Voice Dev 1

CS571: Building User Interfaces

Cole Nelson & Yuhang Zhao

Today's Warmup

- Clone today's code to your machine.
  - Run the command `npm install` inside of the `starter` and `solution` folders.
- Sign up for a wit.ai account.
  - This does *not* need to be your personal Facebook account, you can sign up for a Meta account using your @wisc.edu email!

Learning Objectives

1. Be able to define important conversational terms such as agent, utterance, intent, and entity.
2. Be able to use JavaScript `async` / `await` syntax.
3. Be able to build a command-and-control chat agent!

Voice User Interfaces

VUIs are a common form of **agent-based design** as opposed to **direct manipulation**.

Conversational interfaces can be used to...

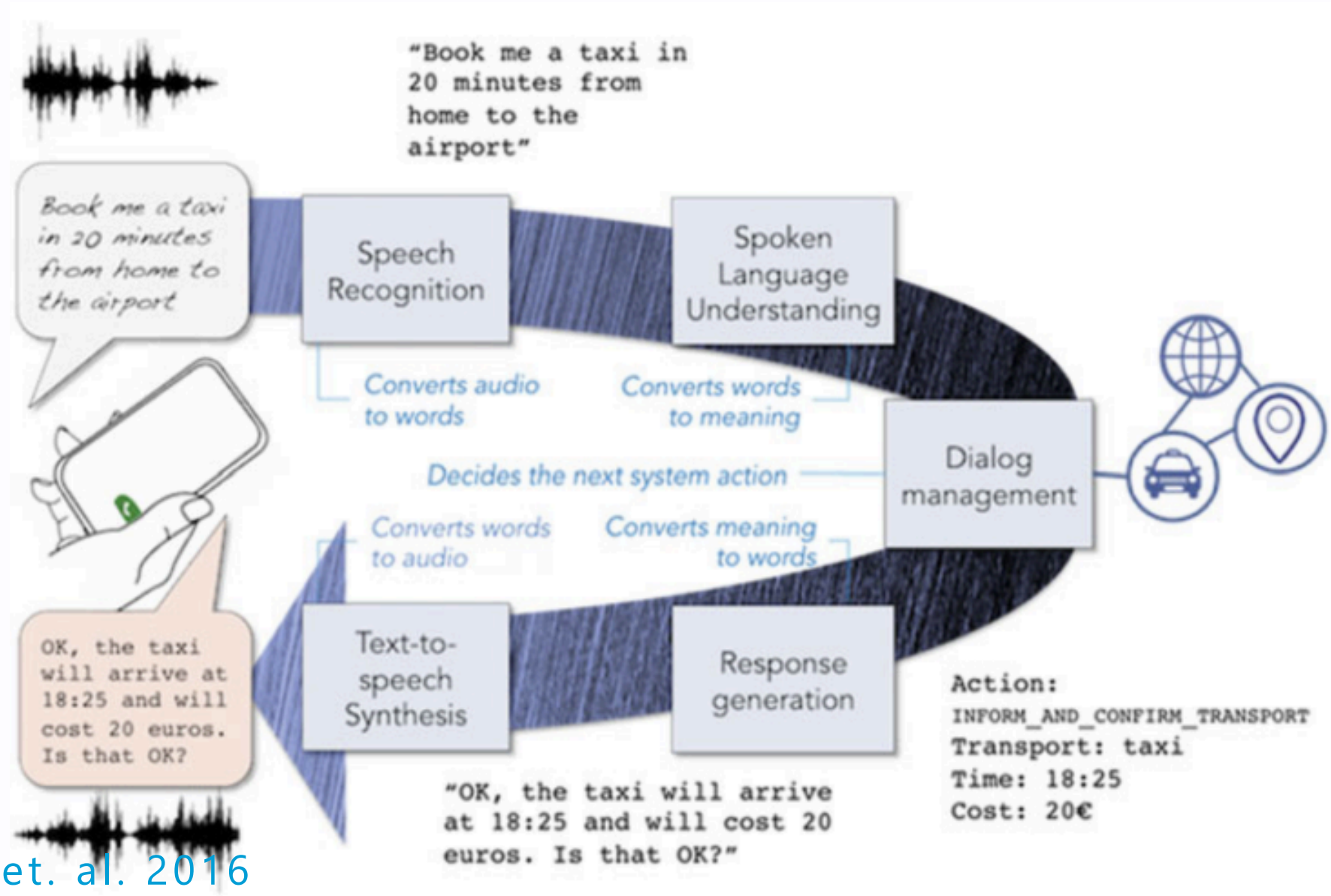
→ Address accessibility needs
→ Address context-specific problems (e.g. driving)
→ Augment the user experience

Voice User Interfaces

VUIs integrate a number of technologies and ideas...

