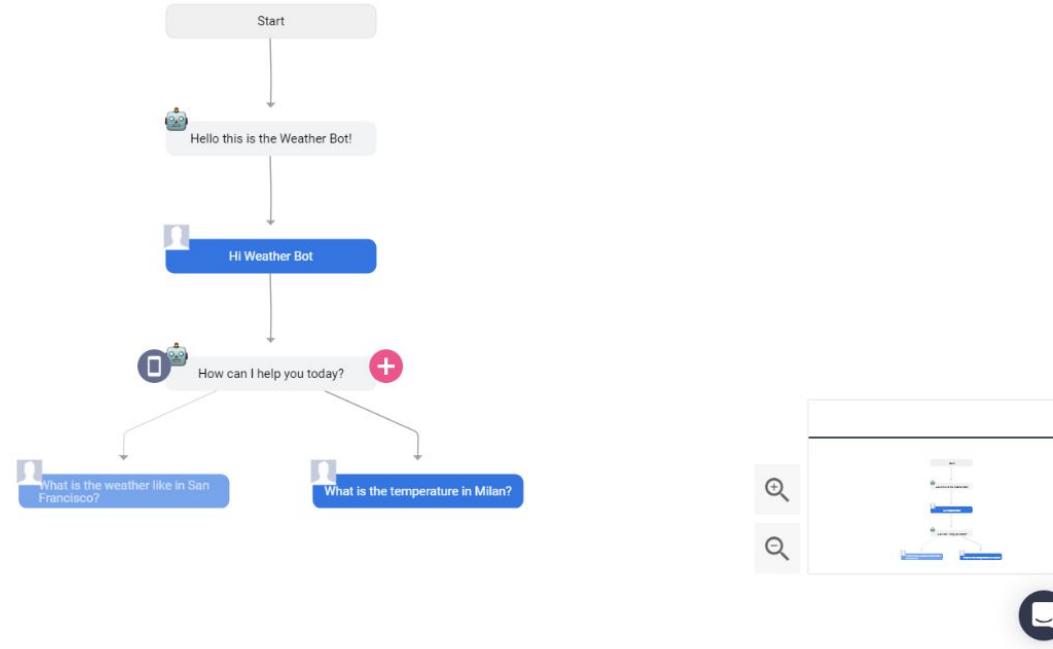
## Conversation Design

# Scope Conversation

  Showing all messages  Showing links

 Search message (Ctrl+F)

X



Source: Botsociety

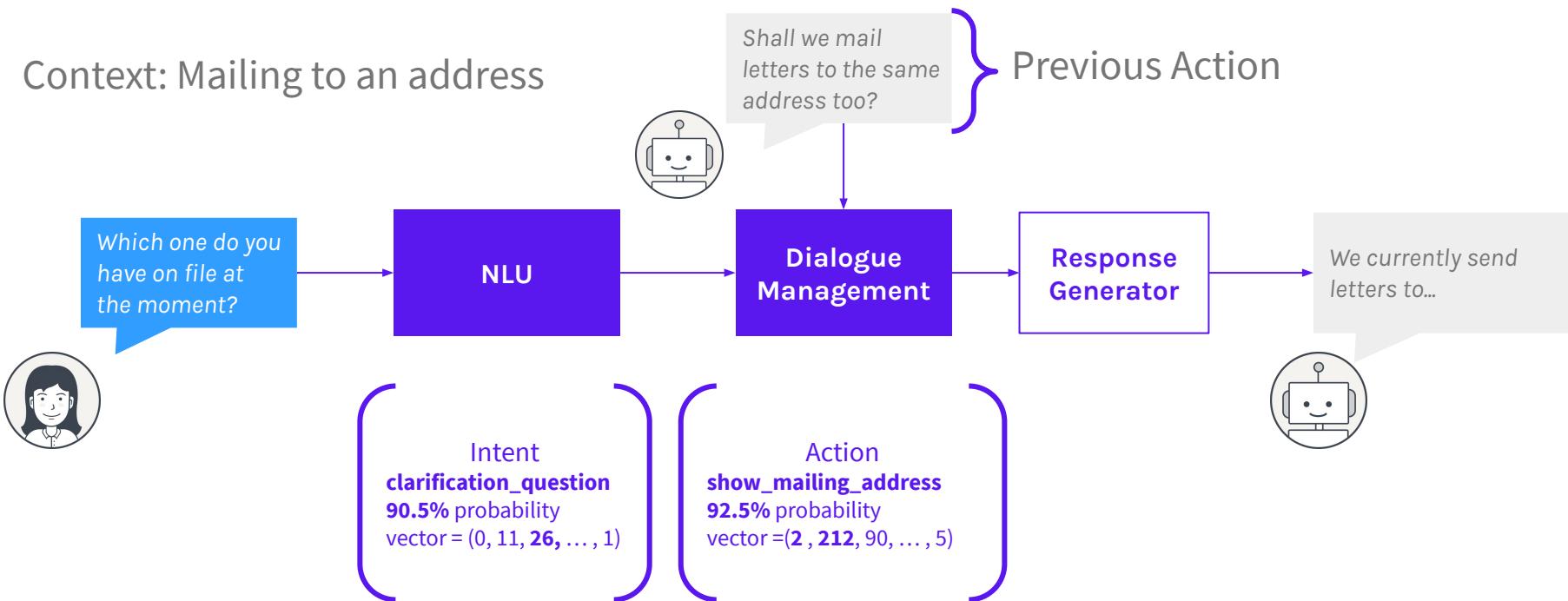
Livecoding

## Let's create some stories together!

- Check account balance

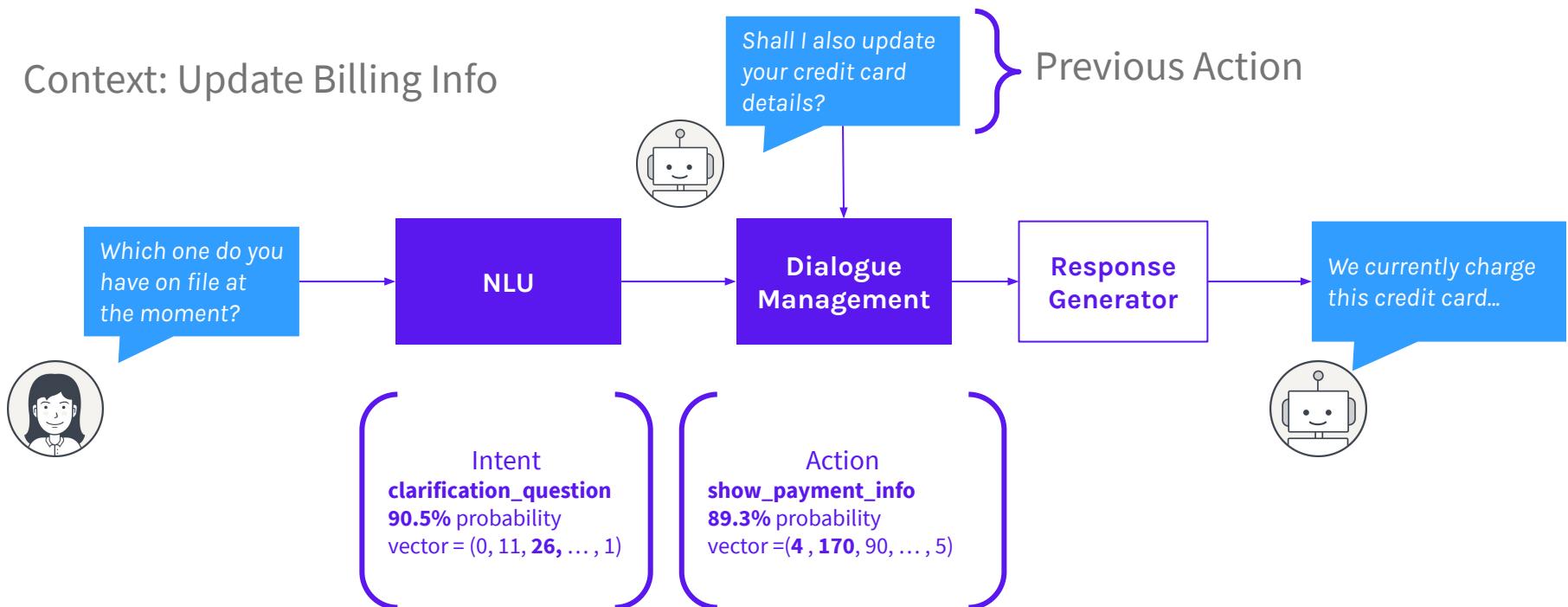
## The importance of context

Context: Mailing to an address

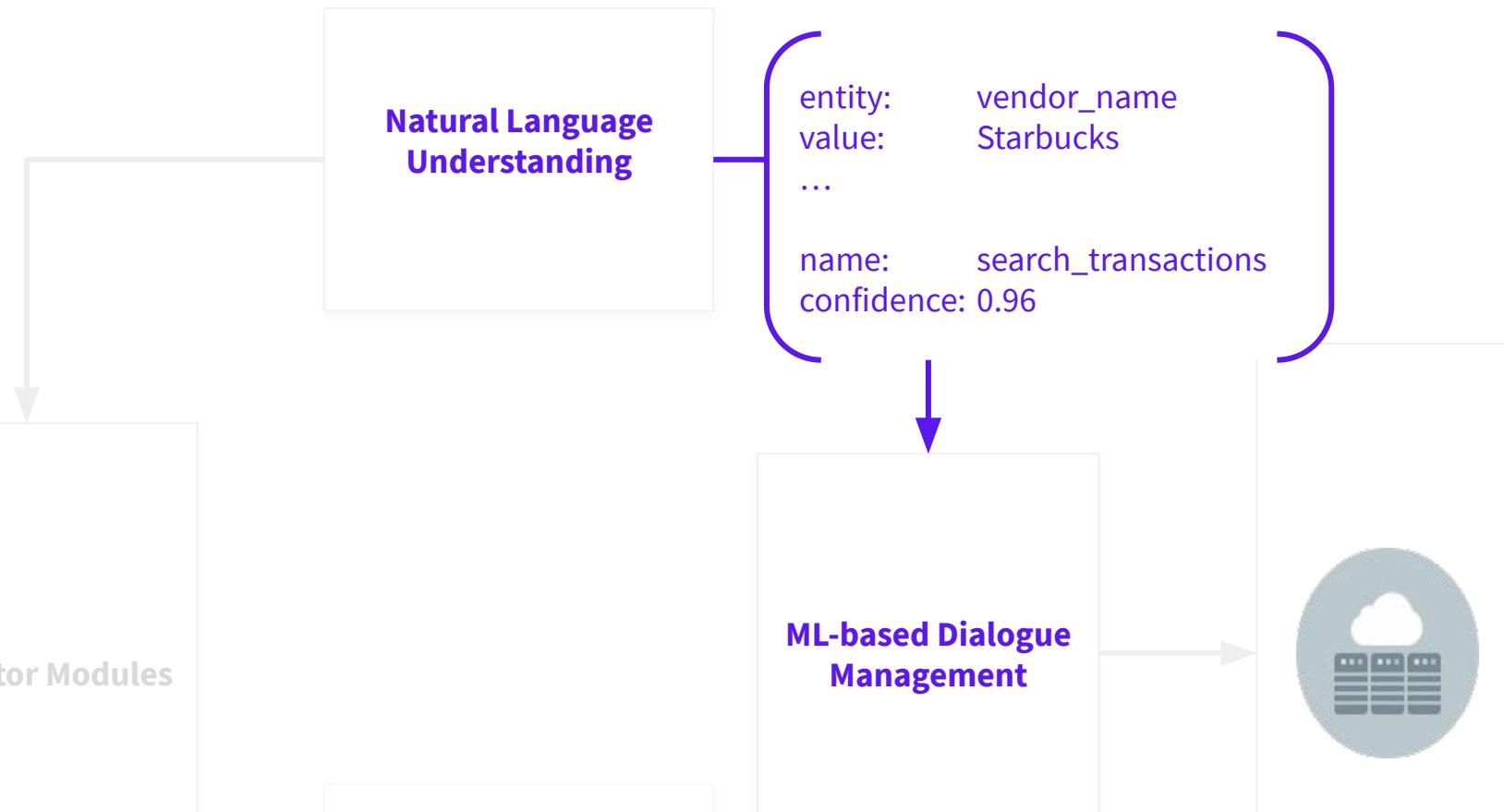


## The importance of context

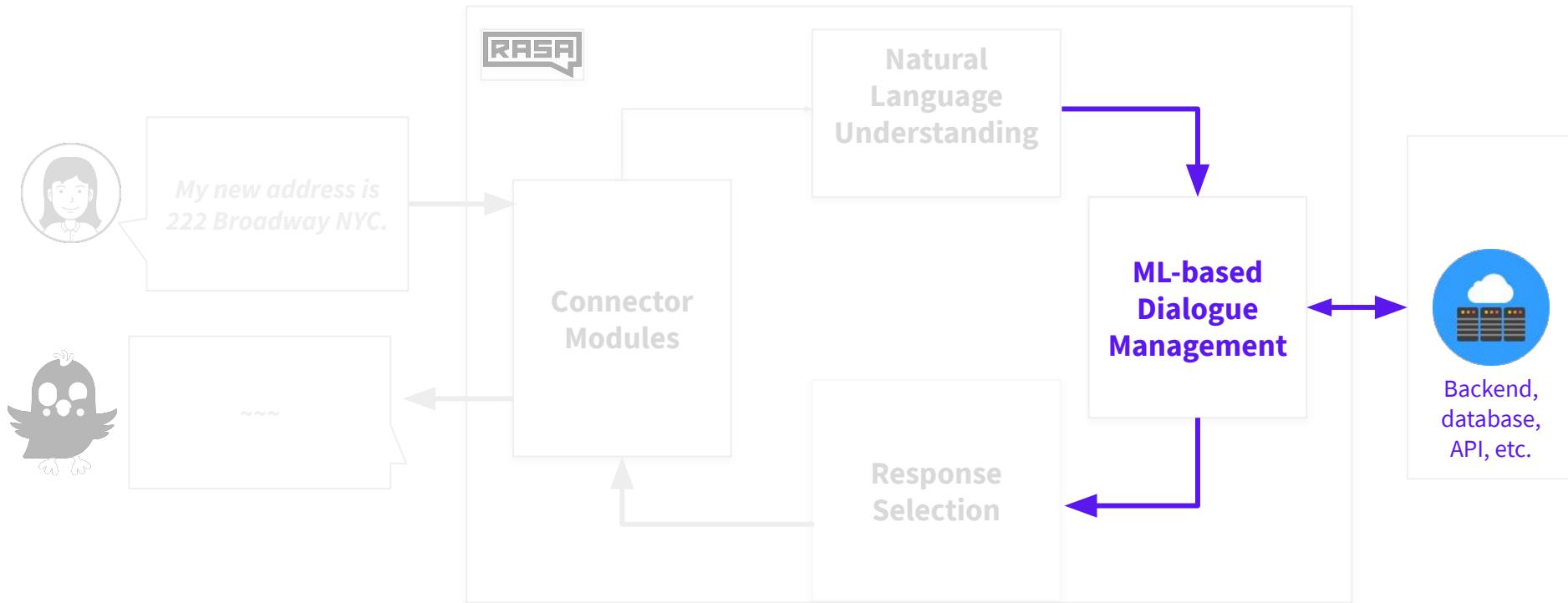
Context: Update Billing Info



## NLU: Output to Dialogue Management



# Dialogue Management

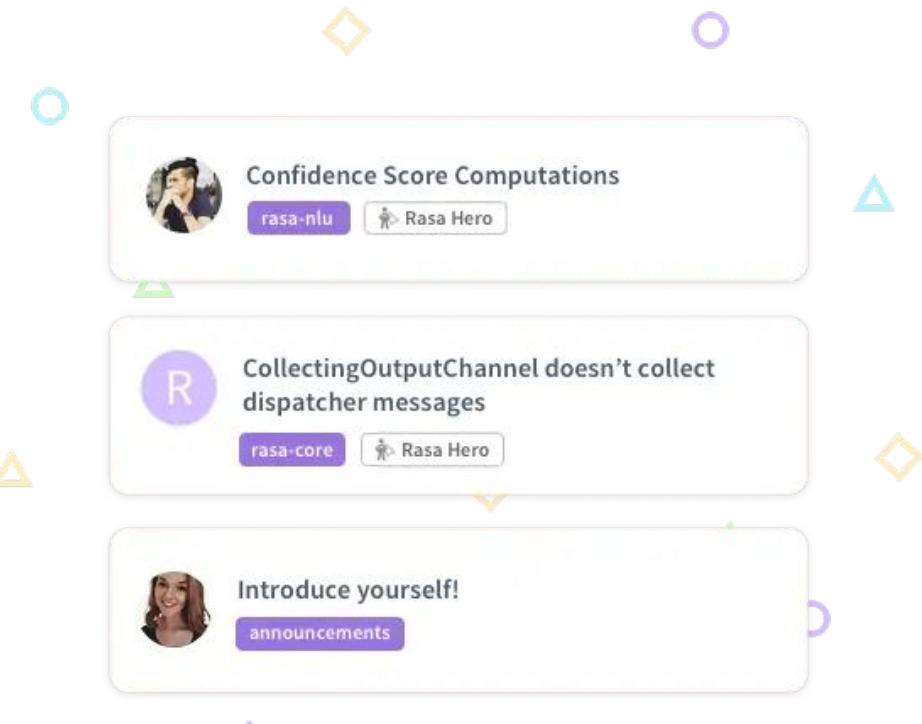


## Which featurizer is best depends on your dataset, so try different ones!

We don't believe in "one size fits all" machine learning

- We aim to provide sensible defaults and suggestions
- BUT even more important that these models are easy to customize

Share your results and compare notes with 8000+ developers at [forum.rasa.com](https://forum.rasa.com)



# Transformer Embedding Dialogue policy (TED)

## Conversational AI requires NLU and Dialogue management



Your total is \$15.50 - shall I charge the card you used last time?

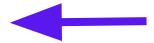
Yes.



intent: affirm, 99%



Done. You should have your items tomorrow.



action: confirm, 87%

## Happy paths are already solved

---

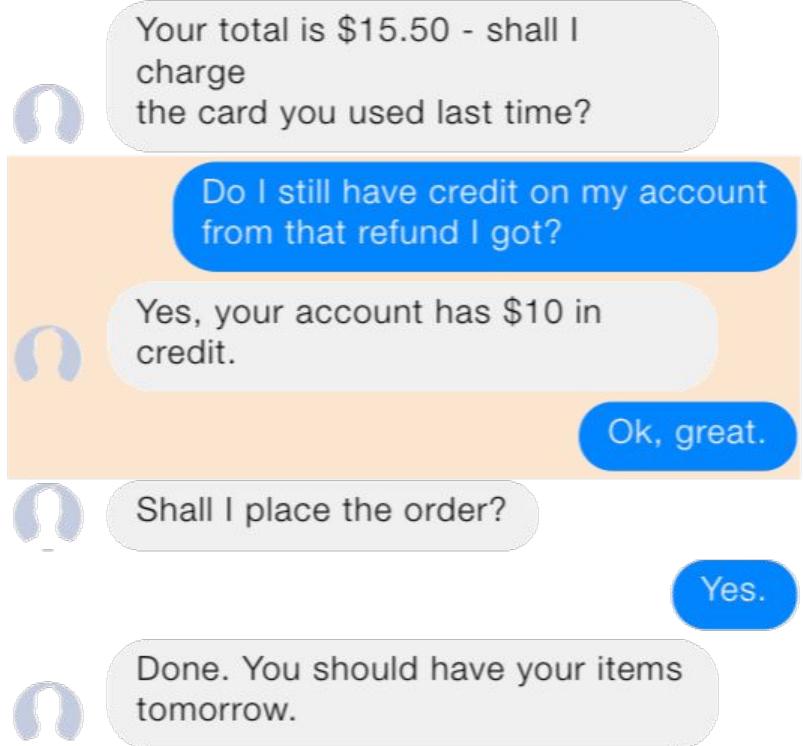
```
● ● ●

class CheckoutForm(FormAction):
 """Perform steps for checkout"""

 def name(self):
 return "checkout_form"

 @staticmethod
 def required_slots(tracker):
 return ["address", "card_number", "shipping_method"]
```

## You can't predict user behavior



## Users will interject and loop back to earlier topics

Your total is \$15.50 - shall I charge the card you used last time?

Do I still have credit on my account from that refund I got?

Yes, your account has \$10 in credit.

Ok, great.

Shall I place the order?

Yes.

Done. You should have your items tomorrow.

so did that use my credit?

## Real conversations don't follow the happy path

Your total is \$15.50 - shall I charge the card you used last time?

Do I still have credit on my account from that refund I got?

Yes, your account has \$10 in credit.

Wait, are you a bot?

Why yes I am! I may not be human, but I can help with lots of useful tasks 😊

Ok, great.

Shall I place the order?

Take the last thing I added out of my order. I changed my mind.

Okay, removing Reusable water bottle - Color blue

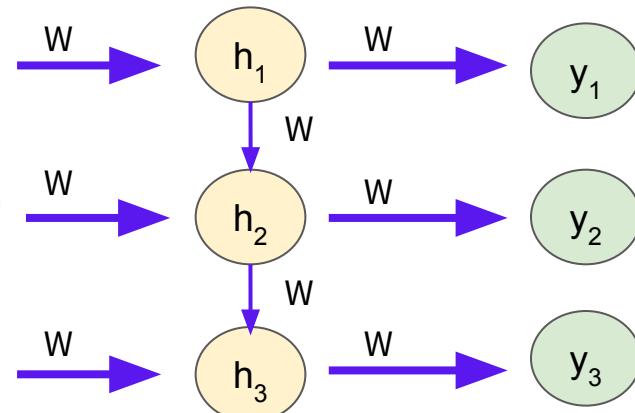
Cool, place the order now.

## People typically use a recurrent neural net (RNN) to model dialogue

Your total is \$15.50 - shall I charge the card you used last time?

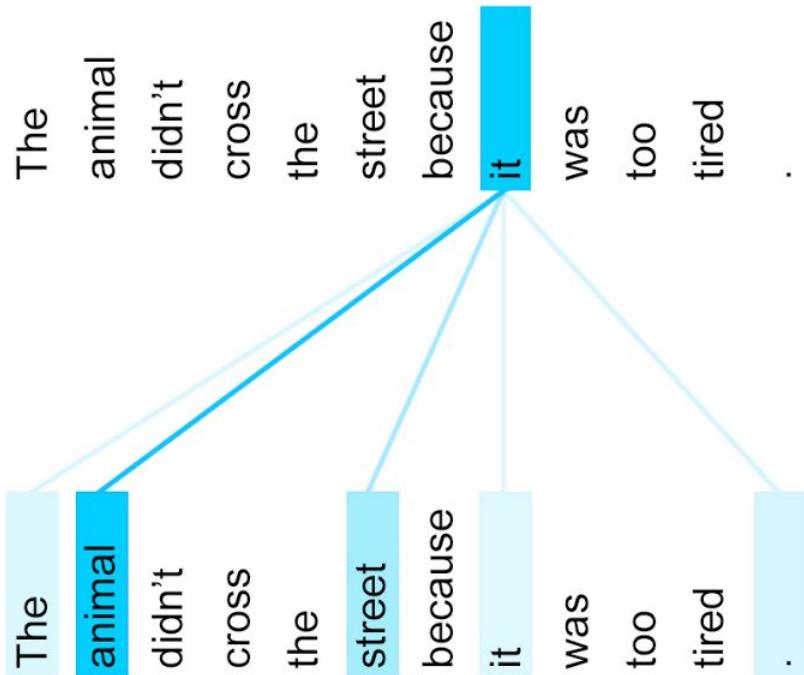
Yes.

Done. You should have your items tomorrow.

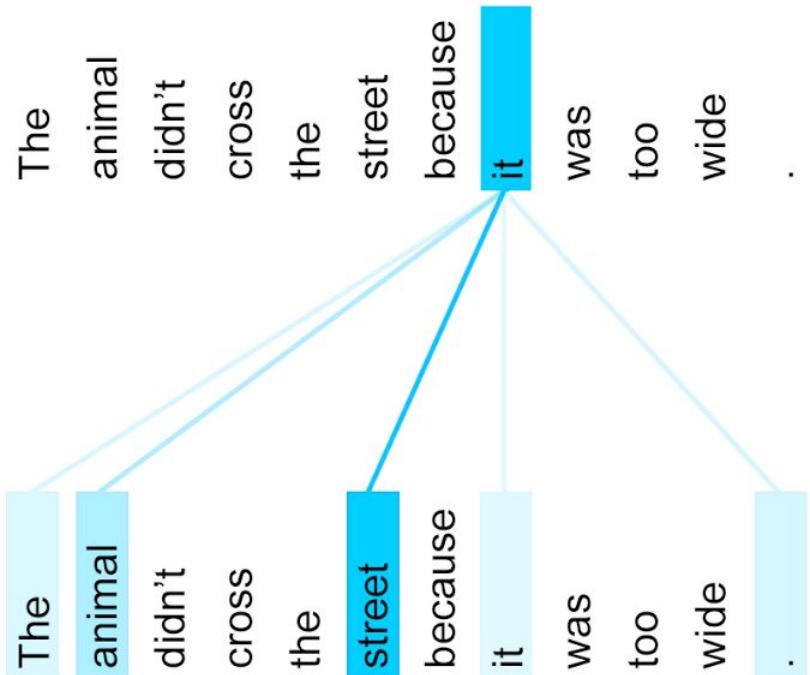


## But not all input should be treated equally

The animal didn't cross the street because it was too tired .



The animal didn't cross the street because it was too wide .



# Transformers (AKA self-attention) are now state of the art for many tasks

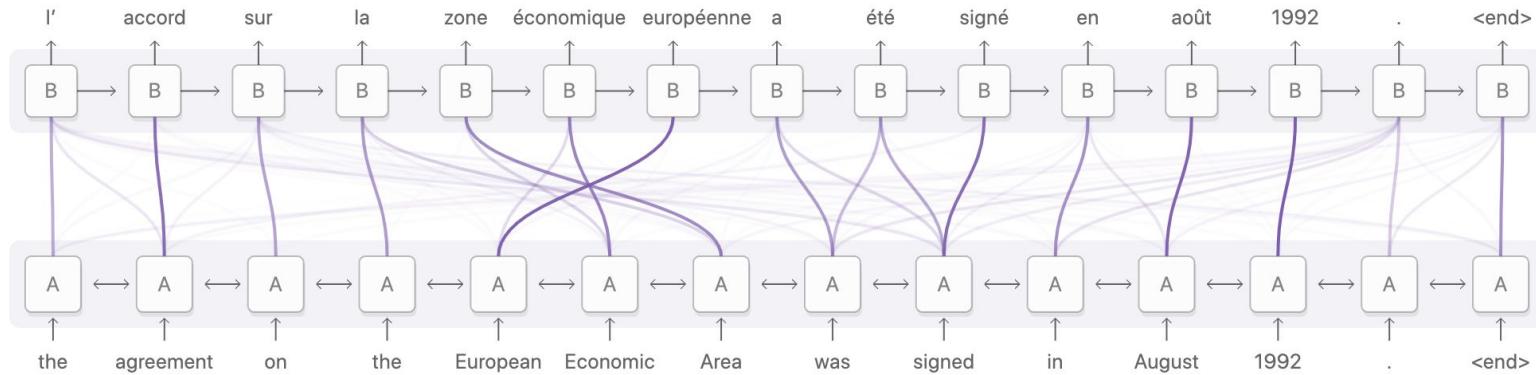
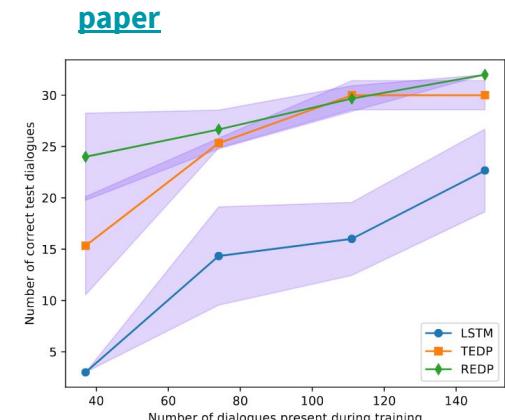
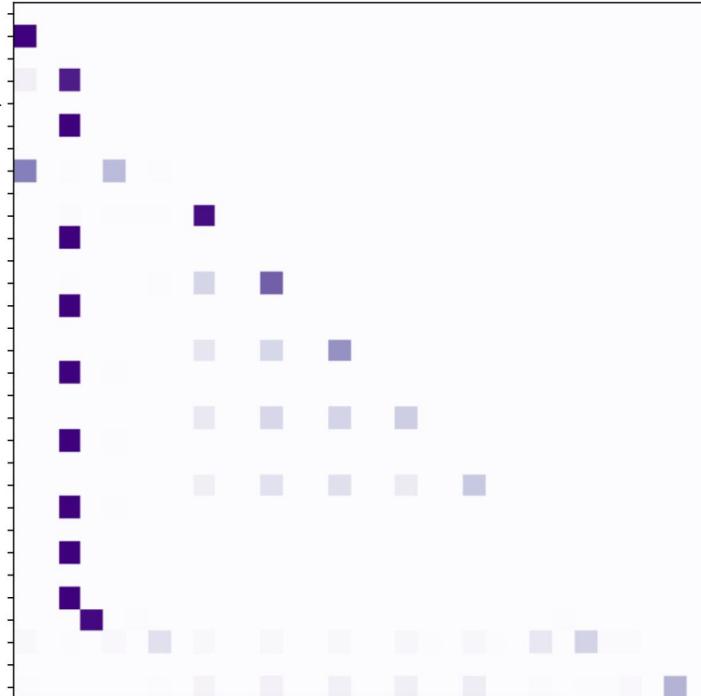


Diagram derived from Fig. 3 of Bahdanau, et al. 2014

<https://distill.pub/2016/augmented-rnns/>

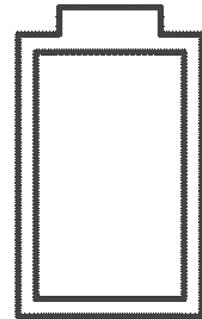
# We found out that the Transformer Embedding Dialogue policy can untangle sub-dialogues

```
* request_hotel
 - utter_ask_details
* inform{"enddate": "May 26th"}
 - utter_ask_startdate
* inform{"startdate": "next week"}
 - utter_ask_location
* inform{"location": "rome"}
 - utter_ask_price
* chitchat
 - utter_chitchat
 - utter_ask_price
* inform{"price": "expensive"}
 - utter_ask_people
* inform{"people": "4"}
 - utter_filled_slots
 - action_search_hotel
 - utter_suggest_hotel
* affirm
 - utter_happy
```



**Time for a short break (10 minutes)**

## HUMANS RECHARGING



## Conversation Driven Development (CDD)

### The challenge

When developing assistants, it's impossible to anticipate all of the things your users might say.

### The approach

A user-centric process: listening to your users and using those insights to improve your AI assistant.

Blog post: [Conversation-Driven Development](#)

# Conversation Driven Development



Rasa X

## Successful production-ready bots follow engineering best practices

The image shows a screenshot of the Rasa Tools and Products interface. It features two main status cards side-by-side.

**Left Status Card:**

- Icon:** A gear icon.
- Status:** Not connected
- Text:** Not connected to a Git repository. Status: Integrated Version Control has not been set up.
- Buttons:** Connect to a repository, Learn more in the docs

**Right Status Card:**

- Icon:** A gear icon.
- Status:** master
- Text:** master. Status: You have changes the Git server does not include. You also do not have the latest changes from the Git server.
- Buttons:** Add changes to Git server, Discard changes

## From zero to production-ready deployment in under 5 minutes

1. Create a VM
2. curl -s <http://get-rasa-x.rasa.com> | sudo bash

script takes ~4 mins and you have a full containerised deployment of rasa open source and rasa x  <https://lnkd.in/dsS-s9h>

What it does:

- installs k3s
- installs helm
- installs a helm chart with everything

## THE APPROACH

# 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

# 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**

## Testing

# 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/ras

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
```

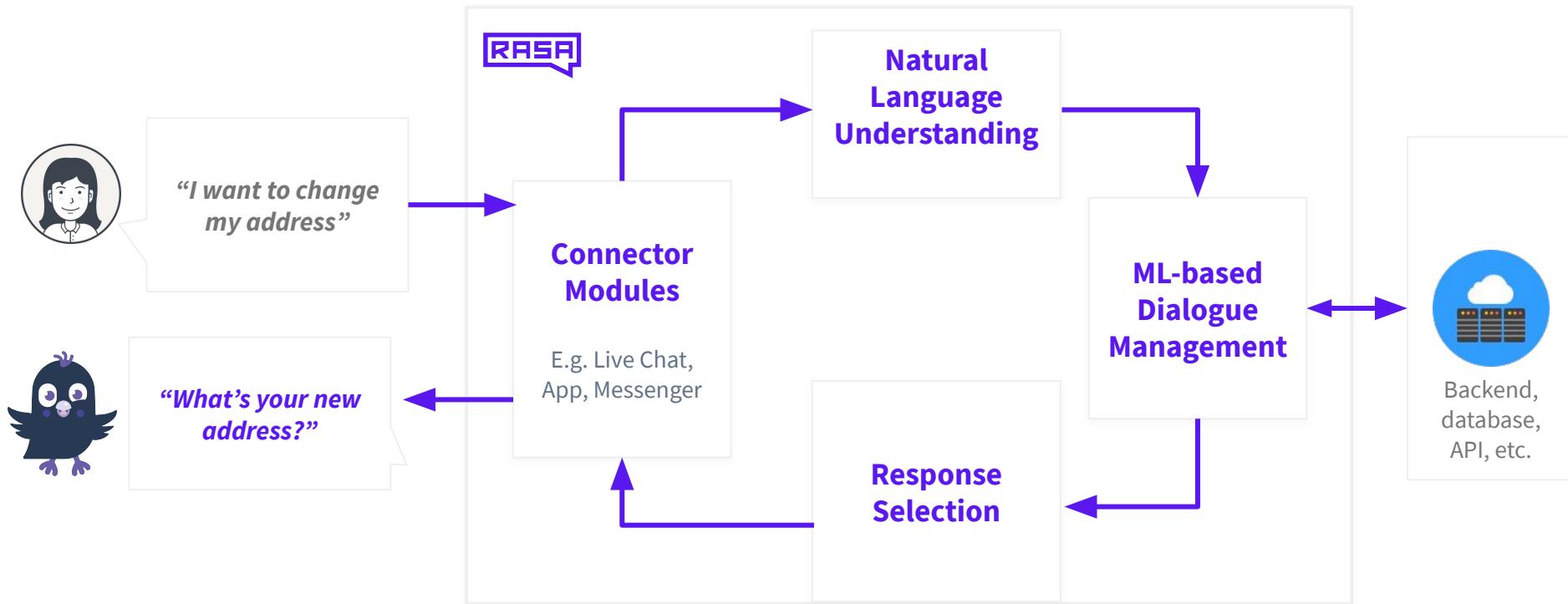
```
Your Rasa model is trained and saved at '/Users/ty/Documents/product_mgmt/docs/rasa/test'
((rasa_env) BERMB00017:temp ty$ rasa test
Processed Story Blocks: 100%|██████████
2020-03-24 14:24:31 INFO rasa.core.test - Evaluating 7 stories
Progress: 100%|██████████
2020-03-24 14:24:32 INFO rasa.core.test - 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
```

# **Part 3: Adding custom actions and implementing forms**

## Roadmap for the session

1. Rasa events and slots: talk and coding
2. Custom actions: talk and coding
3. Forms: talk and coding
4. Recap

# Rasa Open Source



## Project setup: Files

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

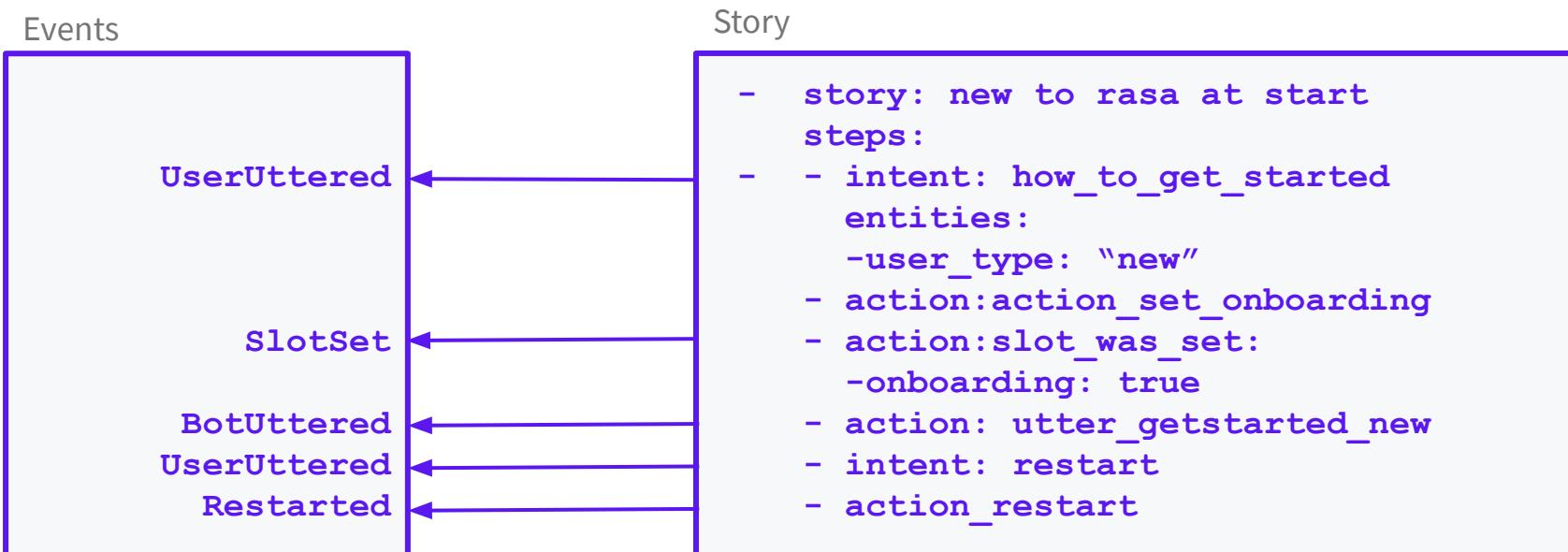
## Project setup: Files

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

# Rasa events and slots

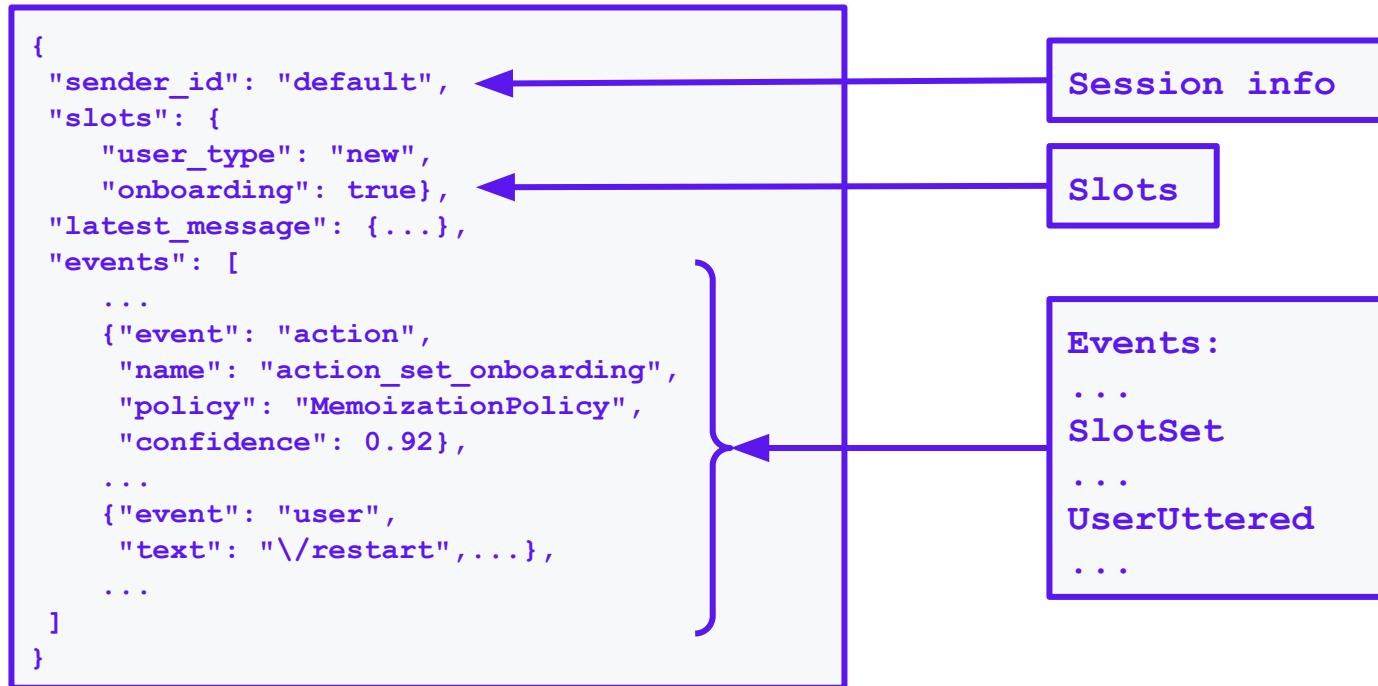
## Events

- **Internally**, conversations are represented as a sequence of **events**.
- Some events are automatically tracked



## Tracker

- Trackers maintain the **state of a dialogue** between the assistant and the user.
- It keeps track of events, slots, session info etc.



## Slots

Your bot's memory

Hi, my name is **Tom**

Hello! How can I help?

I would like to check my account balance

**Sure. Can you tell me your surname?**

Hi

Hello! How can I help?

I would like to check my account balance

**What is your name?**

## Slots

### Your bot's memory

- Can store:
  - user-provided info
  - info from the outside world
- Can be set by:
  - NLU (from extracted entities, or buttons)
  - Custom Actions
- Can be configured to affect or not affect the dialogue progression

## Slots

Slots can be configured to affect or not to affect the dialogue progression

Setting the property `influence_conversation` allows you to configure whether or not the slot should influence how the conversation goes.

```
slots:
ages:
type: text
influence_conversation: true
```

Slot will influence the next action prediction.

```
slots:
ages:
type: text
influence_conversation: false
```

Slot will not influence the next action prediction.

**The way a slot will influence the conversation will depend on a slot type.**

## Slots types

Text and List slots influence conversation path based on **whether it is set or not.**

**Text slots** can be used to store **text values**

```
slots:
location:
type: text
```

“I am based in London”

**List slots** can be used to store **a list of values**

```
slots:
appointment:
type: list
```

“Are there available appointments on  
Monday or Wednesday this week?”

## Slots types

Categorical, Boolean and Float slots influence conversation path based on **the value of the slot.**

**Categorical slots** can be used to store slots which can take **one of N values**

```
slots:
 price_level:
 type: categorical
 values:
 - low
 - medium
 - high
```

“I am looking for a restaurant in a low price range.”

**Boolean slots** can be used to store **true or false values**

```
slots:
 is_authenticated:
 type: boolean
```

**Float slots** can be used to store **numerical values**

```
slots:
 radius:
 type: float
 min_value: 0
 max_value: 100
```

“Find me a restaurant within 2 mile radius.”

## Slots types

Slots of the type **any** don't have any influence on the dialogue.

**Slots of type any** can be used to store any arbitrary values of any type (list, dict, etc).

```
slots:
shopping_items:
type: any
```

“Add milk and bread to my shopping list.”

## Slots

There are a few different ways how slots can be set

- Slots set from NLU

```
- intent:search_transactions
examples: |
 - how much did I spend at [Target] (vendor_name) this week?
 - what is my typical spending at [Amazon] (vendor_name) ?
```

```
entities:
- vendor_name

slots:
 vendor_name:
 type: text
```

## Slots

There are a few different ways how slots can be set

- Slots set from NLU

```
- intent:search_transactions
examples: |
 - how much did I spend at [Target] (vendor_name) this week?
 - what is my typical spending at [Amazon] (vendor_name)?
```

```
entities:
- vendor_name

slots:
 vendor_name:
 type: text
```

## Slots

There are a few different ways how slots can be set

- Slots set from NLU

```
- intent:search_transactions
examples: |
- how much did I spend at [Target] (vendor_name) this week?
- what is my typical spending at [Amazon] (vendor_name)?
```

```
entities:
- vendor_name

slots:
vendor_name:
type: text
```

## Slots

There are a few different ways slots can be set

- Slots set by clicking buttons

```
responses:
utter_ask_transfer_form_confirm:
 - text: "Would you like to transfer $100 to Tom?"
 buttons:
 - title: Yes
 payload: /affirm
 - title: No, cancel the transaction
 payload: /deny
```

Would you like to transfer \$100 to Tom?

Yes

No, cancel the transaction

## Slots

There are a few different ways how slots can be set

- Slots set by custom actions

```
-story: account_balance_story
steps:
- intent: check_account_balance
- action: action_account_balance
- slot_was_set:
 - account_balance: 1000
```

## Slots

There are a few different ways how slots can be set

- Slots set by custom actions

```
-story: account_balance_story
steps:
- intent: check_account_balance
- action: action_account_balance
- slot_was_set:
 - account_balance: 1000
```

## Let's define the slots for our Financial Bot assistant!

1. For the practical part of our workshop we will be using the financial-demo repository:

```
git clone https://github.com/RasaHQ/financial-demo.git
```

2. Pull the latest updates pushed to the repo:

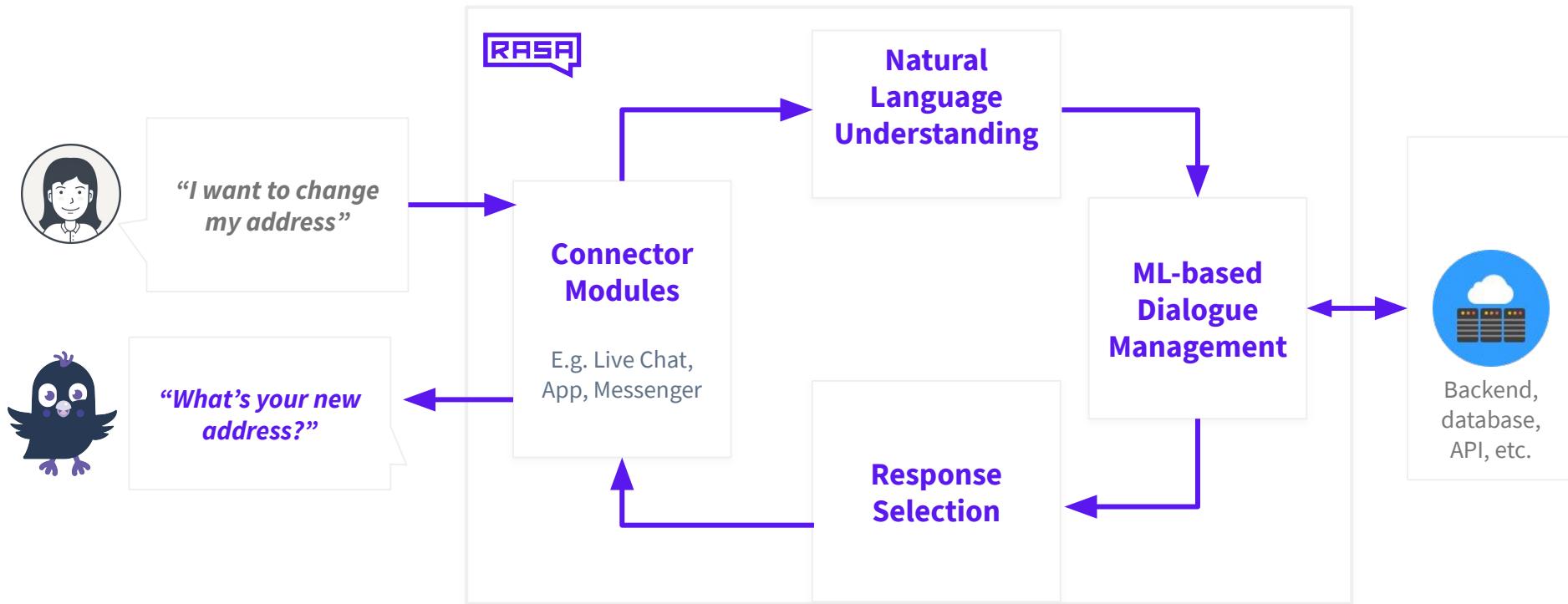
```
git pull
```

3. Checkout the branch named “rasa-2-0”:

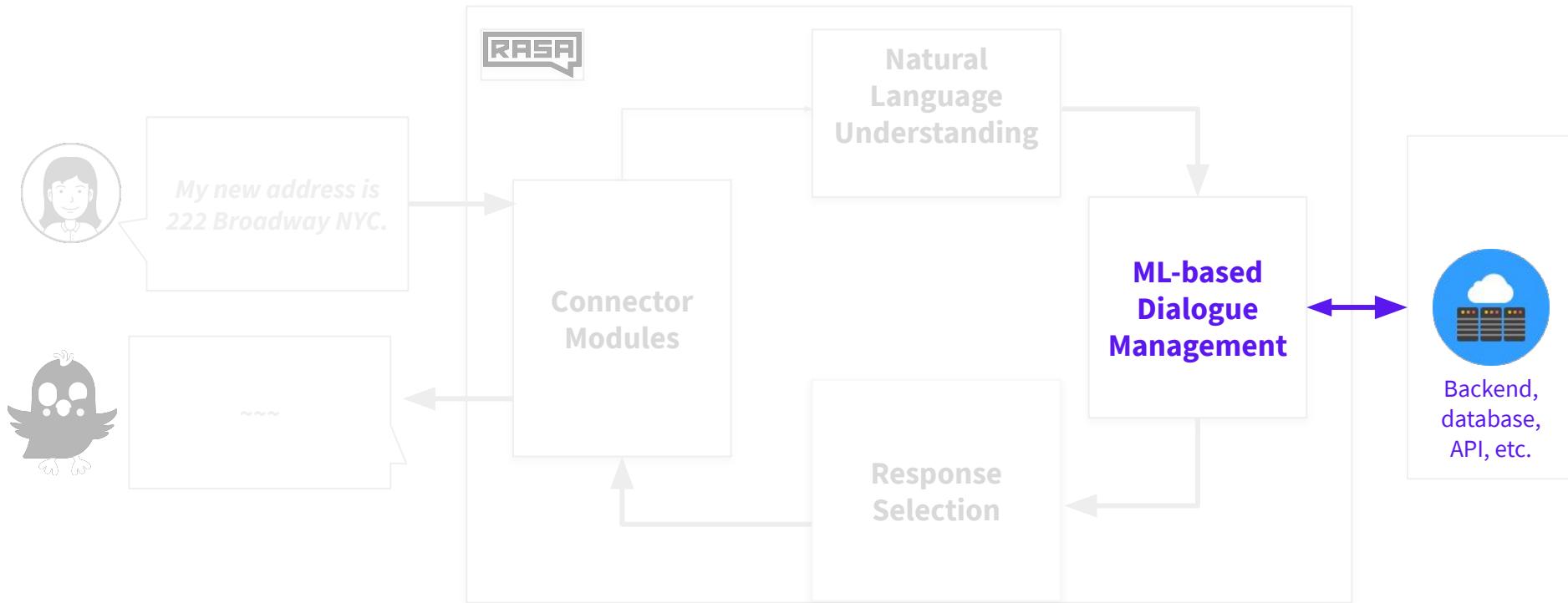
```
git checkout -b rasa-2-0
```

# Custom actions

## Custom Actions: Connecting to the outside world



## Custom Actions: Connecting to the outside world



## Actions

Functions your bot runs in response to user input.

**Four different action types:**

- **Utterance actions:** `utter\_`
  - send a specific message to the user
  - specified in `responses` section of the domain
- **Retrieval actions:** `respond\_`
  - send a message selected by a retrieval model
- **Custom actions:** `action\_`
  - run arbitrary code and send any number of messages (or none).
  - return events
- **Default actions:**
  - built-in implementations available but can be overridden
  - E.g. action\_listen, action\_restart, action\_default\_fallback

## Custom Actions: The Action Server

Custom action code is run by a **webserver** called the **action server**

- **How custom actions get run:**
  - When a custom action is predicted, Rasa will call the **endpoint** to your action server
  - The action server:
    - runs the code for your custom action
    - [optionally] returns information to modify the dialogue state
- **How to create an action server:**
  - You can create an action server in any language you want!
  - You can use the **Rasa-SDK** for easy development in Python

## Custom Actions: Examples

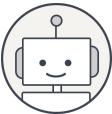
Custom actions can do whatever you want them to! e.g.

- **Send multiple messages**
- **Query a database**
- **Make an API call to another service**
- **Return events e.g.**
  - Set a slot
    - e.g. based on a database query)
  - Force a follow up action
    - force a specific action to be executed next
  - Revert a user utterance
    - I.e. remove a user utterance from the tracker

## Custom Action Example: Query a Database

using the Rasa SDK

"Should I also update your mailing address?"



"What do you have on file?"



```
class ActionCheckAddress(Action):
 def name(self) -> Text:
 return "action_check_address"

 def run(self,
 dispatcher: CollectingDispatcher,
 tracker: Tracker,
 domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
 idnum = tracker.get_slot('person id')
 q = "SELECT Address FROM Customers \
 \WHERE CustomerID={};'{}''.format(idnum)
 result = db.query(q)

 return [SlotSet(
 "address",
 result if result is not None else "NotOnFile")]
```

## Let's implement a custom action for our financial assistant

We will implement a custom action which will allow users to pay their credit card:

Hello

Hello! How can I help?

I would like to pay a credit card

Towards which bank would you like to make a payment?

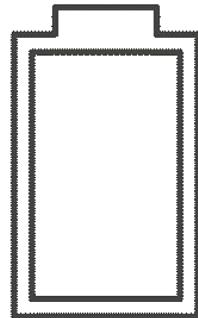
...

Would you like to make a \$100 payment towards the iron bank?

Yes

**Time for a short break (15 minutes)**

# HUMANS RECHARGING



# Forms

## Forms

Rasa Forms is the best way to enable your assistant to collect information in order to do something.

### ## Happy path 1:

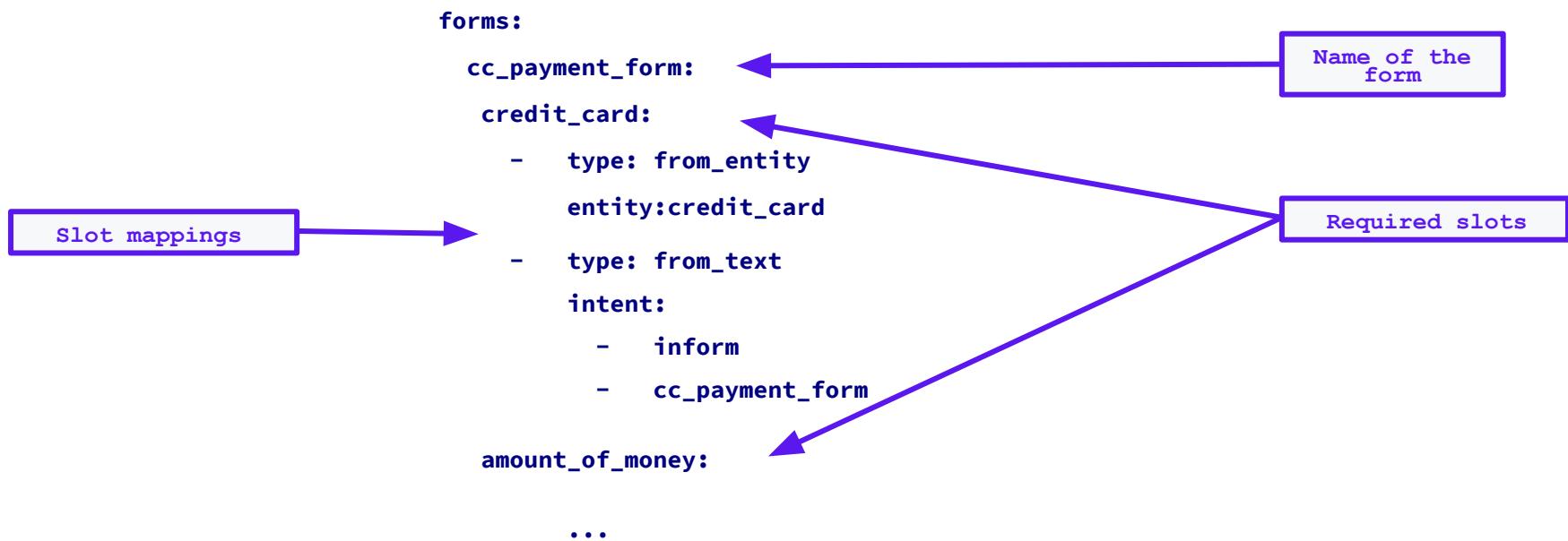
U: I would like to make a money transfer  
B: Towards which credit card would you like to make a payment?  
U: Towards my justice bank credit card  
B: How much do you want to pay?  
U: \$100  
B: The transaction has been scheduled.

### ## Happy path 2:

U: I would like to make a \$100 money transfer  
B: Towards which credit card would you like to make a payment?  
U: Towards my justice bank credit card  
B: The transaction has been scheduled.

## Forms

Forms are defined in your assistant's domain:



...

## Forms

To activate a form you have to add a story or a rule which describes when the form should be activated.

- **rule:**
- steps:**
  - **intent: pay\_cc**
  - **action: cc\_payment\_form**
  - **active\_loop: cc\_payment\_form**

## Forms: Submitting the form

Once your form fills in all the requested slots, there should be an actions that your assistant runs. Those actions can be utterances or custom actions.

```
- rule:submit_form
 condition: cc_payment_form
 steps:
 - action: cc_payment_form
 - active_loop: null
 - slot_was_set:
 - requested_slot: null
 - action: action_pay_cc
```

## Forms: Custom slot mappings

The slot mappings define how to extract slot values from user inputs.

With slot mappings you can define how certain slots can be extracted from:

- Entities
- Intents
- Support free text input
- Support yes/no inputs

```
slot_name:
 - type: from_entity
 entity: entity_name
 role: role_name
 group: group_name
 intent: intent_name
 not_intent: excluded_intent
```

## Forms: Validating user input

After extracting a slot value from user input you can validate the slot against a set of values.

I would like to make a \$100 payment.

slot: amount\_of\_money



validate

“Minimum balance”: 85

“Current balance”: 550

## Forms: Validating user input

Slot validation can be implemented as a custom action by extending the FormValidationAction class.

```
class ValidatePayCCForm(CustomFormValidationAction):
 """Validates Slots of the cc_payment_form"""

 def name(self) -> Text:
 """Unique identifier of the action"""
 return "validate_cc_payment_form"
```

## Forms

### Handling unhappy paths

Users are not always cooperative - they can change their mind or interrupt the form.

- Create stories or rules to handle the interruptions:

```
- rule: chitchat
 condition:
 active_loop: cc_payment_form
 steps:
 - intent: chitchat
 - action: utter_chitchat
 - action: cc_payment_form
 - active_loop: null
```

## Forms

### Handling unhappy paths

Users are not always cooperative - they can change their mind or interrupt the form.

- Use action `action_deactivate_loop` to handle the situation where users decide not to proceed with the form:
  - **story: uncooperative behaviour**  
**steps:**
    - `intent: pay_cc`
    - `action: cc_payment_form`
    - `active_loop: cc_payment_form`
    - `intent: stop`
    - `action: utter_ask_continue`
    - `intent: deny`
    - `action: action_deactivate_loop`
    - `active_loop: null`

Let's Code

## Let's implement a form for paying out a credit card

We will implement a form which will enable our assistant to collect necessary information to pay a credit card:

Hello

Hello! How can I help?

I would like to pay my credit card

Towards which account would you like to make a payment?

Iron bank

How much would you like to transfer?

\$1000

When would you like the transaction to be made?

Today

Do you want to schedule a \$1000 transfer to iron bank for today?

Yes

# Recap

## Slots

### Your bot's memory

- Can store:
  - user-provided info
  - info from the outside world
- Can be set by:
  - NLU (from extracted entities, or buttons)
  - Custom Actions
- Can be configured to affect or not affect the dialogue progression

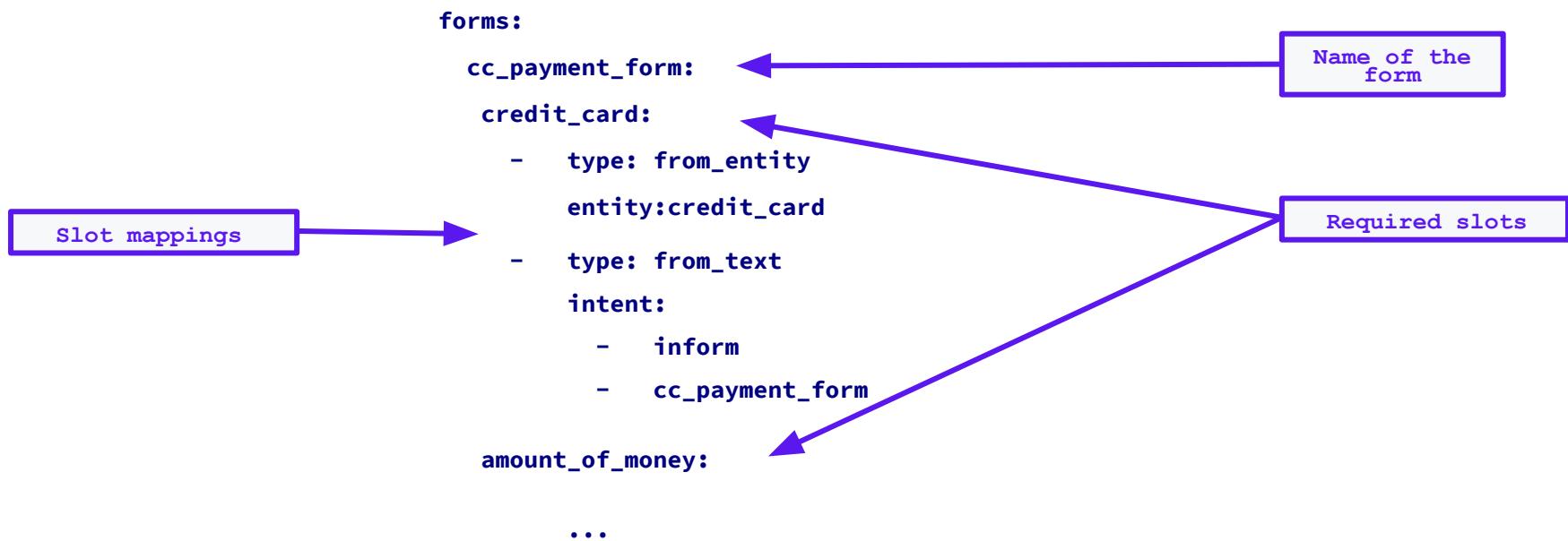
## Custom Actions: Examples

Custom actions can do whatever you want them to! e.g.

- **Send multiple messages**
- **Query a database**
- **Make an API call to another service**
- **Return events e.g.**
  - Set a slot
    - e.g. based on a database query)
  - Force a follow up action
    - force a specific action to be executed next
  - Revert a user utterance
    - I.e. remove a user utterance from the tracker

## Forms

Rasa Forms is the best way to enable your assistant to collect information in order to do something.



# Part 4: Messaging Channels, Conversation Driven Development, and Rasa X

# Agenda

Theme for this section: Connecting with your users

- I. Conversation Driven Development
  - A. Combining user input and software engineering best practices at every stage of the development cycle
  - B. Conversations = training data
- II. Rasa X
  - A. A tool for Conversation Driven Development
  - B. Launching Rasa X locally
  - C. Reviewing conversations
- III. Messaging Channels
  - A. Live coding: connecting Telegram
- IV. Review

# Conversation Driven Development

## CDD captures the lessons we've learned as a community

If you've built conversational AI before, you know that:

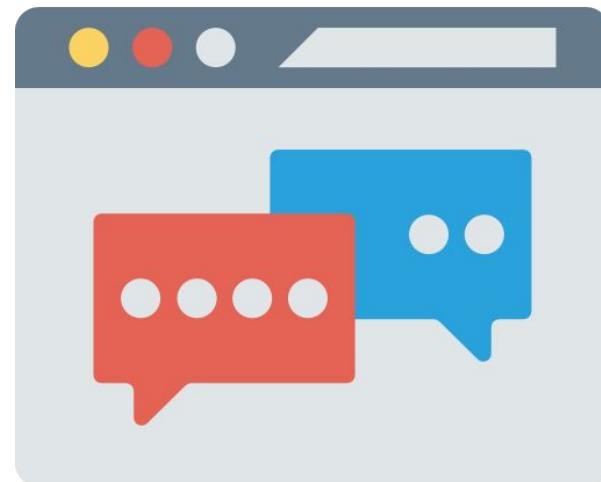
- It is very hard.
- Building a prototype is not the hard part.
- The hard parts all show up when you want to go from a prototype to something you'd want to ship.

Conversation-Driven Development should:

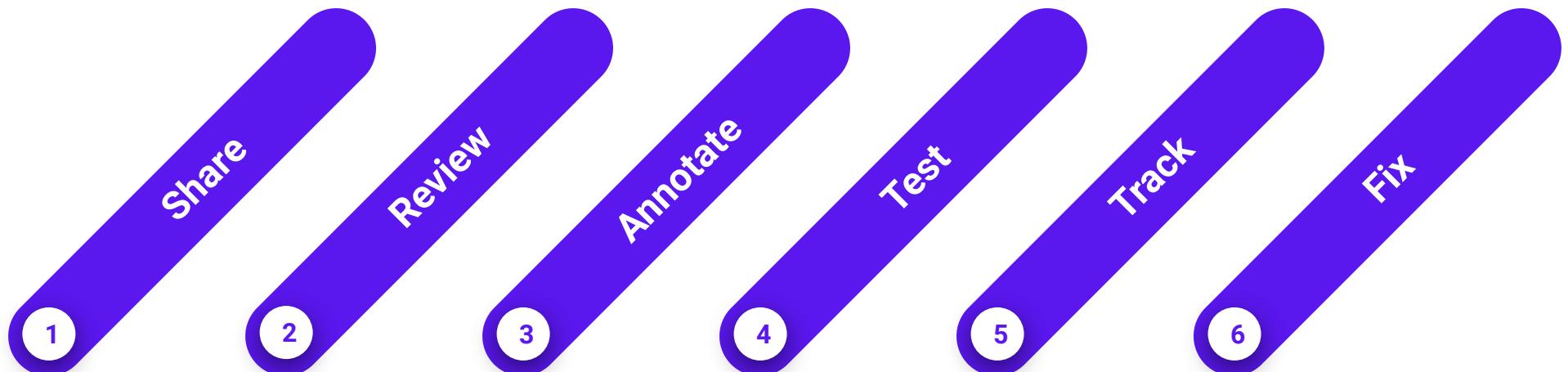
- Help all of us build better conversational AI.
- **Save newcomers from having to learn this the hard way.**

## What is the opposite of conversation driven development?

- Developing your assistant for months in isolation before testing it with users
- Only looking at top-level metrics, not conversations themselves
- Autogenerating training data that doesn't reflect ways users really interact
- Pushing changes to production without running tests



# Conversation-Driven Development



## Share



Users will always surprise you.

So get some test users to try your prototype **as early as possible**.



Projects that ship without a bunch of test users are bound to fall down in production.



# Review



At every stage of a project, it is worth reading what users are saying.

Avoid getting caught up in metrics right away. Conversations are valuable data.

The screenshot displays a list of messages from a user on Facebook. The messages are timestamped and categorized by the AI's action. A 'help me' button is visible at the top right, and a 'faq' count of 3 is shown. The messages are:

- 4:59 am, 12 May 2020: facebook - action\_listen (red dot)
- 10:39 am, 11 May 2020: facebook - respond\_faq (green dot)  
I can help you calculate and buy carbon offsets for your flights.
- 8:50 am, 9 May 2020: facebook - action\_listen (red dot)
- 5:10 am, 9 May 2020: facebook - respond\_faq (green dot)  
I can help you calculate and buy carbon offsets for your flights.
- 9:14 am, 8 May 2020: facebook - action\_listen (red dot)
- 2:46 am, 8 May 2020: facebook - respond\_faq (green dot)  
I can help you calculate and buy carbon offsets for your flights.
- 12:44 pm, 7 May 2020: facebook - action\_listen
- 9:51 pm, 3 May 2020: facebook - respond\_faq  
I'm great! Thanks for asking.
- 1:46 pm, 3 May 2020: facebook - action\_listen
- 12:26 pm, 30 Apr 2020: facebook - respond\_faq  
that's not something I can help with
- 7:37 pm, 24 Apr 2020: facebook - action\_listen
- 5:22 pm, 24 Apr 2020: Tester - inform("city":"new york")

Buttons for 'what you do for me' (3 faq), 'how are you', and 'flight details pless' are also visible.

## Annotate



Using a script to generate synthetic training data



Turning real messages into training examples

The interface displays a message history between a user (Sara) and a bot (Bender). The user asks for a menu, and the bot responds with a greeting and a message about ordering pizza. A purple arrow points from the 'greet' action in the history to a configuration panel at the bottom right.

## New story

- \* greet
  - utter\_greet\_user
  - utter\_ask\_pizza\_choice

How can I get the menu for your place?

greet

Hey there! My name is Sara!

To order a pizza, just tell us which kind you'd like!

action\_listen

Next Action: utter\_menu

✓

## Real Conversations > Synthetic Data

During development, training data is created by:

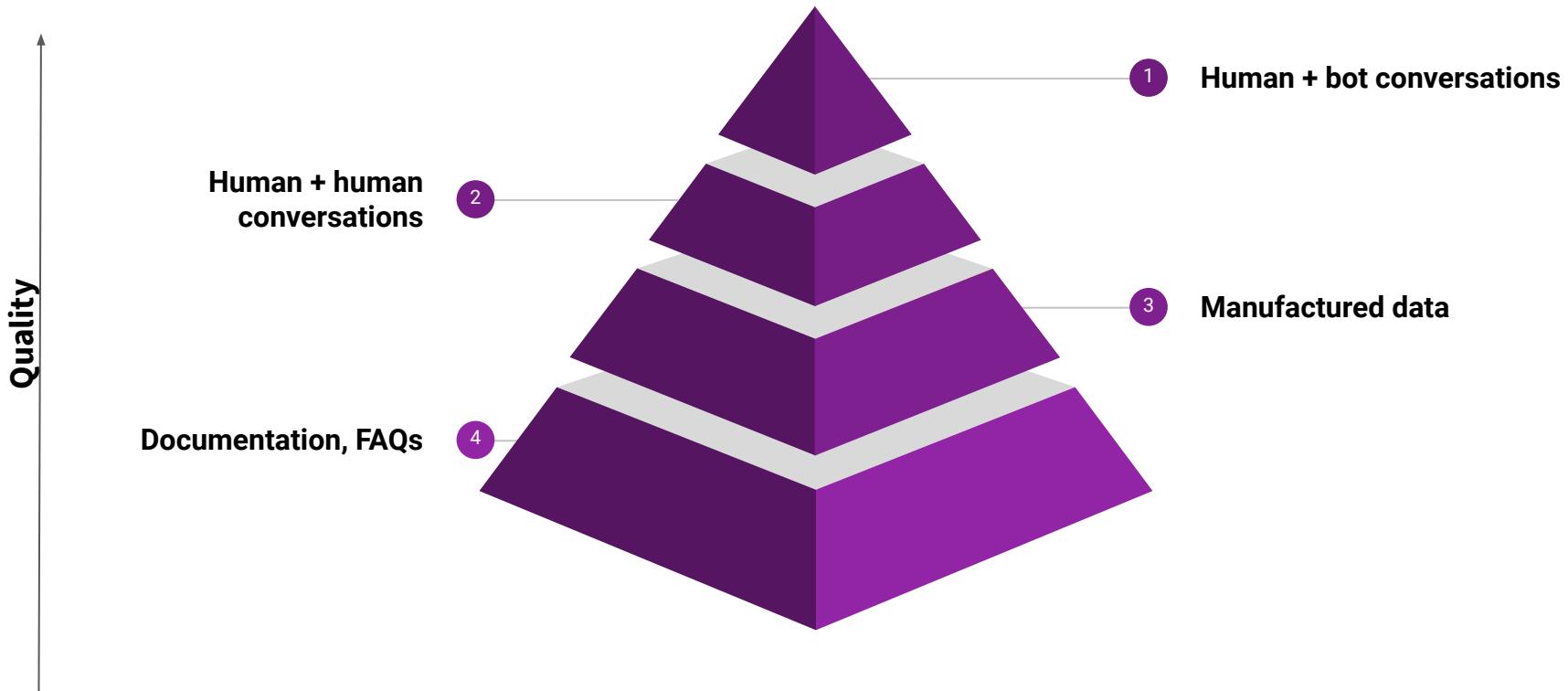
- Writing down hypothetical conversations and training examples
- Studying existing chat transcripts or knowledge bases
- Consulting subject matter experts

That's a great start, but all of those scenarios are based on **human:human** conversations.

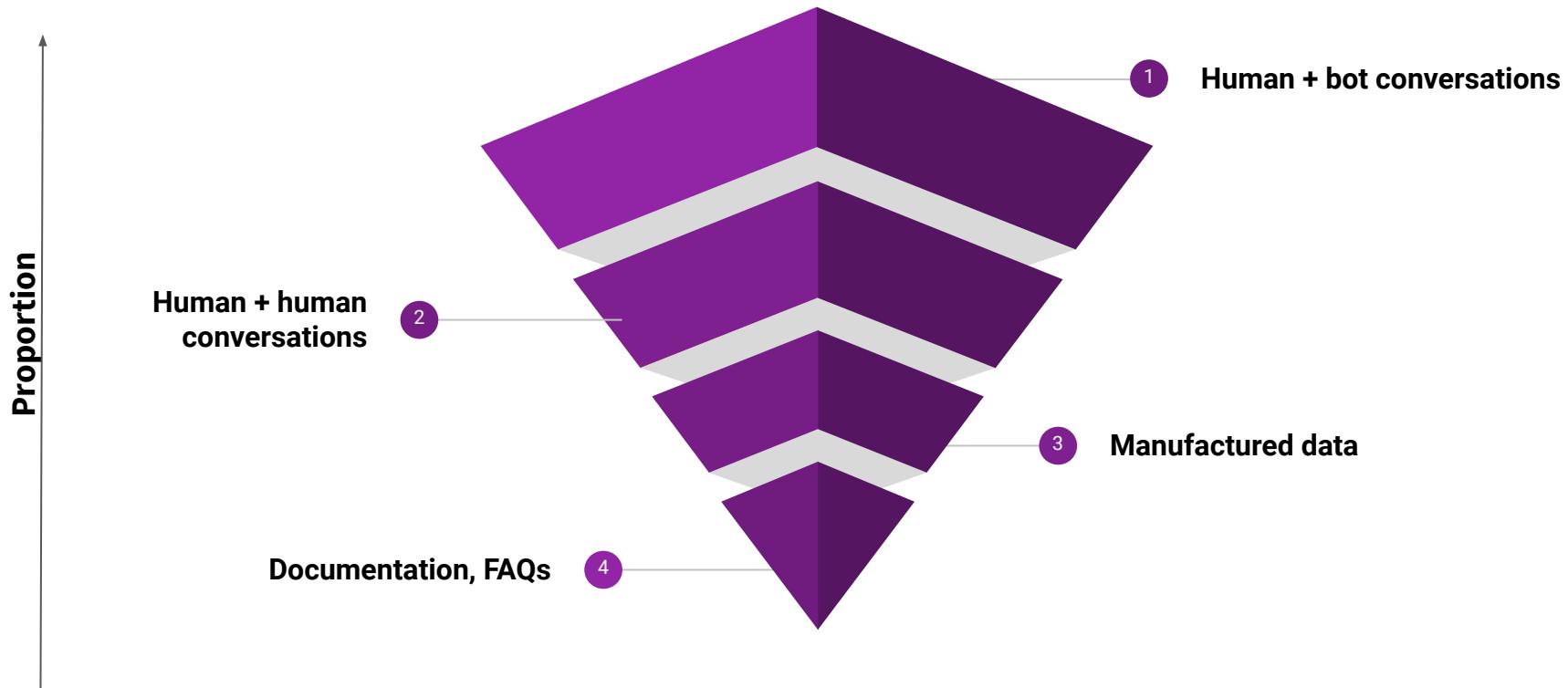
**Human:bot** conversations have different patterns - users talk to a bot differently than they do a human

*The best training data is generated from the assistant's  
actual conversations*

## Quality of data, by source



**By the time you go to production, most of your training data should come from human:bot interactions**



## Test



Professional teams don't ship applications without tests.

Use whole conversations as end-to-end tests

Run them on a continuous integration (CI) server.

All checks have passed  
4 successful checks

✓ build Successfully in 59s — build

✓ test Successfully in 59s — build

✓ publish Successfully in 59s — build

✓ This branch has no conflicts with the base branch  
Merging can be performed automatically.

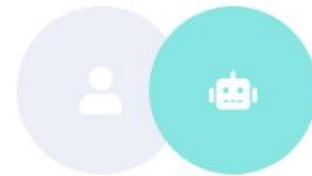
Merge pull request ▾ You can also open this in GitHub Desktop

## Track



Use proxy measures to track which conversations are successful and which ones failed.

'Negative' signals are useful too, e.g. users **not** getting back in touch with support.



**Conversation between  
Demobot and  
1599192453538535**

link-1-clicked

form\_failed\_...



## Fix



Study conversations that went smoothly and ones that failed.

Successful conversations can become new tests 🎉

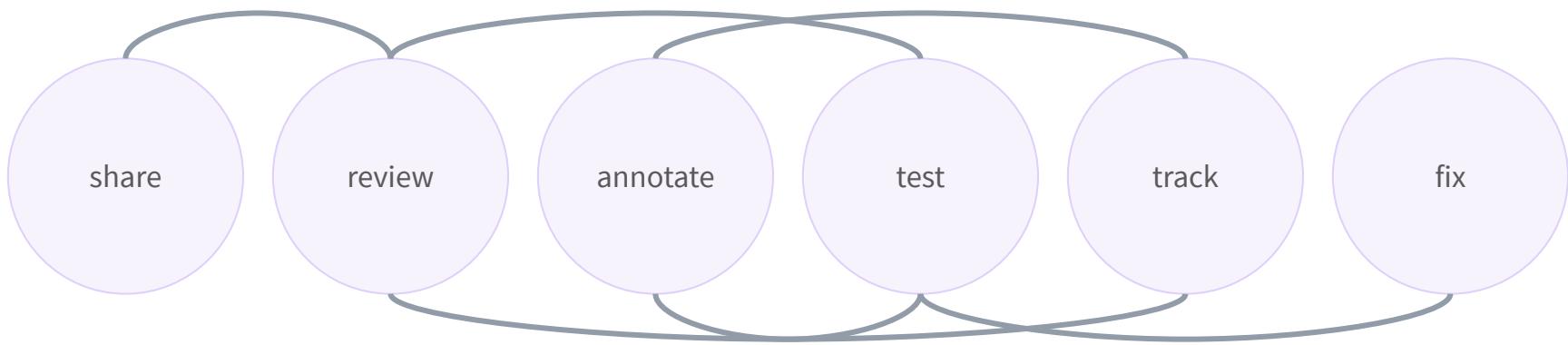
Fix issues by annotating more data and/or fixing your code 🔧



# CDD in practice

PROCESS

**It's not a linear process: you'll find yourself jumping between these actions**



Some actions require software skills, others a deep understanding of the user



## CDD for teams

Track

Review

Share

Test

Annotate

Fix

### Product

- Product Managers
- Innovation and Business Leaders

### Design

- UX Designers
- Conversation Designers
- Content Creators

### Development

- Engineers
- Data Scientists
- Annotators

**Rasa X is a tool to help  
teams do CDD**

## Rasa X turns conversations into training data



Review  Annotate  Improve  


# Rasa X layers on top of Rasa Open Source

Rasa Open Source: Framework for building AI assistants

```
language: en
pipeline:
- name: HFTransformersNLP
 model_weights: "gpt2"
 model_name: "gpt2"
- name: LanguageModelTokenizer
- name: LanguageModelFeaturizer
- name: DIETClassifier
 intent_classification: True
 entity_recognition: True
epochs: 50
```

```
$ rasa train
$ rasa run
```

The screenshot shows the Rasa X web interface. On the left, there's a sidebar with navigation icons: a magnifying glass for search, a person icon for users, a gear icon for settings, and a plus sign for new conversations. The main area is titled "Conversations (744)" and shows a list of messages from various sources (Facebook, Tester) and times (e.g., 9:51 pm, 3 May 2020; 8:00 pm, 3 May 2020). A message from a user asks "how much it will cost?", and the bot responds with a message bubble containing "The cost per ton depends on the project you donate to. Some projects can remove a ton of CO2 for less than \$1!". Another message from the user asks "ok let me see the available projects", and the bot responds with "I'm not sure but hopefully you can find an answer on the project website.". A message from the user says "ok thanks", and the bot responds with "thank". On the right side of the interface, there's a sidebar with sections: "All time" (calendar icon), "Annotate" (person icon), and "Story till now". Under "Story till now", there are two tabs: "Story" and "End-to-end Story". The "Story" tab lists several slots and their values: -airtravel\_form, -slot["travel\_departure":"Dakhla Airport"], -slot["tata\_departure":"VIL"], -slot["requested\_slot":"travel\_destination"], \* inform["city":"rabat"], -slot["city":["rabat"]], -airtravel\_form, -slot["travel\_destination":"Rabat-Salé Air"], -slot["lata\_destination":"RBA"], -slot["travel\_departure":null], -slot["lata\_departure":null], -slot["travel\_destination":null], -slot["lata\_destination":null], -slot["travel\_flight\_class":null]. Below this is a button labeled "Copy to interactive learning".

## Rasa X Local mode vs Server mode

### Local Mode:

- Great for initial testing, familiarizing yourself with Rasa X
- Conversations get saved to a local database

### Server Mode:

- Great for serving your bot and collecting conversations from many testers or users, 24/7
- Production-ready & scalable
- Deploys Rasa X (and your assistant) using Docker Compose or Kubernetes
- Conversations get saved to a production database
- Includes a Git integration (integrated version control)

# Share your bot with testers using just a link

The screenshot shows the RASA app interface. On the left, there's a sidebar with a downward arrow icon, a share icon, and a lightbulb icon. Below these is a button labeled "Share with guest testers". The main area features a large white box containing a RASA logo and the text: "You have been invited to have a conversation with an AI assistant". To the right of this box is a smaller preview window showing a conversation between a user ("hi") and an AI ("start"). At the bottom of the main area is a message input field with the placeholder "Start typing a message..." and a send icon.

# Review conversations coming in from every channel

The screenshot shows the Rasa Platform's Conversations interface. On the left, a sidebar lists various channels: Direct message, Facebook, LinkedIn, Twitter, and Email. The main area displays a list of 748 conversations, each with a timestamp, date, channel, and a red dot indicating unread messages. A specific conversation is highlighted, showing a back-and-forth between a user and a bot. The user's messages are in grey bubbles, and the bot's messages are in teal bubbles. The conversation details are as follows:

- 4:59 am, 12 May 2020: facebook (User) - respond\_faq: I don't have to put up with any abuse from you.
- 10:39 am, 11 May 2020: facebook (Bot) - action\_listen: well
- 8:50 am, 9 May 2020: facebook (User) - respond\_faq: so
- 5:10 am, 9 May 2020: facebook (Bot) - action\_listen: thanks
- 9:14 am, 8 May 2020: facebook (User) - action\_listen: :-)
- 2:46 am, 8 May 2020: Facebook (Bot) - utter\_express\_positive\_emo: bye
- 12:44 pm, 7 May 2020: facebook (User) - action\_listen: :-)
- 9:51 pm, 3 May 2020: facebook (Bot) - action\_listen: bye

At the top right, there is a filter bar set to "All time" and a search icon. To the right of the main list, a summary for a conversation between Demobot and a user with ID 3452770921417556 is shown, indicating 3 link-2-clicked interactions.

## Annotate the messages coming in

Search 63340 logs...

Sentence Predicted Intent and Confidence

|                                                                                  |                          |                                                |                        |
|----------------------------------------------------------------------------------|--------------------------|------------------------------------------------|------------------------|
| <input type="checkbox"/> how to start building your first AI assistant with Rasa | how_to_get_started (1.0) | <input checked="" type="button"/> Mark Correct | <input type="button"/> |
| <input type="checkbox"/> what sub                                                | out_of_scope (0.71)      |                                                |                        |
| <input type="checkbox"/> Installation windows<br>7                               | install_rasa (1.0)       |                                                |                        |

# Push new training data to git and trigger your CI pipeline



Commit changes to git

Status: You have changes the Git server does not include.

Commit these changes to:

master  
 New branch

Note: When you add your changes, you will return to match the latest Git server changes. You must merge your new branch into master on the Git server to see those changes reflected here.

[Cancel](#) [Add changes](#)



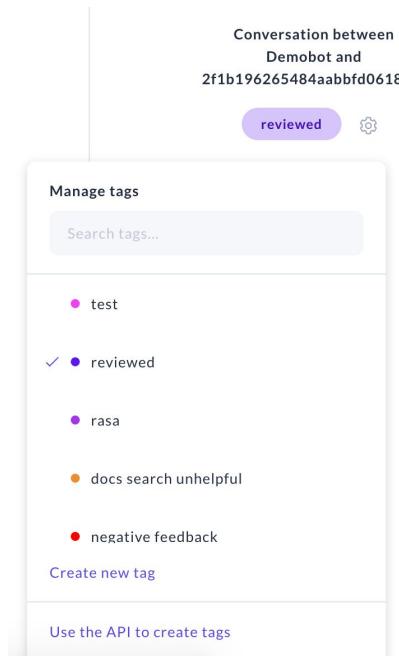
- ▶ ✓ Train model
- ▶ ✓ Run Through Test Stories
- ▶ ✓ Cross-validate NLU model
- ▶ ✓ post cross-val results to PR

## Intent Cross-Validation Results (5 folds)

| class        | support | f1-score | confused_with                                     |
|--------------|---------|----------|---------------------------------------------------|
| macro avg    | 2541    | 0.7999   | N/A                                               |
| weighted avg | 2541    | 0.8392   | N/A                                               |
| faq          | 753     | 0.8172   | inform(29), inquire-ask_clarification-offsets(17) |
| inform       | 619     | 0.9367   | faq(24), estimate_emissions(6)                    |

# Track failures and successes

Use the API to automatically tag conversations, or add tags manually as you read



Turn successful conversations into new end-to-end tests

The screenshot shows the 'Story till now' section of the Rasa UI. It includes tabs for 'Story' and 'End-to-end Story'. The 'Story' tab is active, showing a code block with a copy icon:

```
Story from conversation with 2f1b19626548
- slot{"shown_privacy":true}
- slot{"step":"2"}
* affirm: yes
- utter_thumbsup
* how_to_get_started: how to start building your
- utter_getstarted
- utter_first_bot_with_rasa
* affirm: yes
- action_set_onboarding
- slot{"onboarding":true}
- utter_built_bot_before
* deny: no
```

Below the code block is a large button labeled 'Save end-to-end test'.

## Installing Rasa X Locally

Checkout the master branch

```
git checkout master
```

Install Rasa X in your virtual environment:

```
pip3 install rasa-x --extra-index-url https://pypi.rasa.com/simple
```

Launch Rasa X in the browser by running the following command in your project directory:

```
rasa x
```

## Testing Locally: ngrok

Creates a secure, publicly accessible tunnel URL connected to a process running on localhost.

Allows you to develop as though your application were running on a server instead of locally.

```
ngrok by @inconshreveable

Session Status online
Session Expires 7 hours, 59 minutes
Version 2.3.35
Region United States (us)
Web Interface http://127.0.0.1:4040
Forwarding http://87ee9a2c9b3b.ngrok.io -> http://localhost:5005
Forwarding https://87ee9a2c9b3b.ngrok.io -> http://localhost:5005

Connections ttl opn rt1 rt5 p50 p90
 0 0 0.00 0.00 0.00 0.00
```

## Starting ngrok

1. Start Rasa X. When you start Rasa X locally, it runs on localhost:5002
2. Open a new terminal window and start ngrok. Specify that ngrok should listen for the process running on port 5002
  - a. ngrok http 5002
3. Save the URL generated by ngrok
4. Generate a Share your Bot link in Rasa X.
5. In the Share Your Bot URL, replace localhost:5002 with your ngrok URL

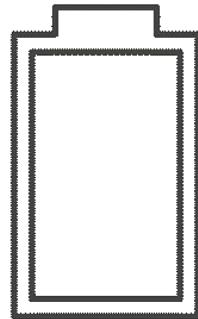
<http://92099d45e813.ngrok.io/quest/conversations/production/76a25a7880ac40f5bda7b42d650bc>  
a75

## Time to pair up!

- You should have been paired with another workshop participant by the Shuffl Slack app. If your partner isn't online or you didn't get matched, post in the the #workshop-help channel and someone can match with you.
- Start a DM with your partner
- Exchange Share your bot links to test your assistant! Remember, you'll need to replace

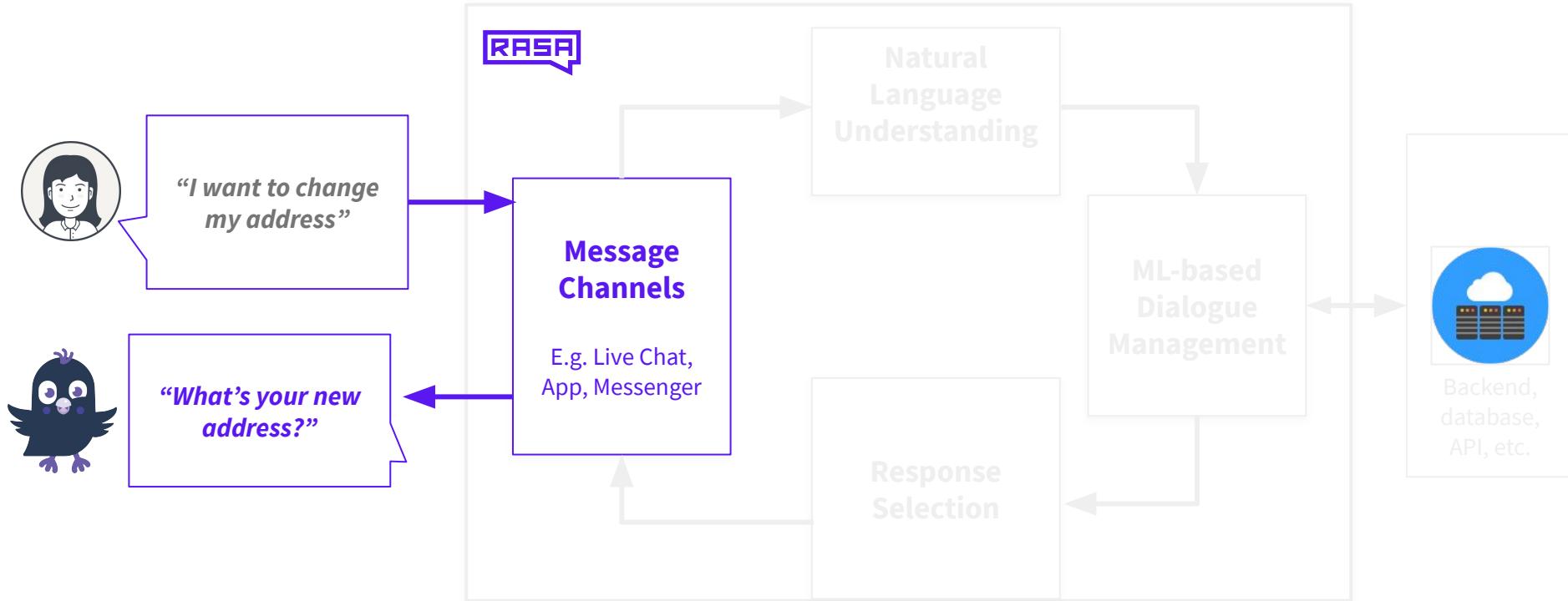
**Time for a short break (10 minutes)**

# HUMANS RECHARGING



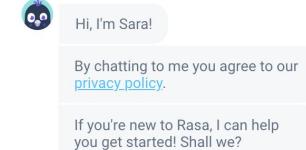
# Messaging Channels

# Rasa Open Source



## MESSAGING CHANNELS

### Built in Channels



# Telegram



# Messaging Channels

- Channel connection details are kept in the **credentials.yml** file
- When you move to production, you'll want to re-create these credentials securely on your server
- After you update credentials.yml, start (or restart) the server with `rasa run`



```
! config.yml ! domain.yml • ! credentials.yml ×
financial-demo > ! credentials.yml
27
28 # pw: "<bot token>"
29 # webhook_url: "<callback URL>"
30
31 rasa:
32 url: "http://localhost:5002/api"
33
34 telegram:
35 access_token: "1292722675:AAGwNrGwTrIv0eZxjItqwZLhZxakDuGLVvY"
36 verify: "rasaworkshop_bot"
37 webhook_url: "https://e6e218a18360.ngrok.io/webhooks/telegram/webhook"
38
```

# Telegram

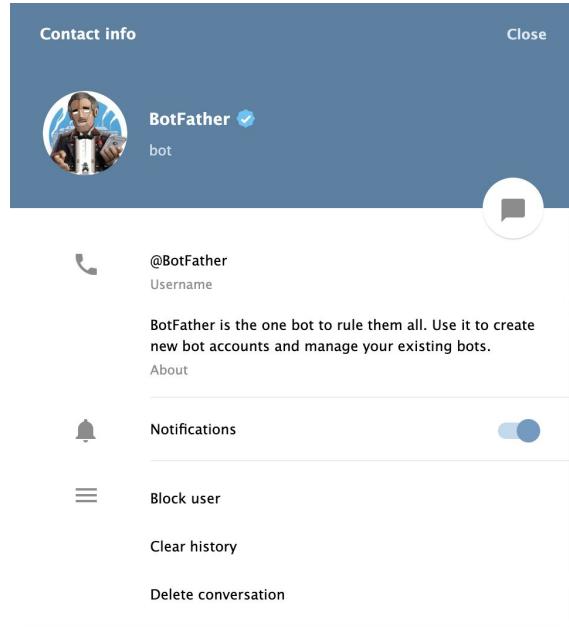
Register a new bot:

- Go to <https://web.telegram.org/#/im?p=@BotFather>
- Type /newbot
- Name shows up next to the bot's profile icon
- Username is the bot's handle, and ends in \_bot

credentials.yml

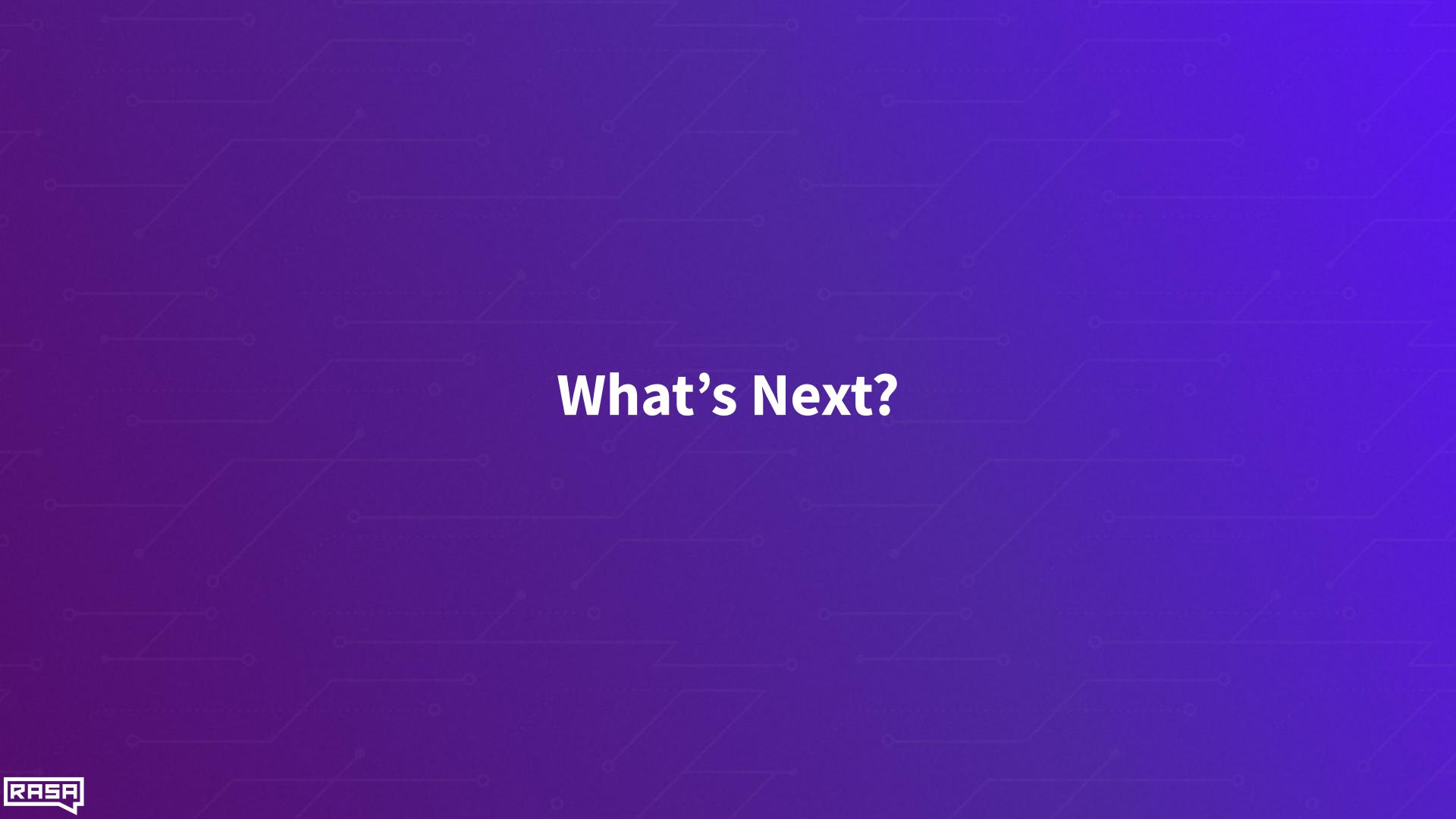
Telegram:

```
access_token: "your API token"
verify: "bot's username (ends in _bot)"
webhook_url: "https://<ngrok-url>/webhooks/telegram/webhook"
```



## Testing Telegram locally

1. Start the rasa server (this command starts the server without a chat session on the command line)
  - a. rasa run
2. Start ngrok on port 5005
  - a. ngrok http 5005
3. Add the ngrok URL to your Telegram webhook URL - save the file
4. Restart the Rasa server (so changes to credentials.yml get picked up)
  - a. /stop
  - b. rasa run
5. Make sure your action server is running!
6. Try talking to your bot on Telegram



# What's Next?

## Get people to chat with your assistant and keep improving it!



### Where to go from here?

1. Look at conversations and ask:  
how does your assistant struggle?
2. Learn how to solve problem from  
[Rasa Docs](#) and [Community Forum](#)
3. Improve assistant
4. Test and deploy your update
5. Repeat 😎

## THE APPROACH

# 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)**

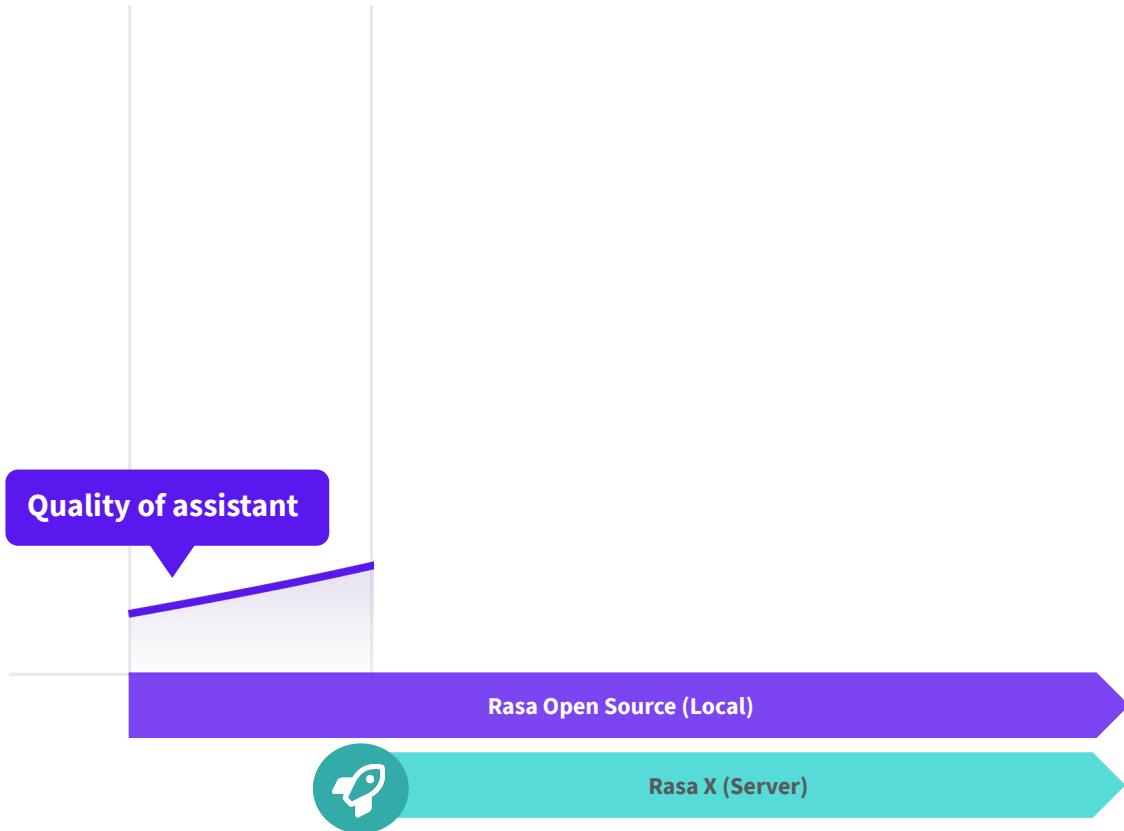


Collect conversations between users and your assistant

Review conversations and improve your assistant based on what you learn

THE APPROACH

## The path to a contextual assistant



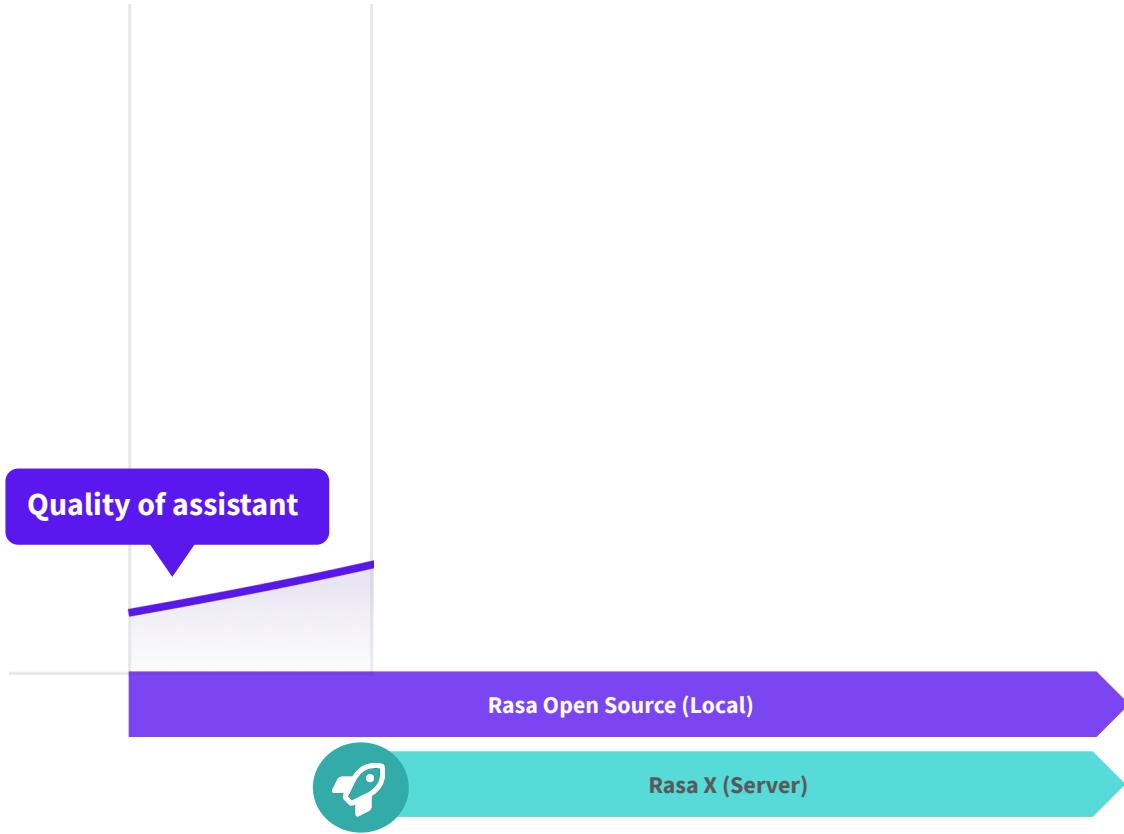
THE APPROACH

## The path to a contextual assistant



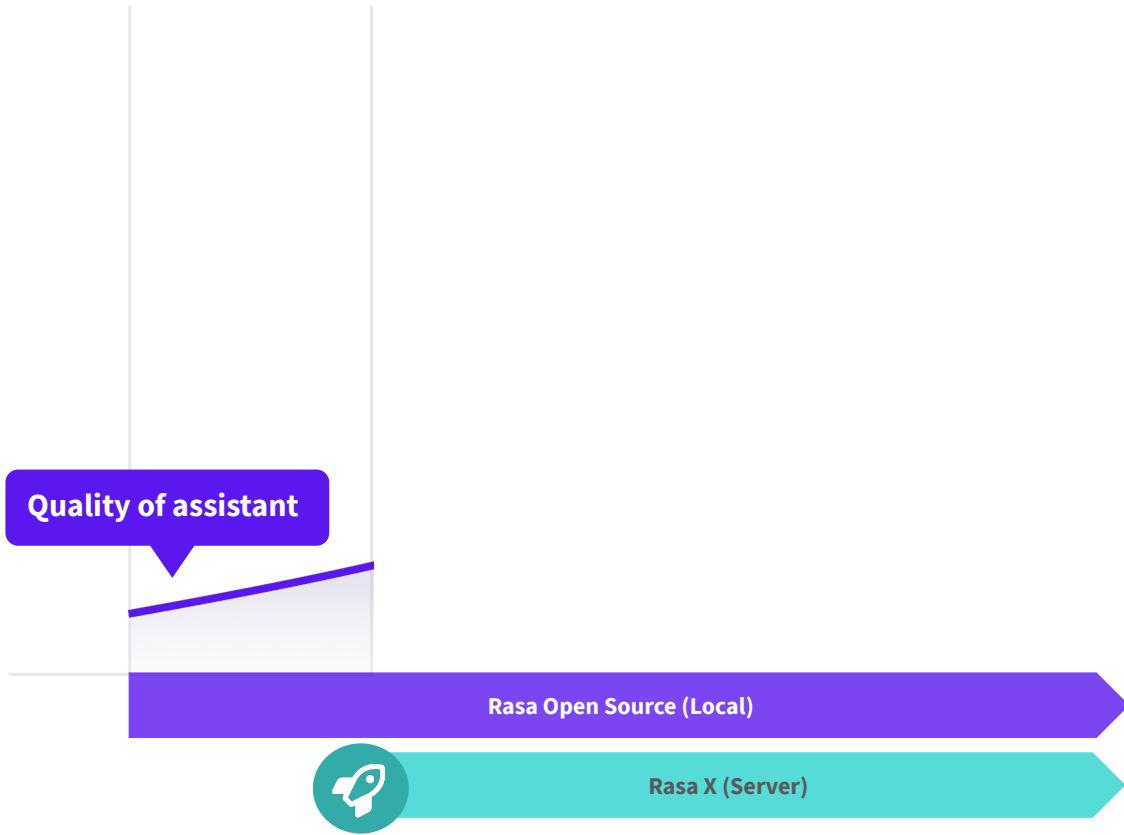
THE APPROACH

## The path to a contextual assistant



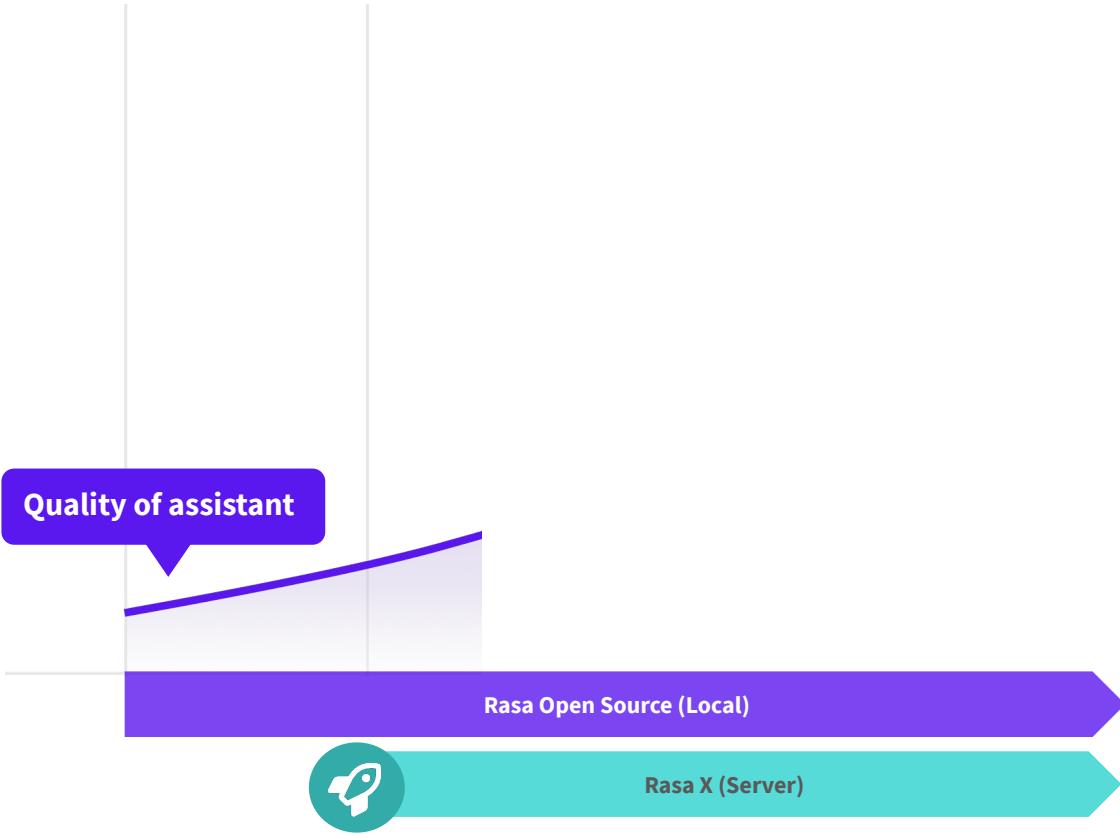
THE APPROACH

## The path to a contextual assistant



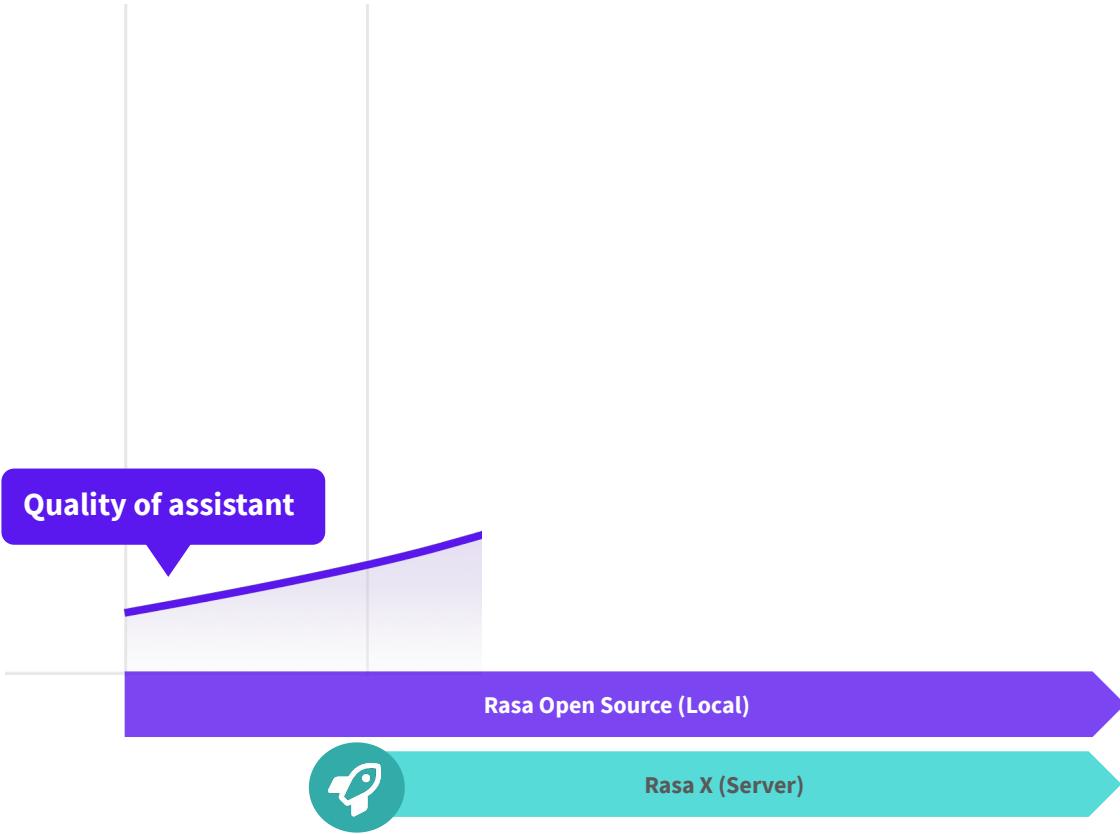
THE APPROACH

## The path to a contextual assistant



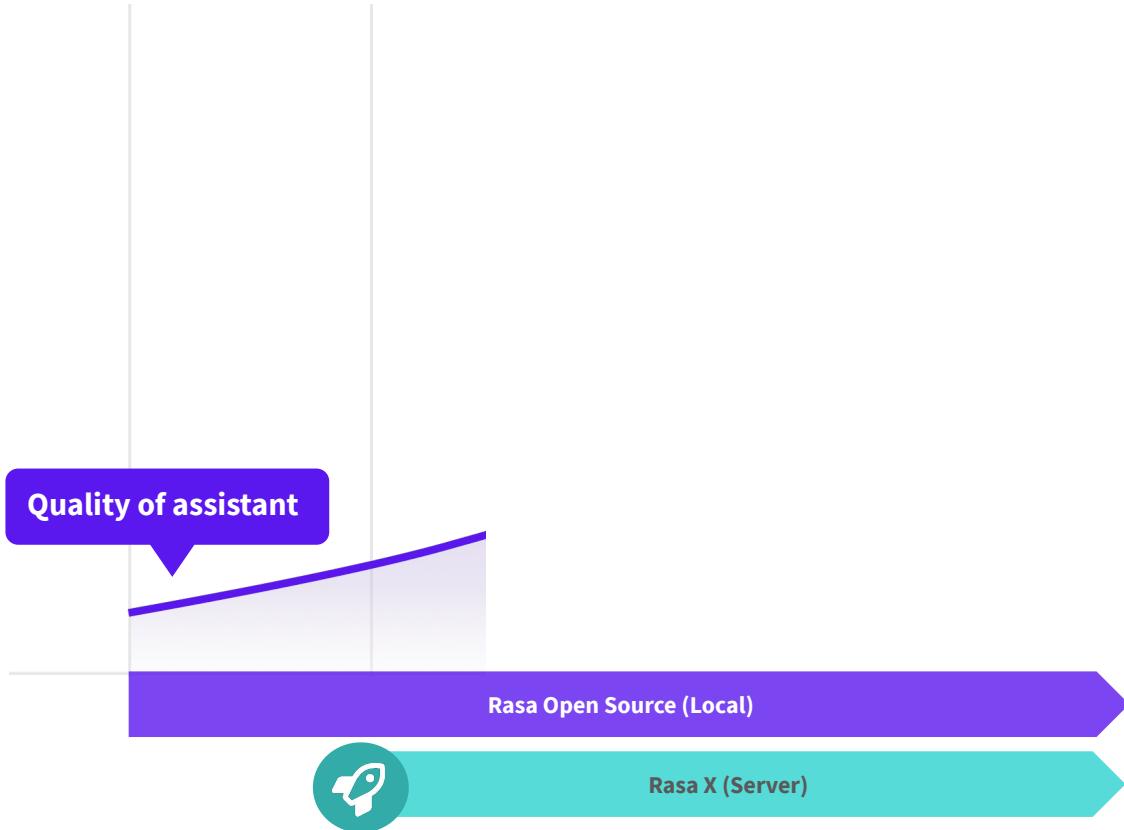
THE APPROACH

## The path to a contextual assistant



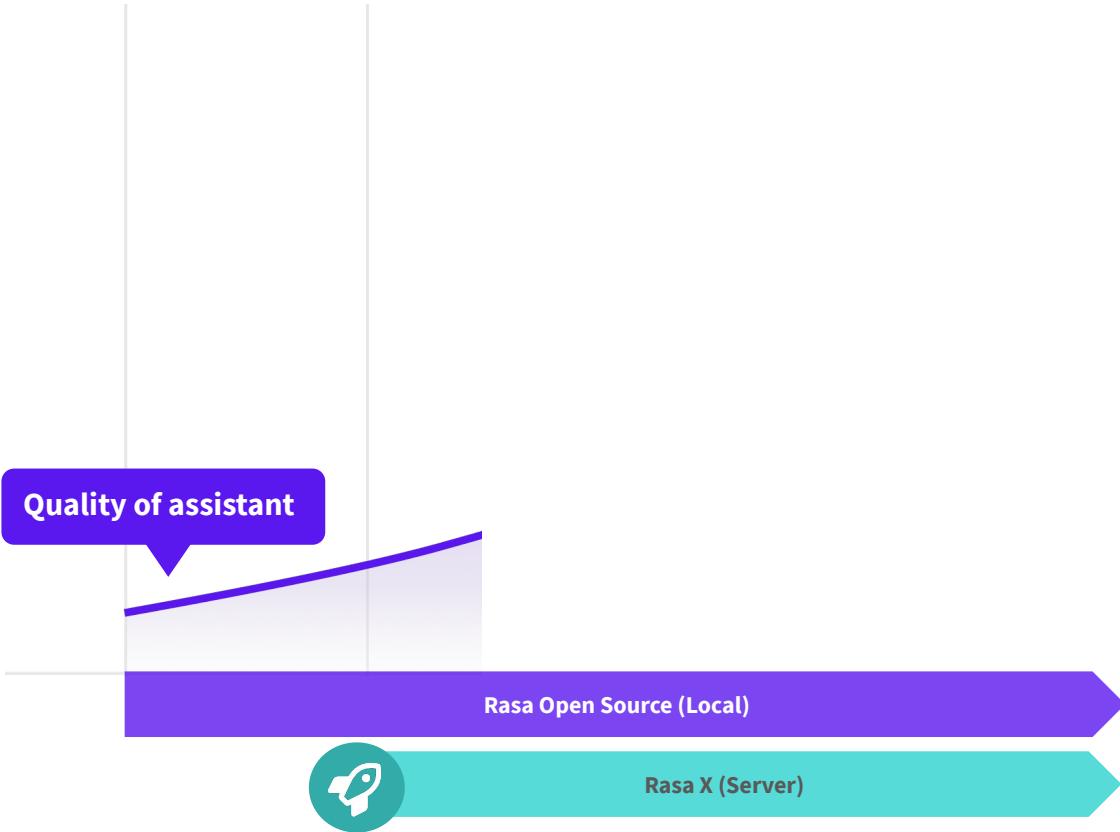
THE APPROACH

## The path to a contextual assistant



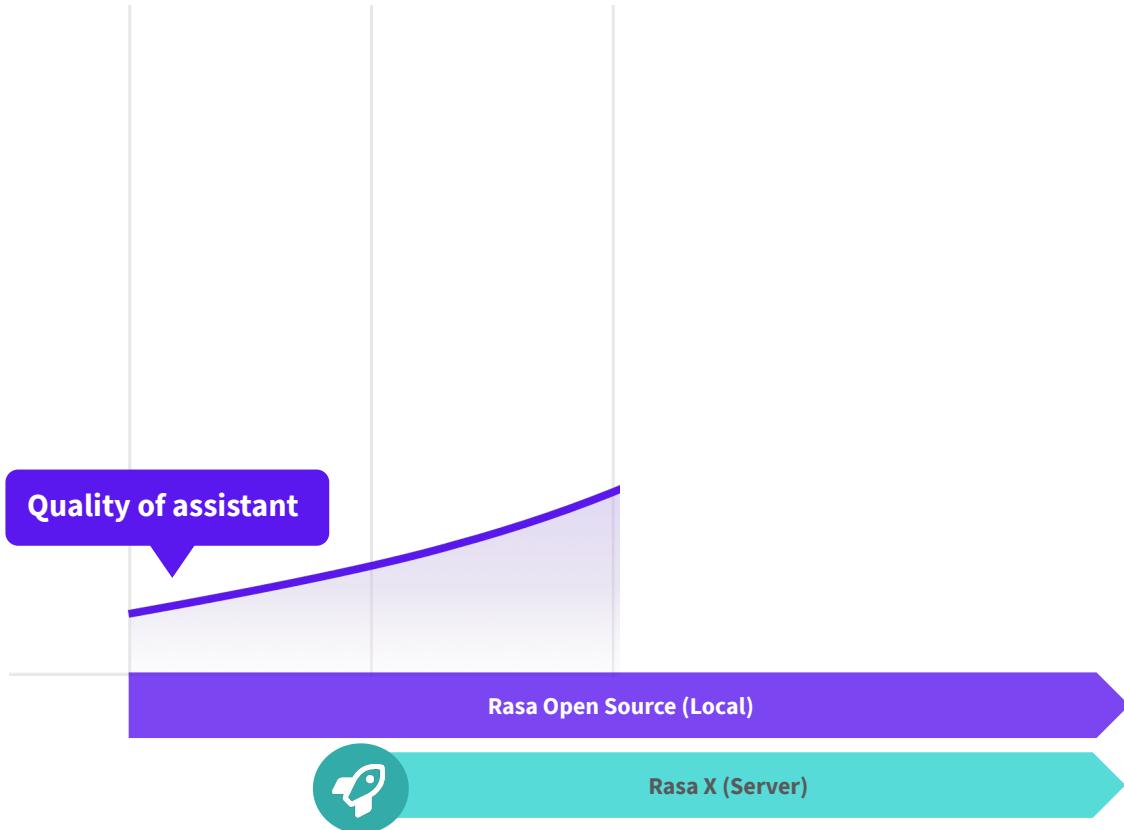
THE APPROACH

## The path to a contextual assistant



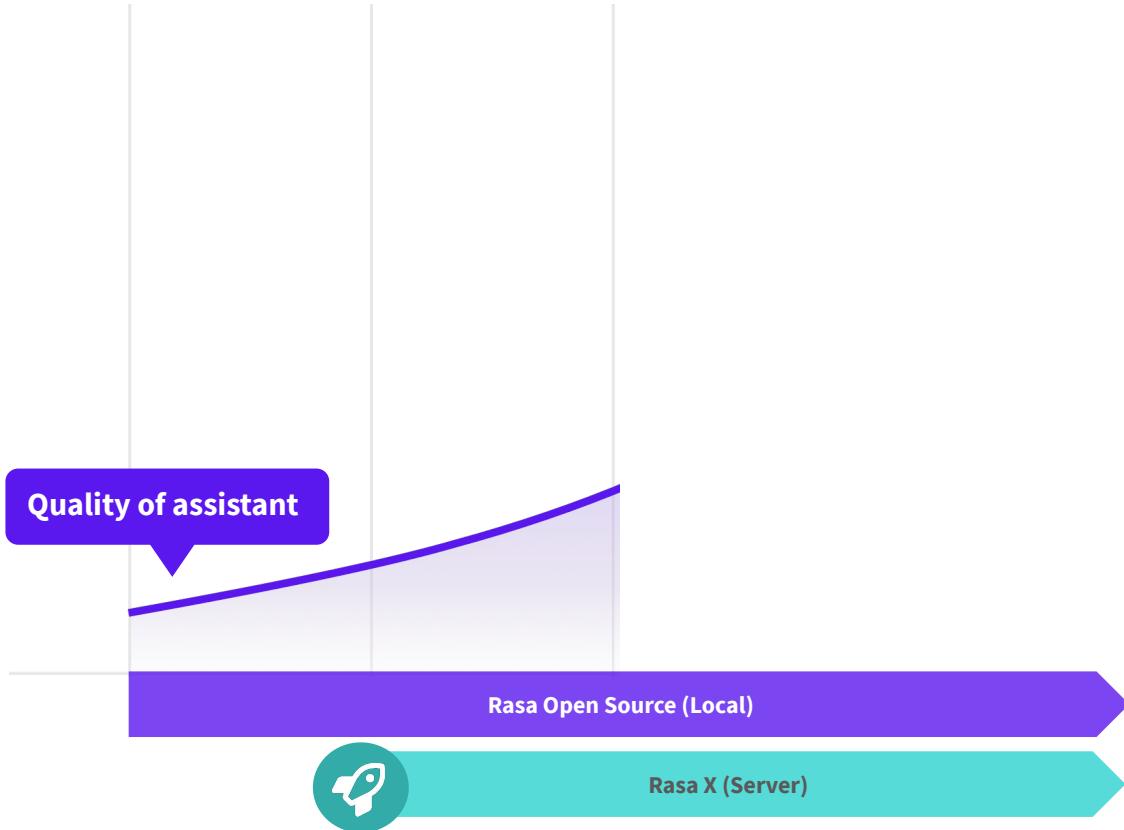
THE APPROACH

## The path to a contextual assistant



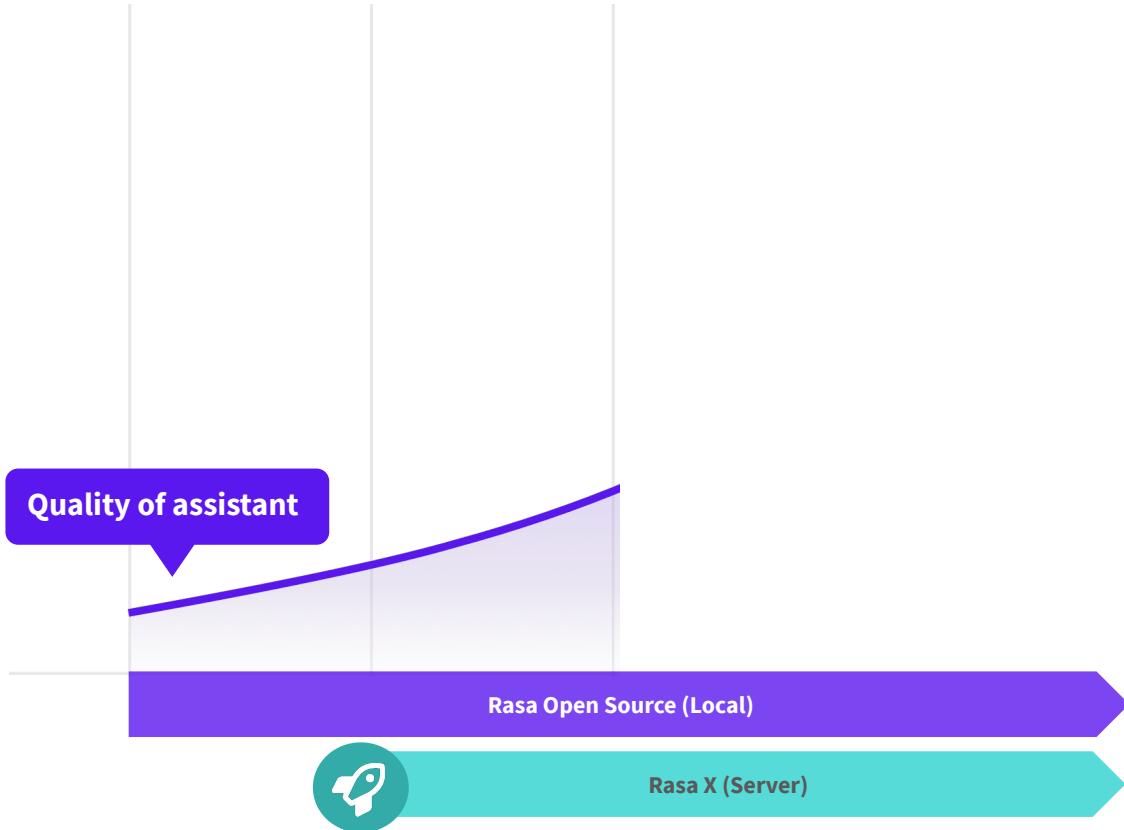
THE APPROACH

## The path to a contextual assistant



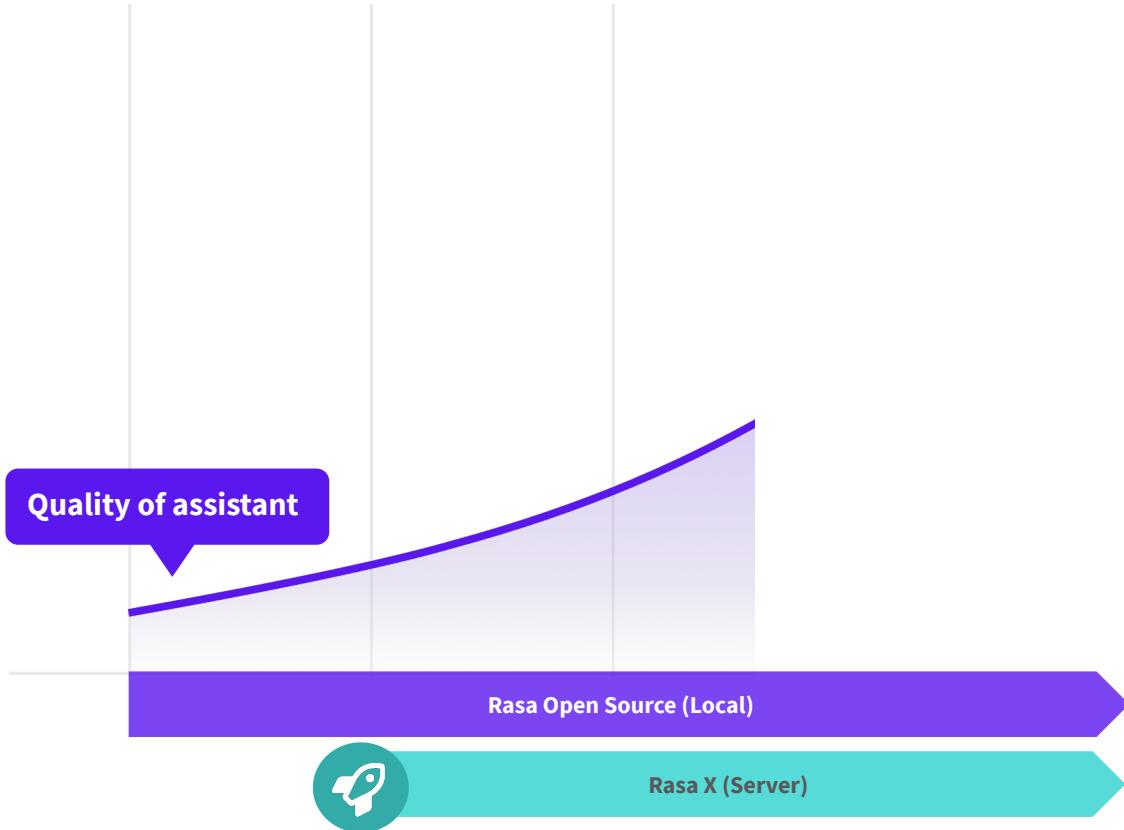
THE APPROACH

## The path to a contextual assistant



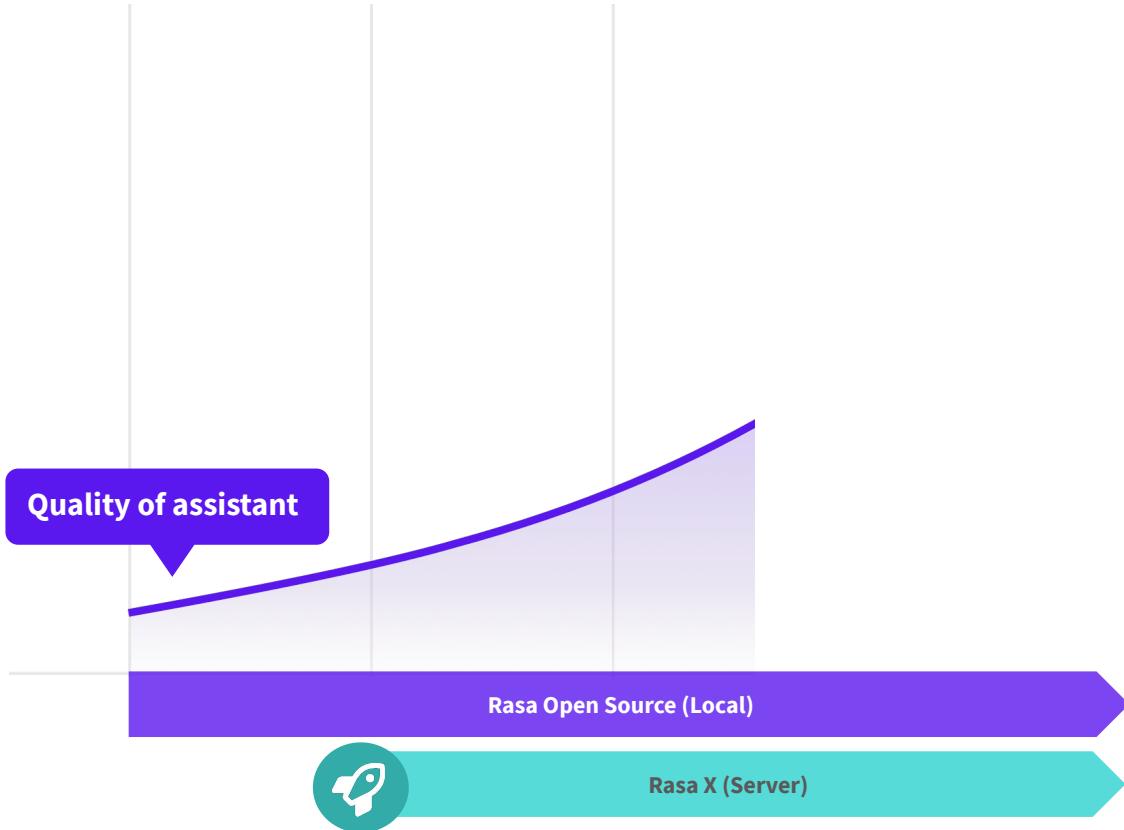
THE APPROACH

## The path to a contextual assistant



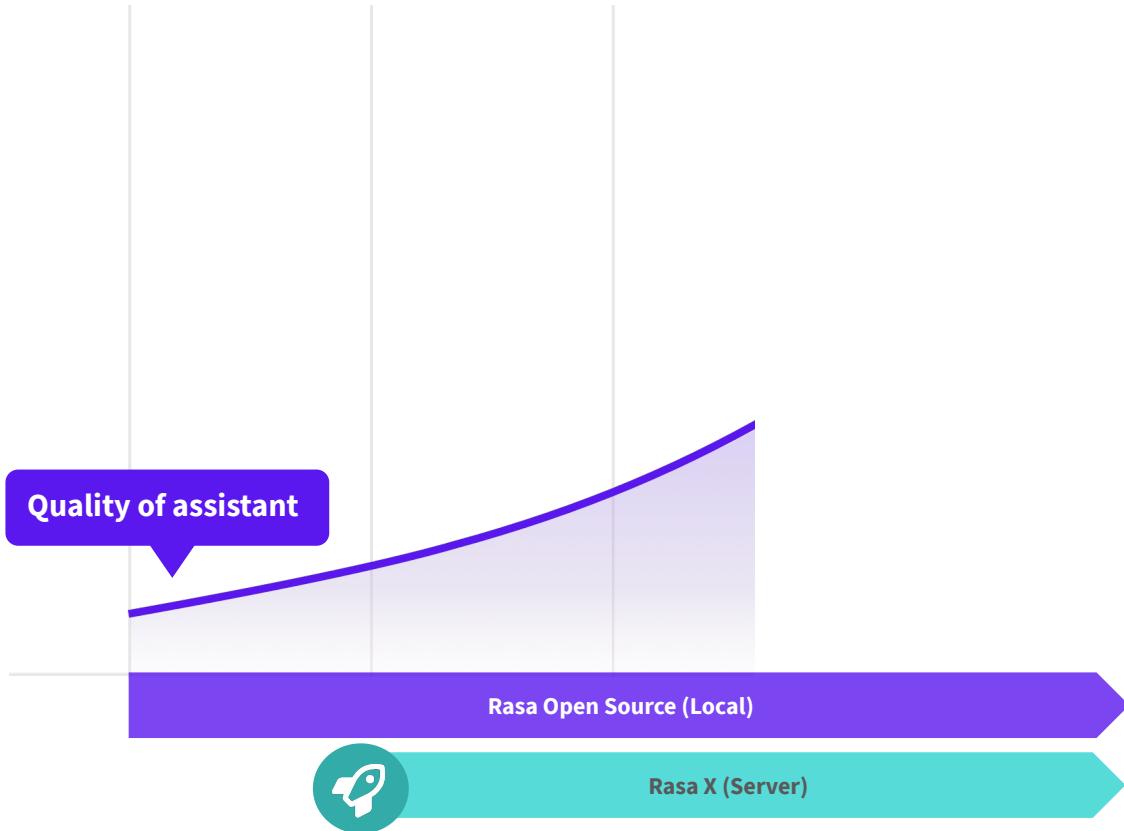
THE APPROACH

## The path to a contextual assistant



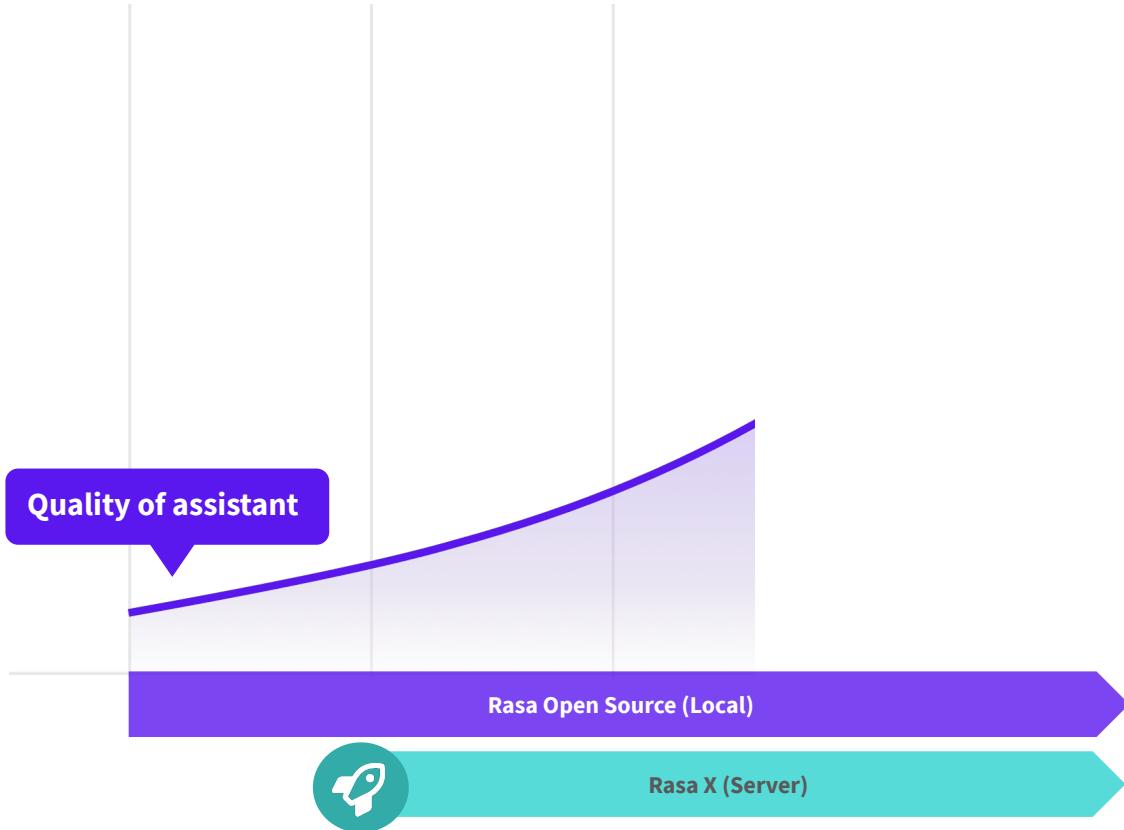
THE APPROACH

## The path to a contextual assistant



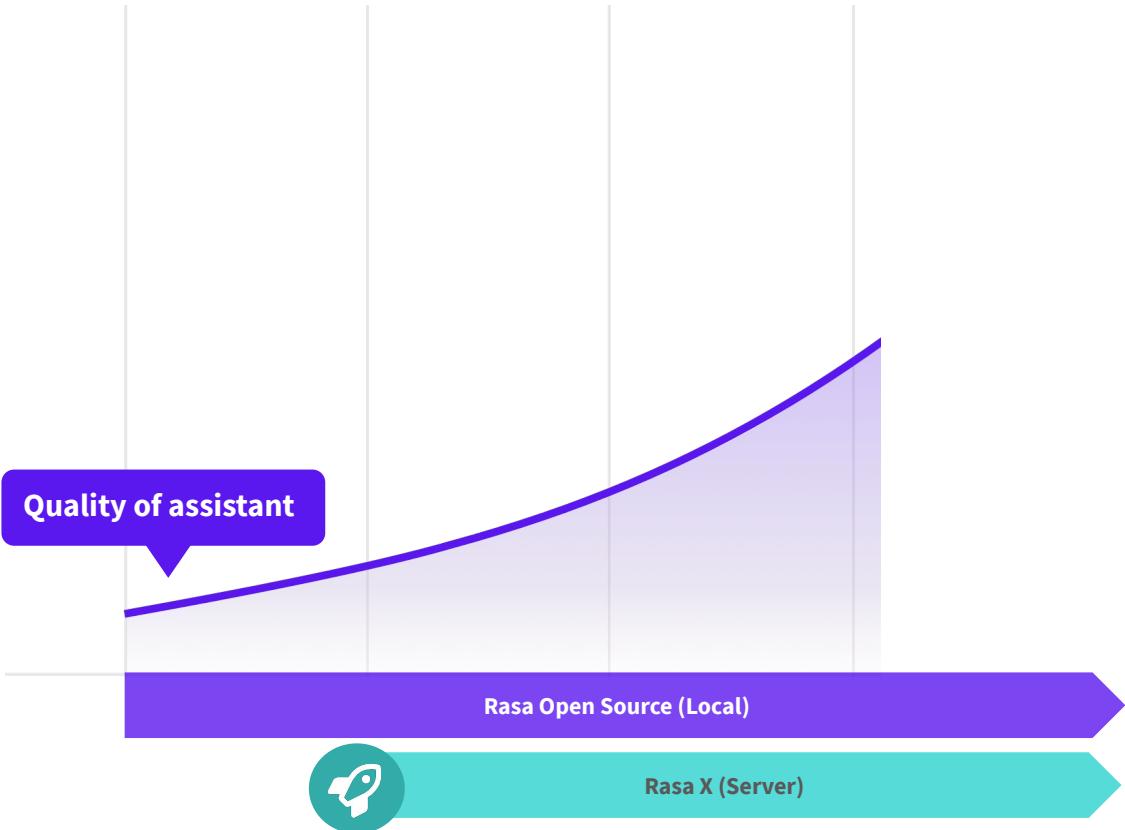
THE APPROACH

## The path to a contextual assistant



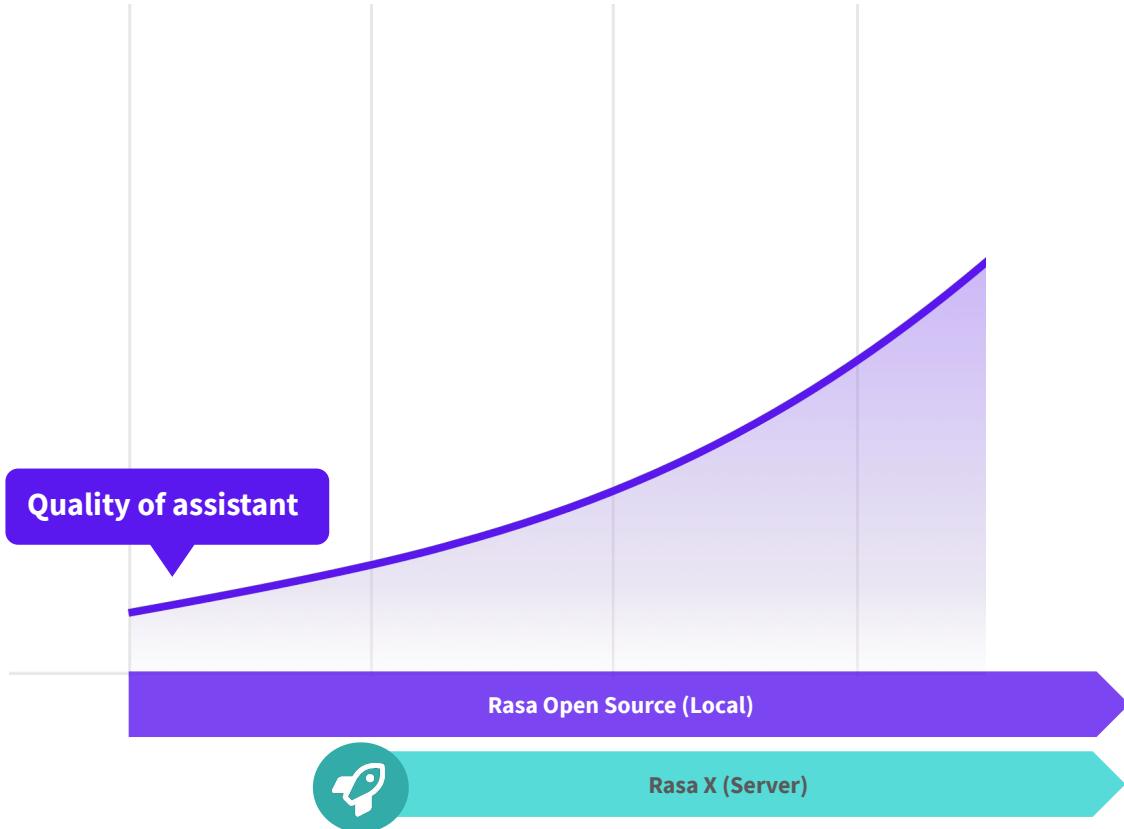
THE APPROACH

## The path to a contextual assistant



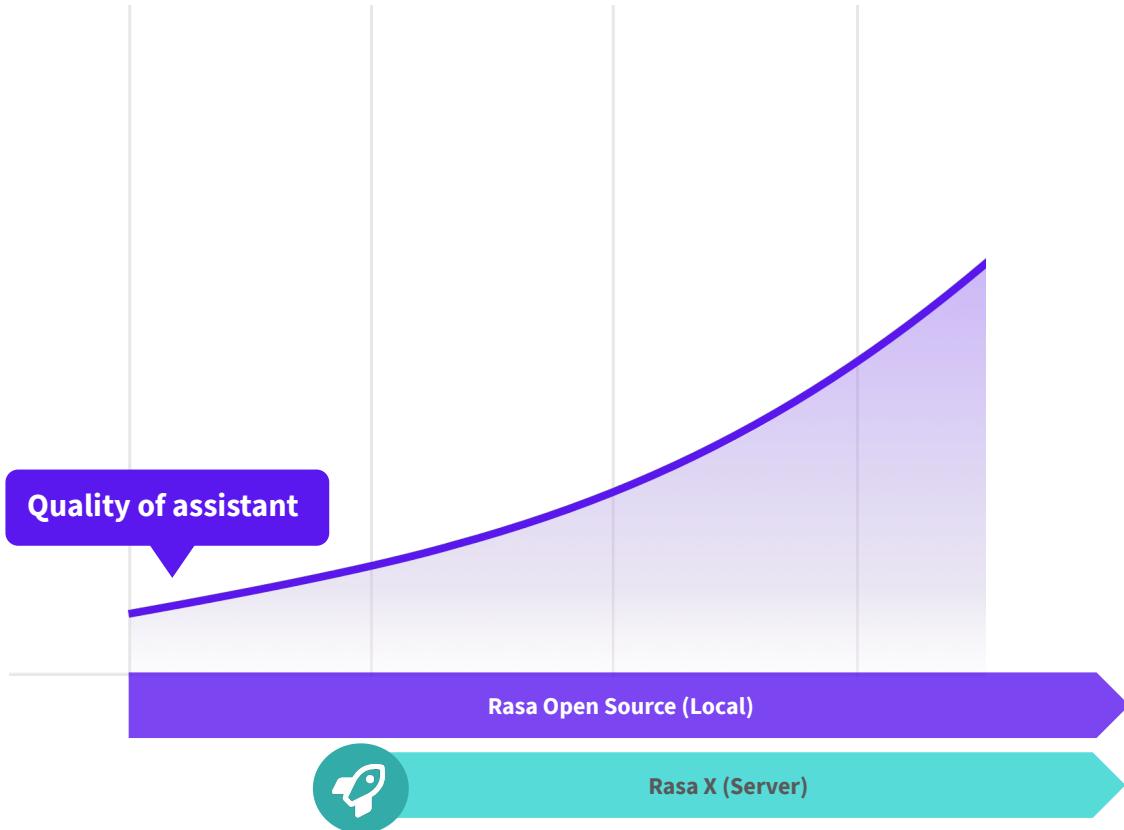
THE APPROACH

## The path to a contextual assistant



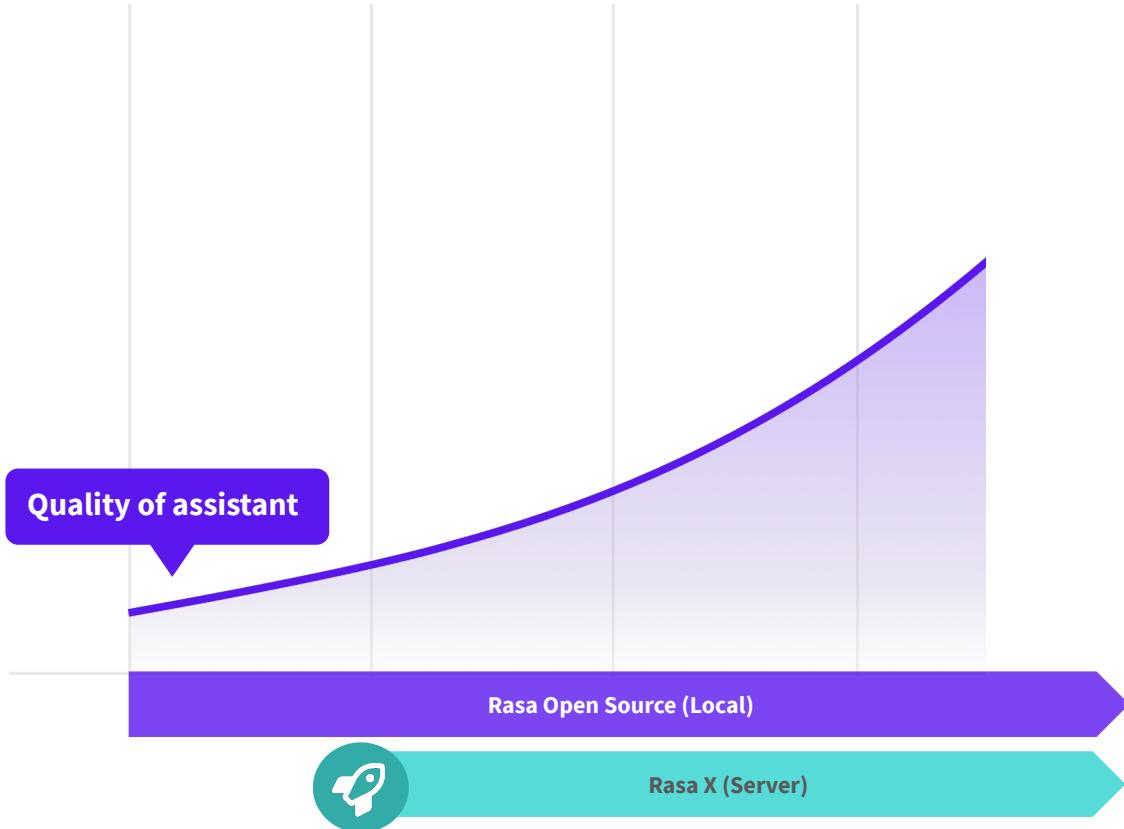
THE APPROACH

## The path to a contextual assistant



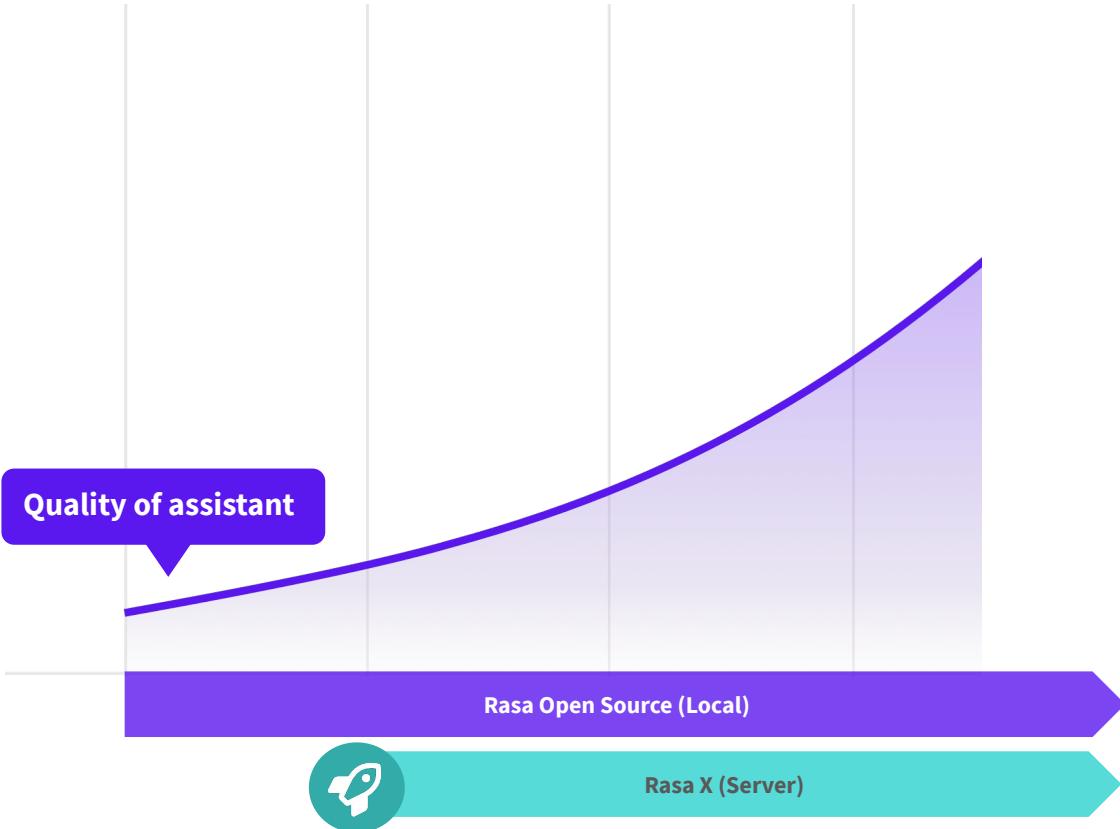
THE APPROACH

## The path to a contextual assistant



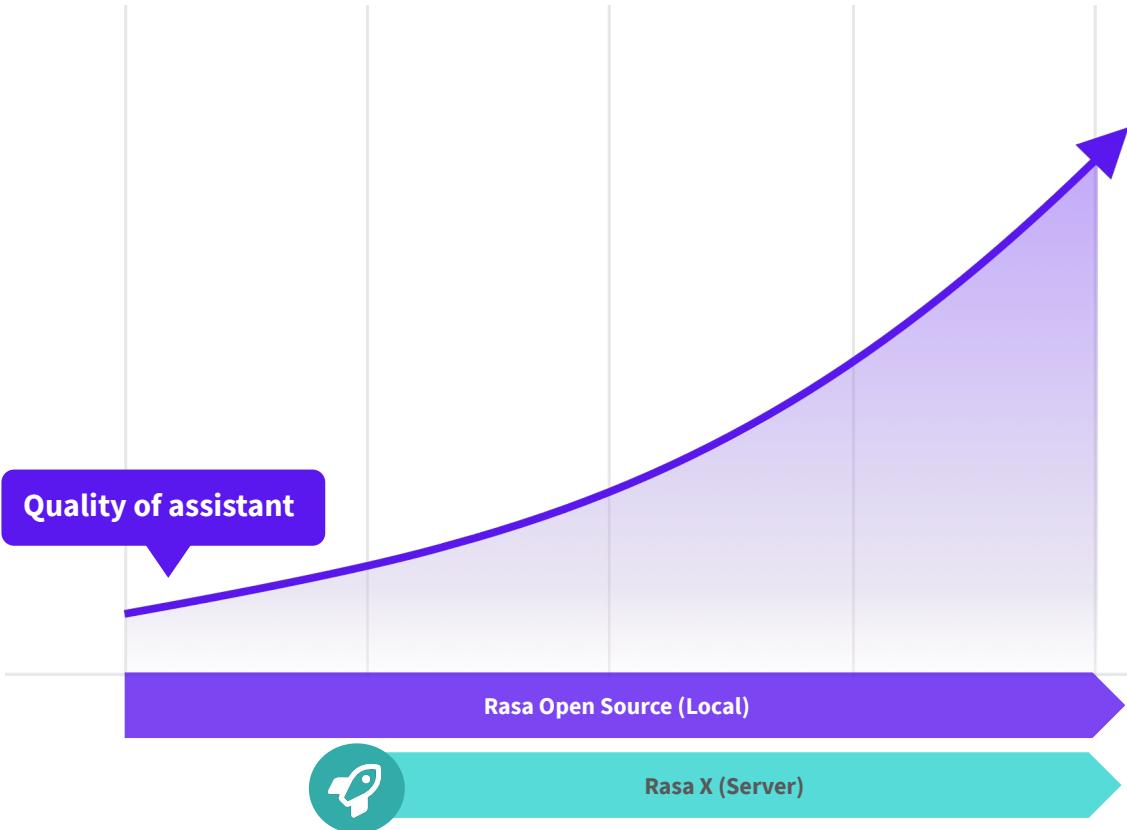
THE APPROACH

## The path to a contextual assistant



THE APPROACH

## The path to a contextual assistant



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

Talk to your bot

Build minimum viable assistant

Improve by talking to the assistant

Improve using conversations with test users

Improve using conversations with real users

Quality of assistant

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



Rasa Open Source (Local)

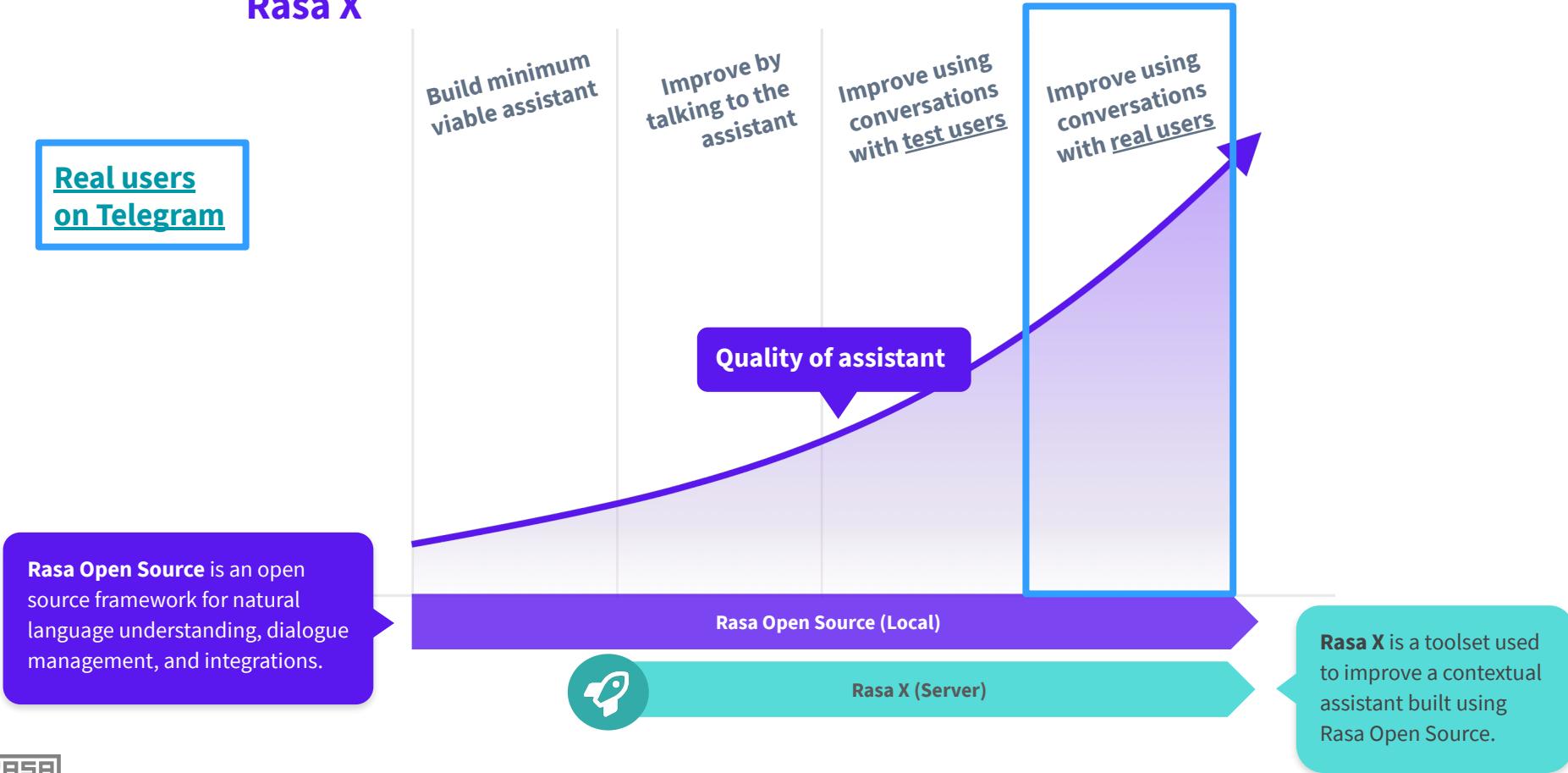
Rasa X (Server)

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

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

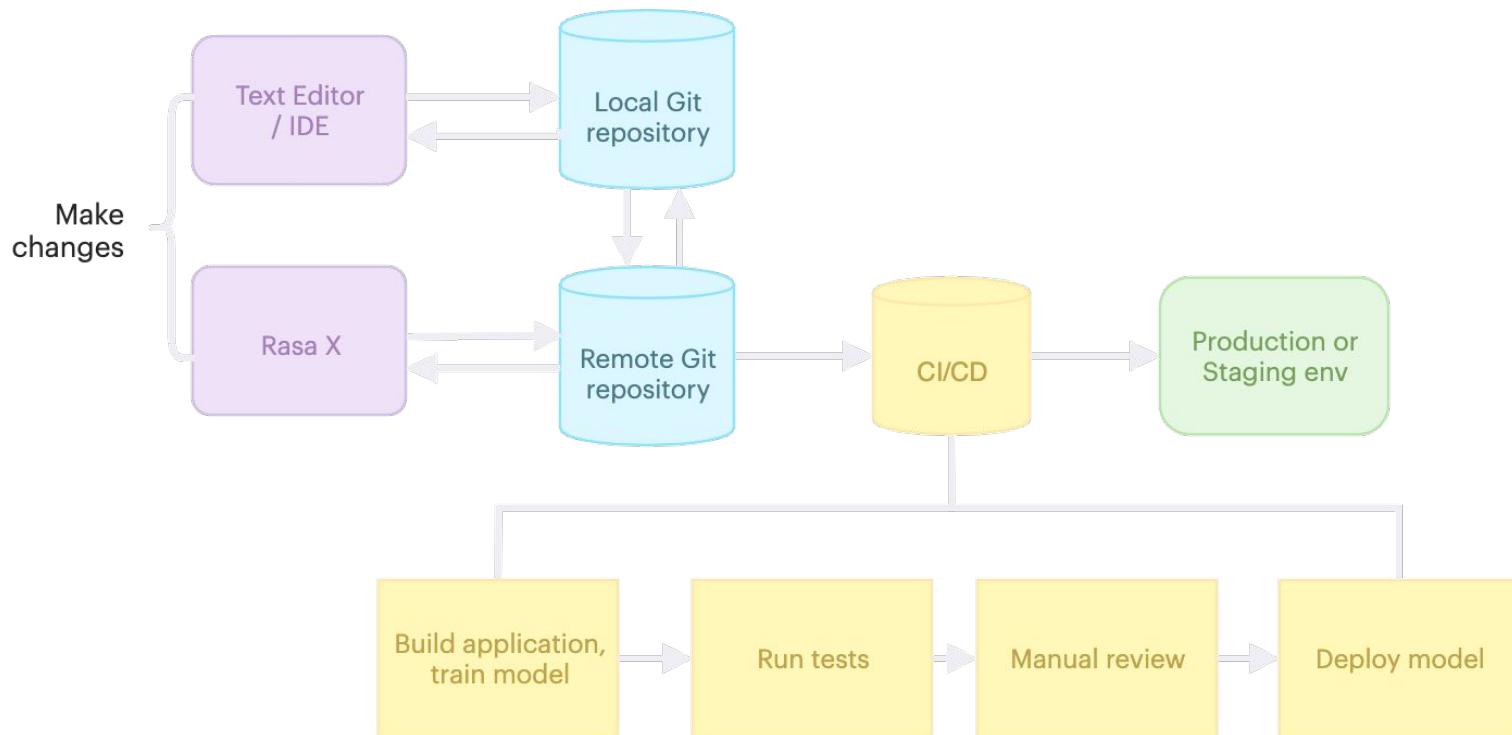


# Build a minimum viable assistant with Rasa Open Source + improve it using Rasa X



DEVELOPMENT WORKFLOW

## End-to-end workflow for managing assistant updates

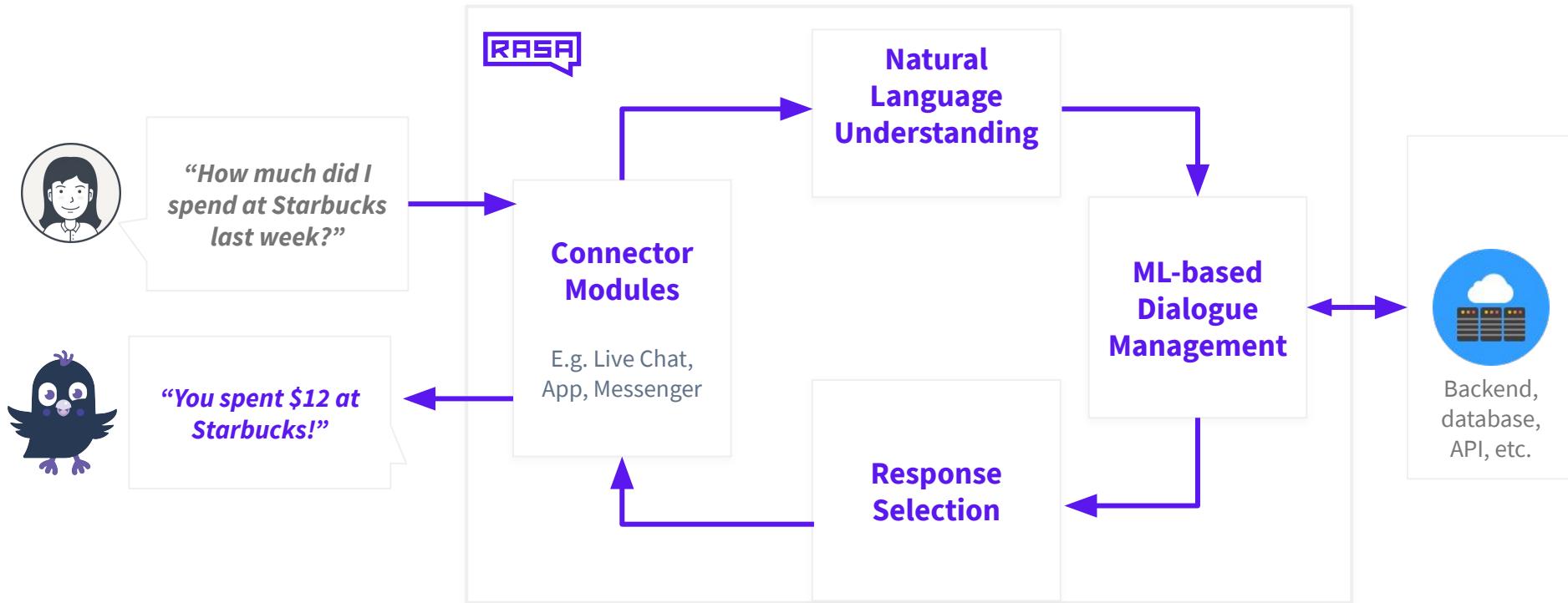


# Recap

## What we've learned so far

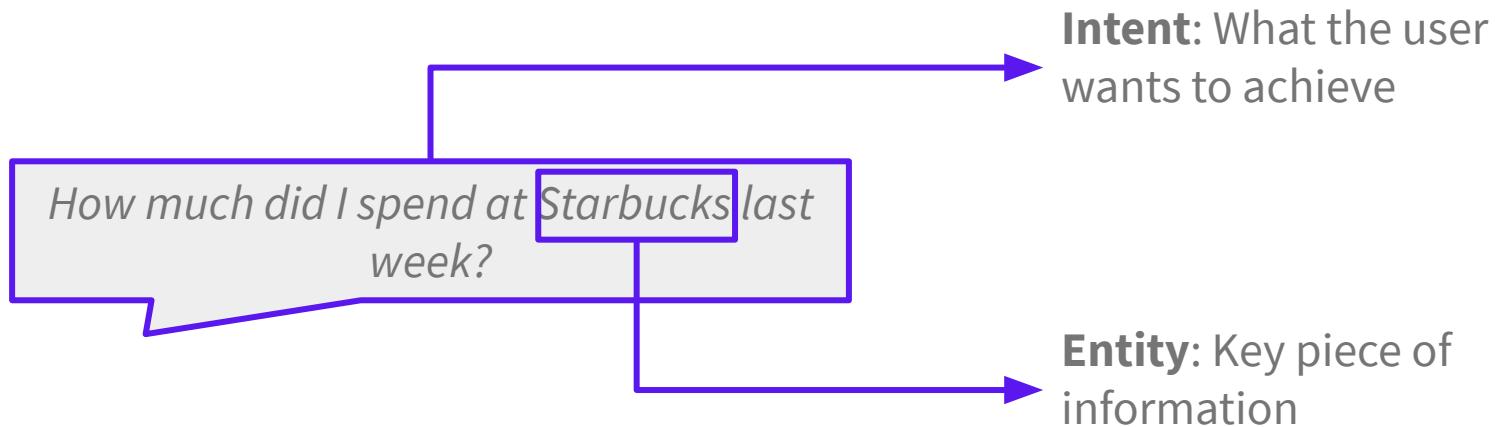
- Set up
- Deep dive of NLU and dialogue management
- Build an MVP assistant
- Share your assistant with the outside world
- Make iterative improvements and take your assistant to the next level

# Rasa Open Source



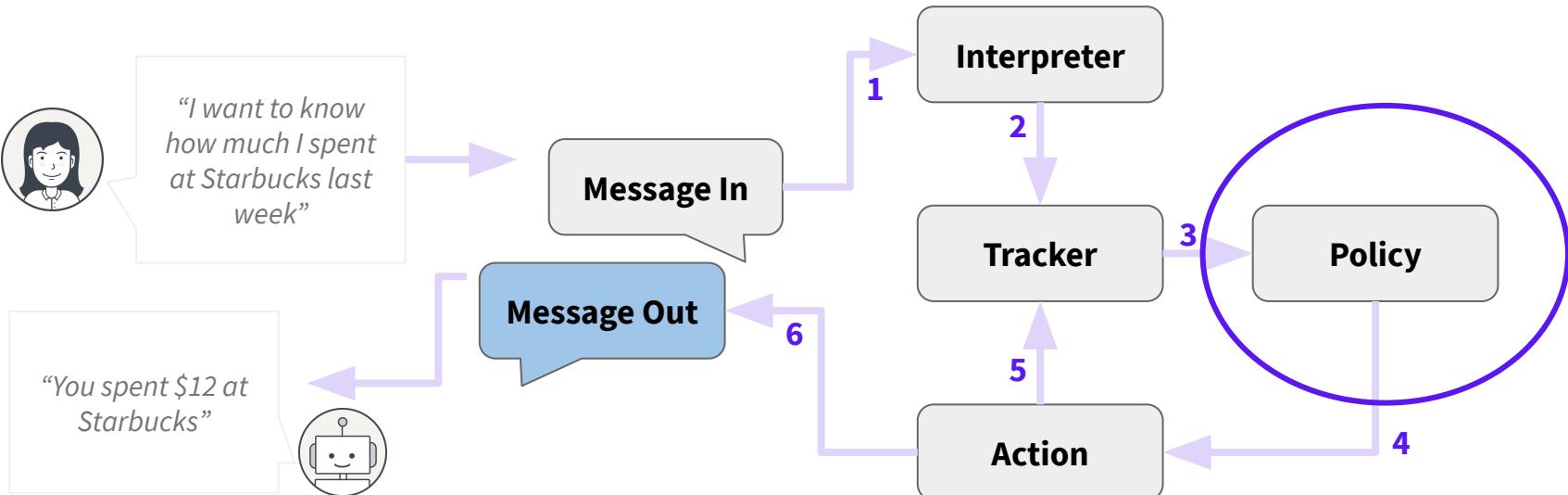
## Intents and Entities

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



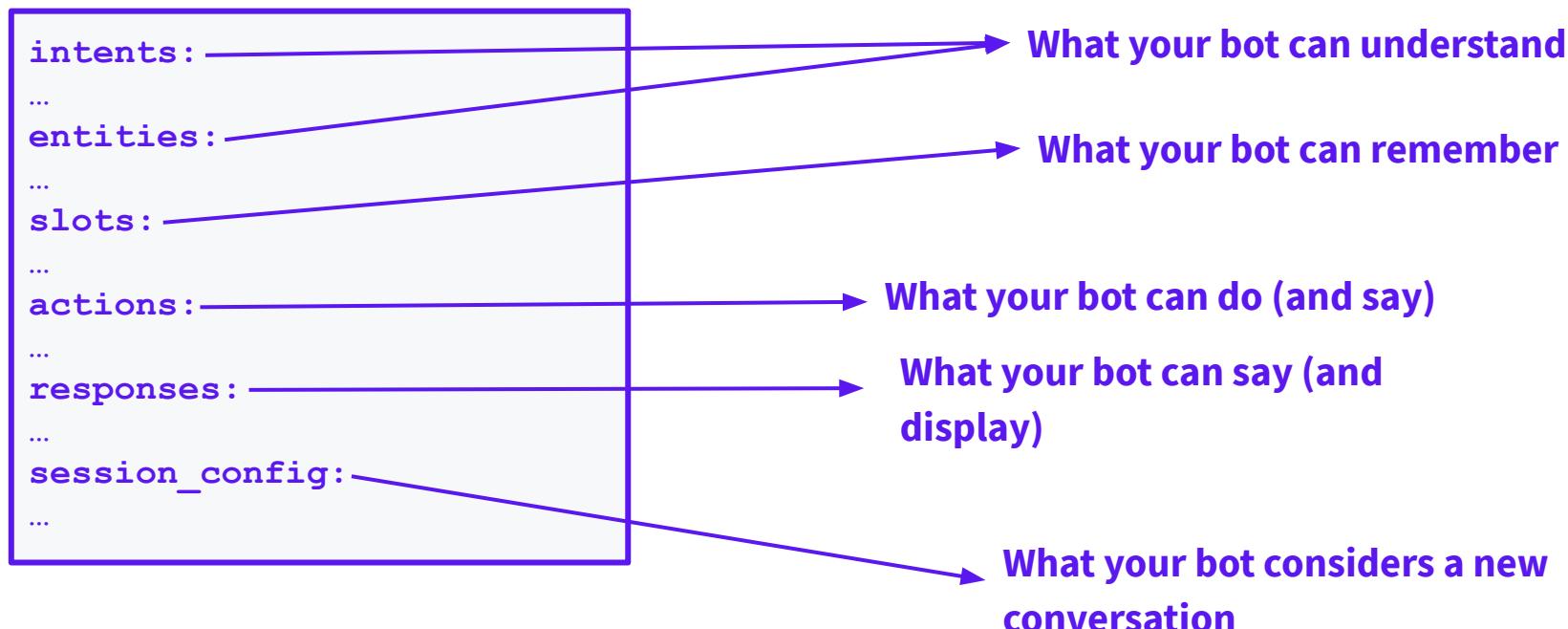
## Policies

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



## Domain

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



## 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

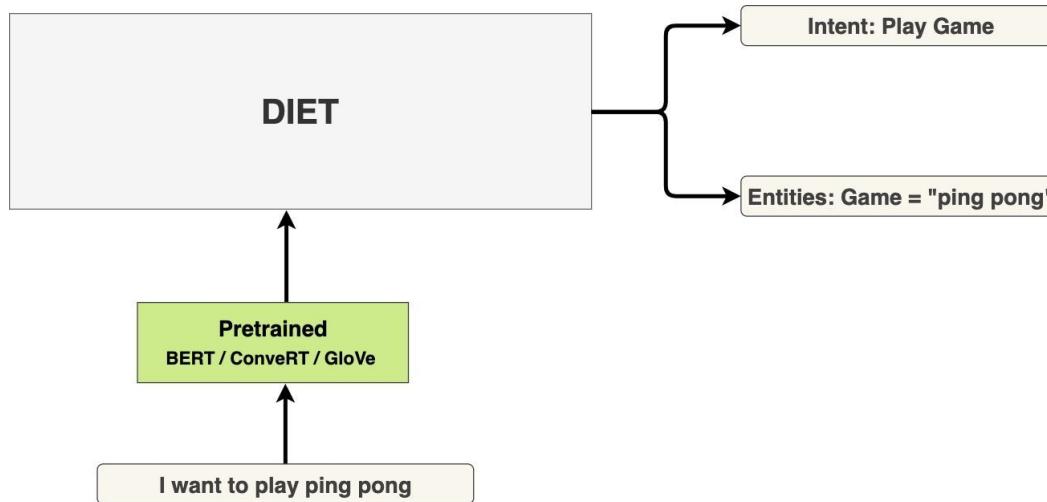
(higher numbers = higher priority)

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

## DIET is our new neural network architecture for NLU

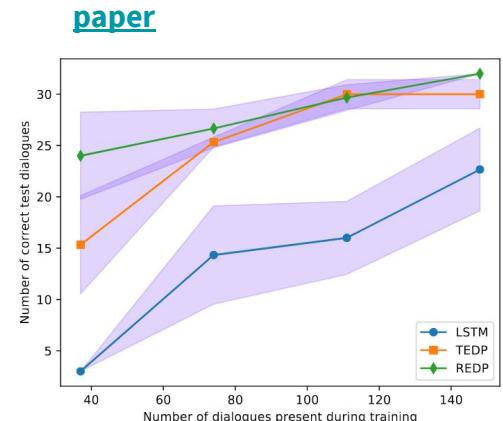
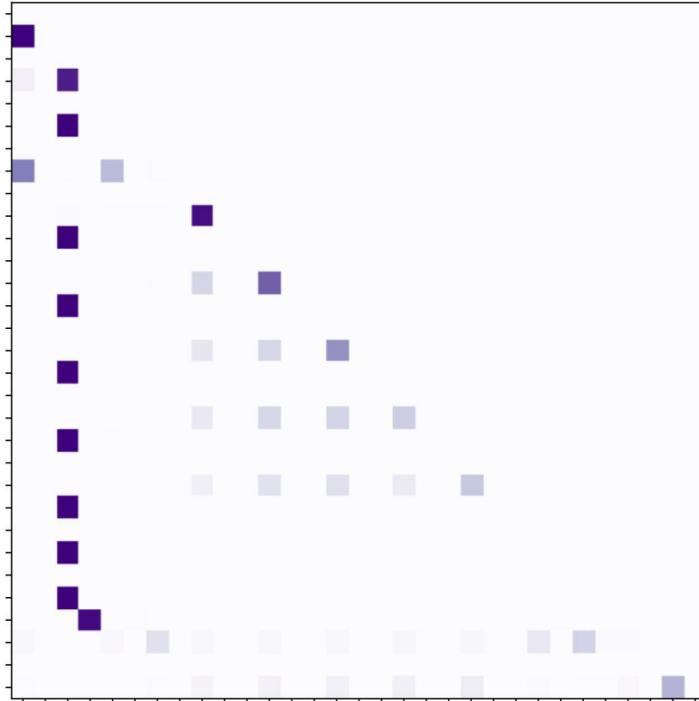
What is DIET?

- New state of the art neural network architecture for NLU
- Predicts intents and entities together
- Plug and play pretrained language models



# We found out that the Transformer Embedding Dialogue policy can untangle sub-dialogues

```
* request_hotel
 - utter_ask_details
* inform{"enddate": "May 26th"}
 - utter_ask_startdate
* inform{"startdate": "next week"}
 - utter_ask_location
* inform{"location": "rome"}
 - utter_ask_price
* chitchat
 - utter_chitchat
 - utter_ask_price
* inform{"price": "expensive"}
 - utter_ask_people
* inform{"people": "4"}
 - utter_filled_slots
 - action_search_hotel
 - utter_suggest_hotel
* affirm
 - utter_happy
```



## 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

## 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
```

## What is a Minimum Viable Assistant (MVA)?

- A basic assistant that can handle the most important **happy path** stories.
  - **Happy path:** If your assistant asks a user for some information and the user provides it, we call that a happy path.
  - **Unhappy path:** All the possible edge cases of the bot

## 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.

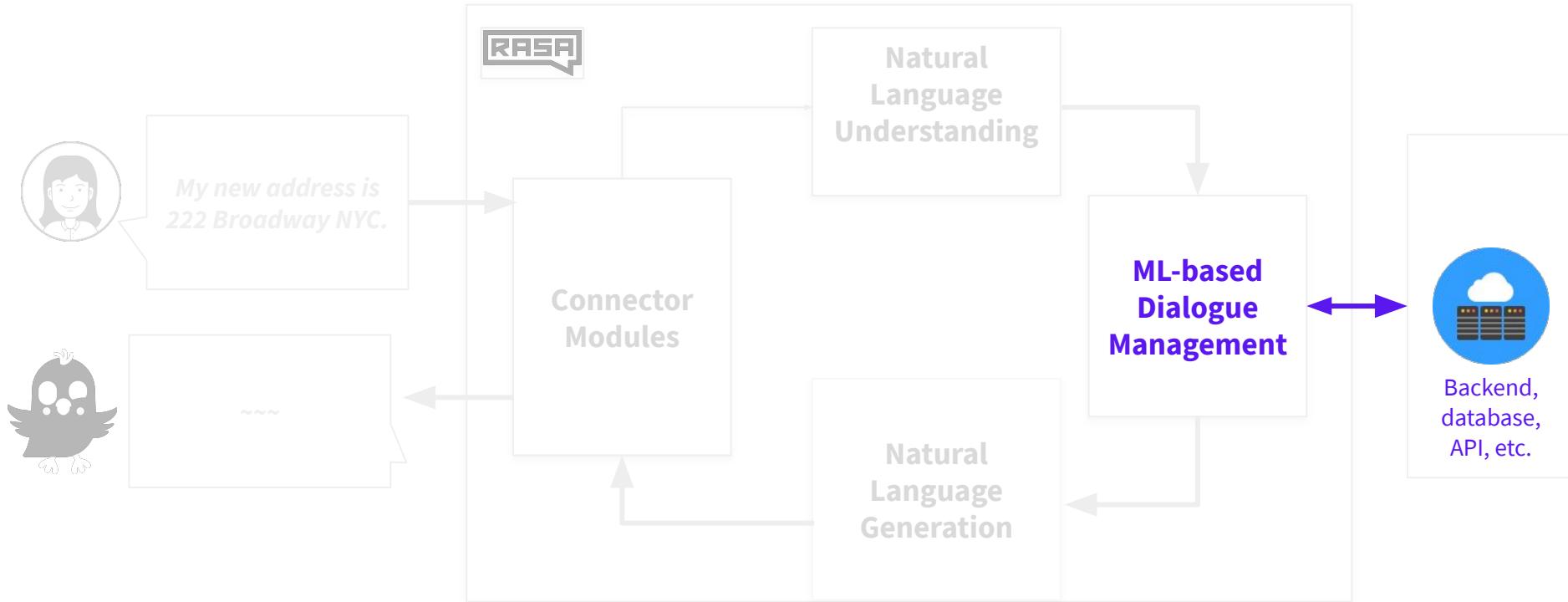
```
$ rasa test nlu -u
data/nlu.md --config
config.yml
--cross-validation
```

### Core Evaluation

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

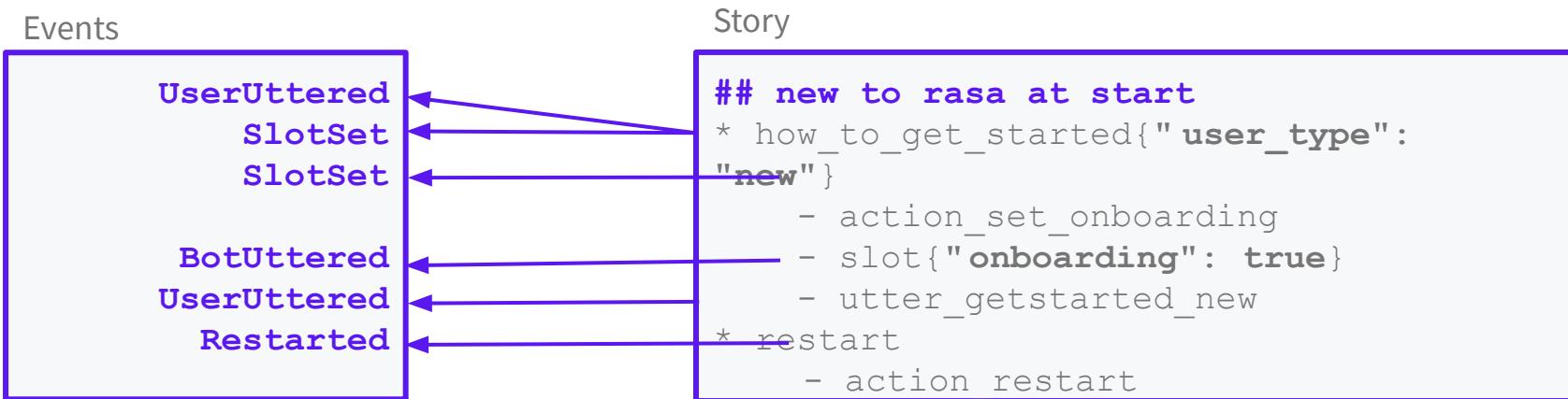
```
$ rasa test core
--stories
test_stories.md --out
results
```

## Custom Actions: Connecting to the outside world



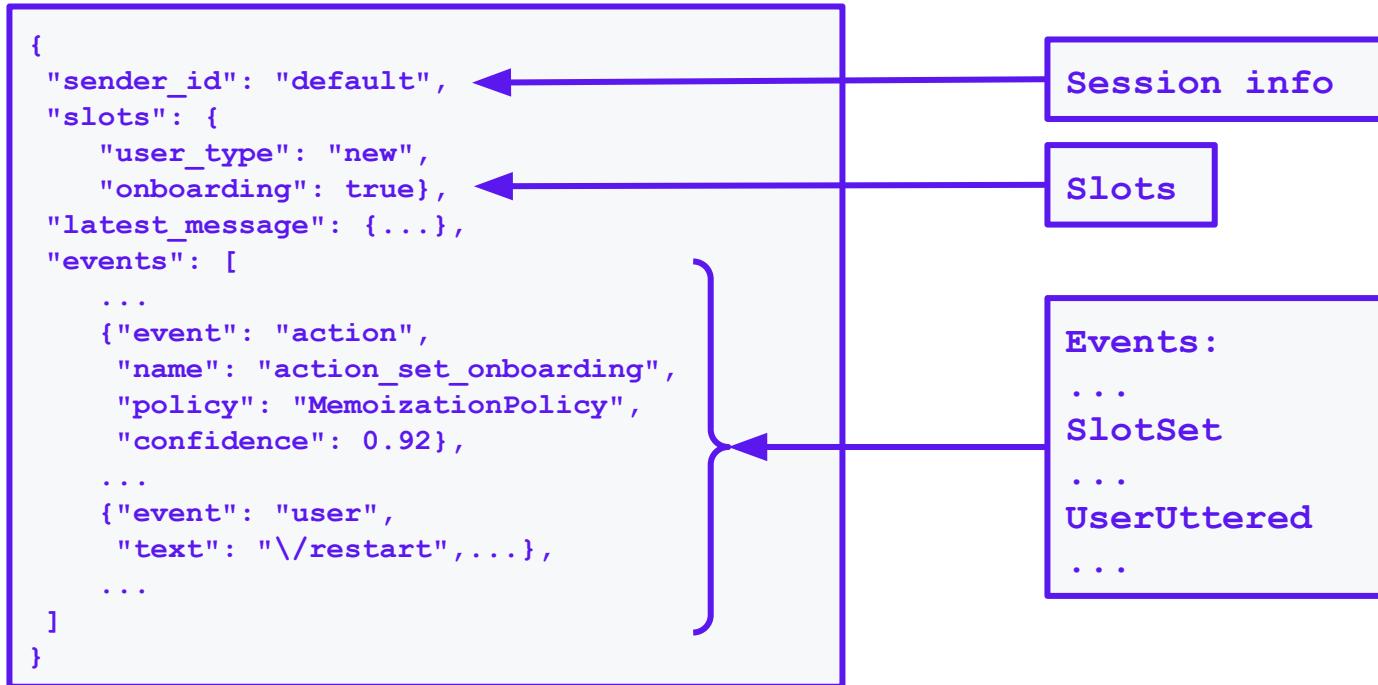
## Events

- **Internally**, all conversations are represented as a sequence of **events**.
- Some events are automatically tracked



## Tracker

- Trackers maintain the **state of a dialogue** between the assistant and the user.
- It keeps track of events, slots, session info etc.



## Slots

### Your bot's memory

- Can store:
  - user-provided info
  - info from the outside world
- Can be set by:
  - NLU (from extracted entities, or buttons)
  - Custom Actions
- Can be configured to affect or not affect the dialogue progression

## Actions

Things your bot runs in response to user input.

**Four different action types:**

- **Utterance actions:** `utter\_`
  - send a specific message to the user
  - specified in `responses` section of the domain
- **Retrieval actions:** `respond\_`
  - send a message selected by a retrieval model
- **Custom actions:** `action\_`
  - run arbitrary code and send any number of messages (or none).
  - return events
- **Default actions:**
  - built-in implementations available but can be overridden
  - E.g. action\_listen, action\_restart, action\_default\_fallback

## Custom Actions: Examples

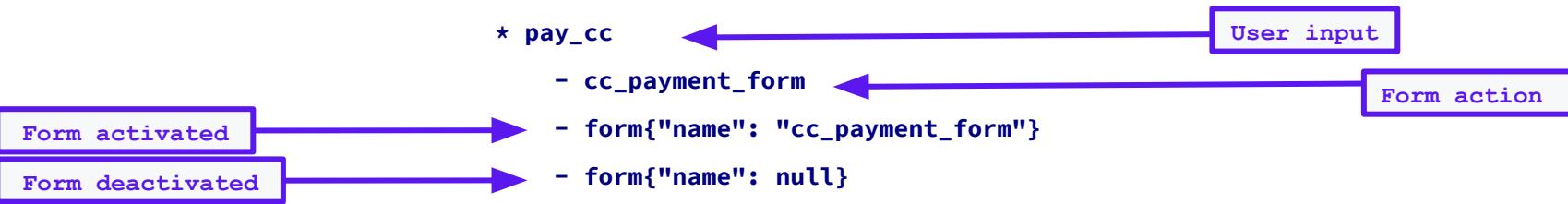
Custom actions can do whatever you want them to! e.g.

- **Send multiple messages**
- **Query a database**
- **Make an API call to another service**
- **Return events e.g.**
  - Set a slot
    - e.g. based on a database query)
  - Force a follow up action
    - force a specific action to be executed next
  - Revert a user utterance
    - I.e. remove a user utterance from the tracker

## Forms

The structure of the form

```
pay credit card happy path
* greet
 - utter_greet
* pay_cc
 - cc_payment_form
 - form{"name": "cc_payment_form"}
 - form{"name": null}
```



## Forms

### Handling unhappy paths

Users are not always cooperative - they can change their mind or interrupt the form.

- Use action `action_deactivate_form` to handle the situation where users decide not to proceed with the form:

```
chitchat
* request_restaurant
 - restaurant_form
 - form{"name": "restaurant_form"}
* stop
 - utter_ask_continue
* deny
 - action_deactivate_form
 - form{"name": null}
```

## Rasa X turns conversations into training data



Review  Annotate  Improve  


## Real Conversations > Synthetic Data

During development, training data is created by:

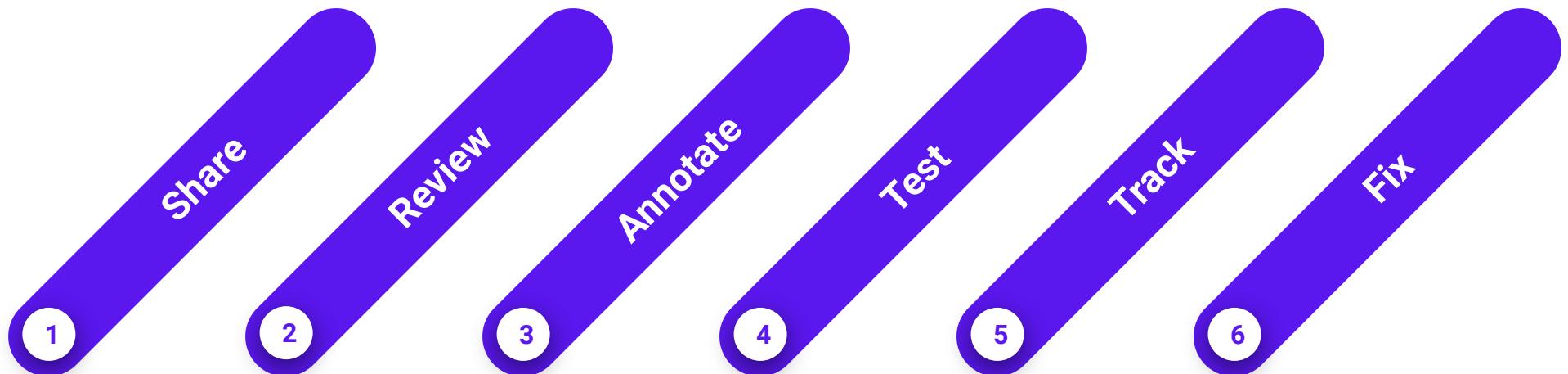
- Writing down hypothetical conversations and training examples
- Studying existing chat transcripts or knowledge bases
- Consulting subject matter experts

That's a great start, but all of those scenarios are based on human:human conversations.

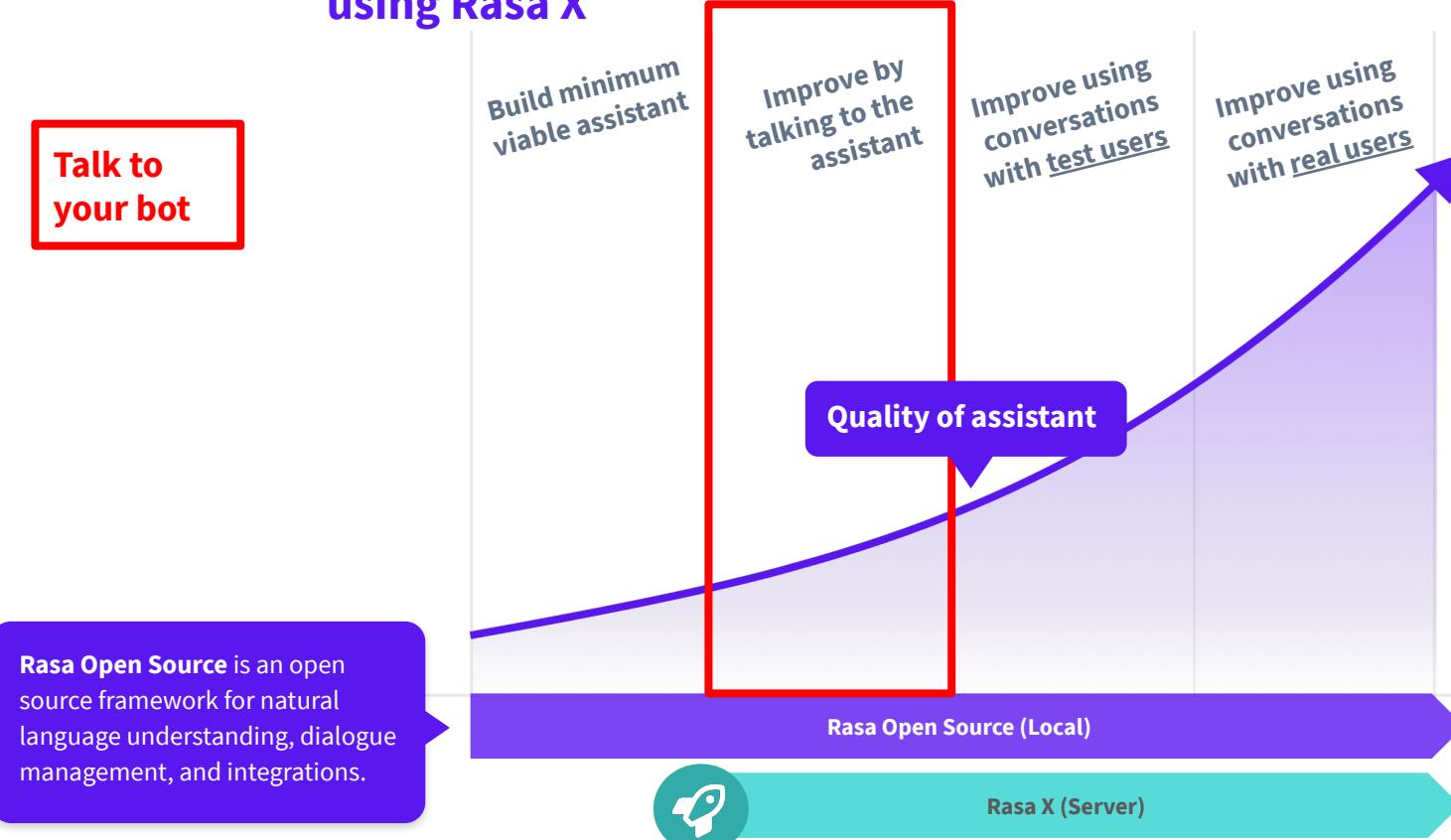
Human:bot conversations have different patterns - users say different things when they are talking to a bot.

*The best training data is generated from the assistant's  
actual conversations*

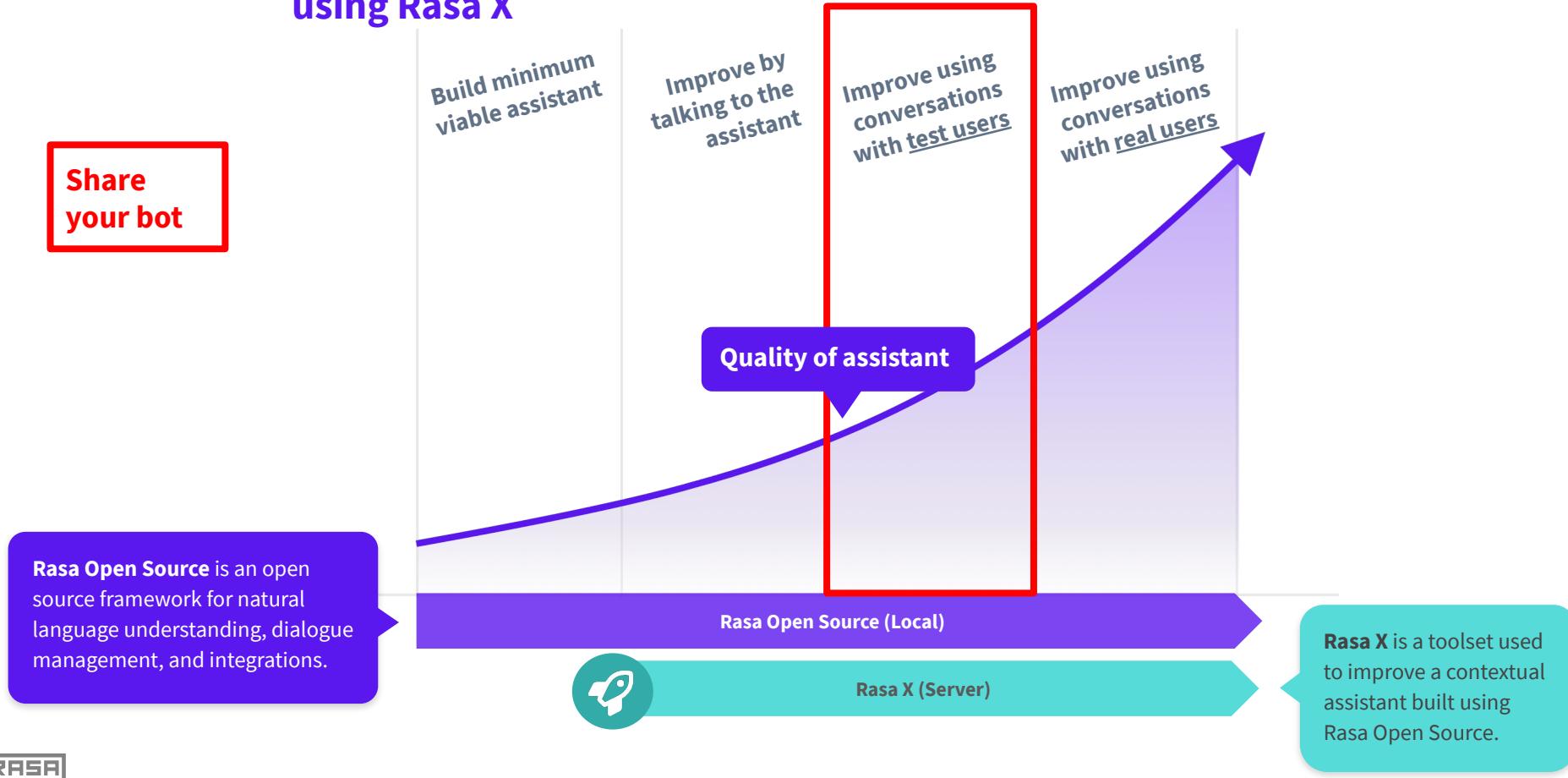
# Conversation-Driven Development



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



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

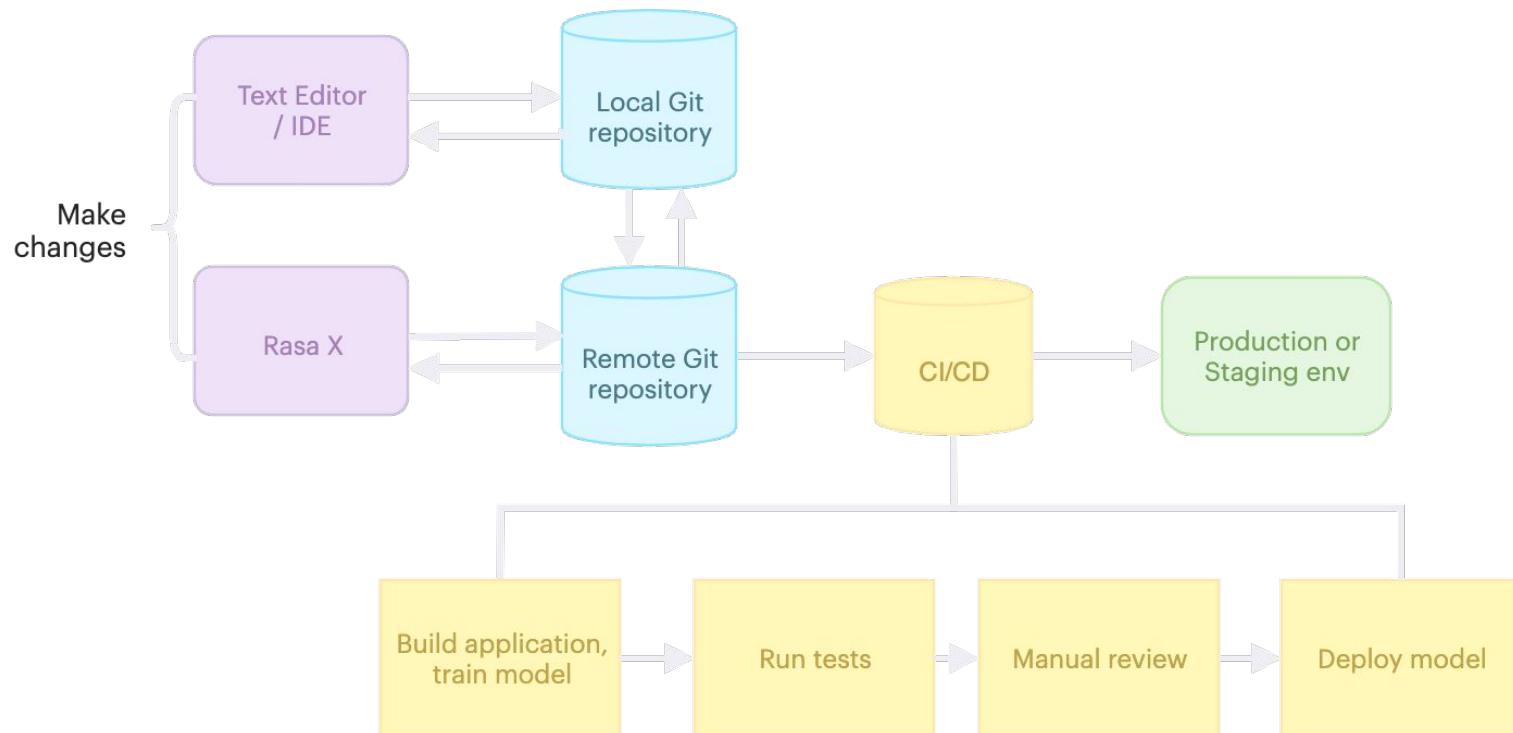


# Build a minimum viable assistant with Rasa Open Source + improve it using Rasa X

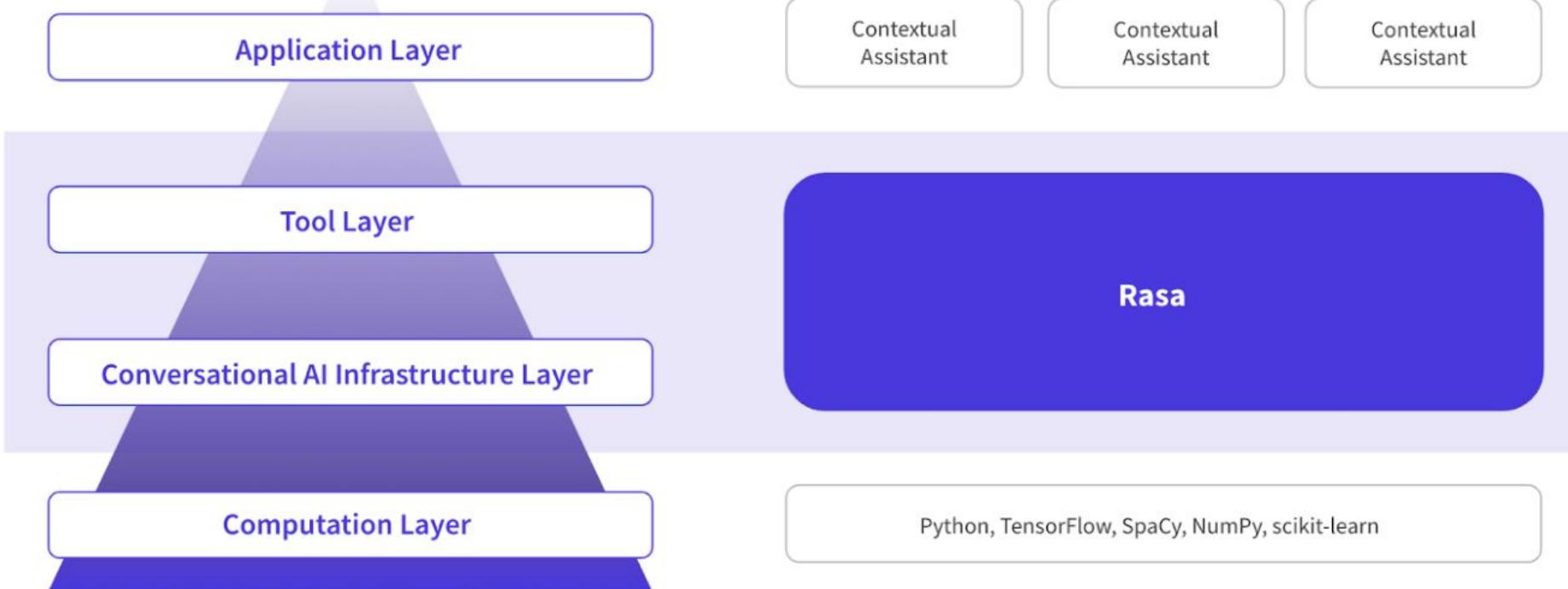


DEVELOPMENT WORKFLOW

## End-to-end workflow for managing assistant updates



## How Rasa fits into your stack



## How Rasa fits into your stack