1. Speech recognition
2. Spoken language understanding
3. Dialog management
4. Response generation
5. Text-to-speech synthesis

McTear et. al. 2016



McTear et. al. 2016

Order Domino's with Alexa!

Implementation Options

We focus on just one avenue of implementation!

- Wit.ai by Facebook
- DialogFlow by Google
- Watson Assistant by IBM
- Lex by Amazon
- Azure Bot Service by Microsoft

A Consideration: Killed by Google

Key Concepts in Wit.AI

- **Agent:** The overarching project consisting of *intents*, *utterances*, and *entities*.
- **Intents:** A higher level meaning of many *utterances*.
- **Utterances:** A string of words.

The goal of our **agent** is to extract the **intent** out of a new **utterance** and map it to a function.

Intents

Consider the following **utterances**...

- What is the weather like *tomorrow*?
- How's it looking out there *right now*?

What is the **intent** of these requests? They're both some sort of `weather_inquiry`!

These also have an **entity** of a time/date.

Intents

```
"intents": [
    {
        "confidence": 0.992232100272184,
        "id": "2117974348567211",
        "name": "tell_joke"
    }
]
```

Each intent consists of a `confidence` from 0 to 1, as well as an `id` linked to the `name` of the intent.

The 0th intent is the best match.

Your Turn!

Let's create a Wit.AI comedian that can understand...

`why_chicken` e.g. "why did the chicken cross the road"

- We will reply with "to get to the other side!"

`tell_joke` e.g. "tell me a joke"

- We will reply with jokes fetched from the Jokes API!

`async` / `await`

Previously, we used `.then` to handle the resolution of a Promise, e.g. when a `Promise` resolves, do something with the data.

`async` / `await` is syntactic sugar for handling a `Promise`. `await` is used within `async` functions to wait for a `Promise` to resolve before continuing in execution.

Equivilant Handling!

```
function printCountries() {
  fetch("https://www.example.com/countries")
  .then(resp => resp.json())
  .then(countries => {
      console.log("Countries received!", countries);
  })
}
```

```
async function printCountries() {
  const resp = await fetch("https://www.example.com/countries");
  const countries = await resp.json();
  console.log("Countries received!", countries);
}
```

`async` Functions

Every `async` function returns a `Promise`

```
// Returns a Promise<undefined>
async function printCountries() {
  const resp = await fetch("https://www.example.com/countries");
  const countries = await resp.json();
  console.log("Countries received!", countries);
}
```

`async` Functions

Every `async` function returns a `Promise`

```
// Returns a Promise<string[]>
async function printAndReturnCountries() {
  const resp = await fetch("https://www.example.com/countries");
  const countries = await resp.json();
  console.log("Countries received!", countries);
  return countries;
}
```

`async` Functions

We can then *consume* this promise in another function!

```
async function getCountries() {
  const countries = await printAndReturnCountries();
  console.log(`There are ${countries.length} countries!`);
}
```

```
function getCountries() {
  printAndReturnCountries().then(countries => {
    console.log(`There are ${countries.length} countries!`);
  })
}
```

`async` Functions

Could we do the following?

```
function getCountries() {
  const countries = await printAndReturnCountries();
  console.log(`There are ${countries.length} countries!`);
}
```

No! `await` is only allowed within `async` functions and at the top-level of our code.

`async` / `await`

- equivalent way to handle asynchronous behavior
- `await` must be inside of an `async` function or at the top-level of the program
- `await` waits for right-hand-side to complete
- a synchronous function may spawn `async` behavior (e.g. a function triggering a `fetch`)
- an `async` function always happens asynchronously, always returning a `Promise`

Your Turn!

Adapt your code to use `async` / `await`.

Entities

We can get all kinds of jokes...

https://v2.jokeapi.dev/joke/any?safe-mode

... or we can get specific jokes!

https://v2.jokeapi.dev/joke/spooky?safe-mode

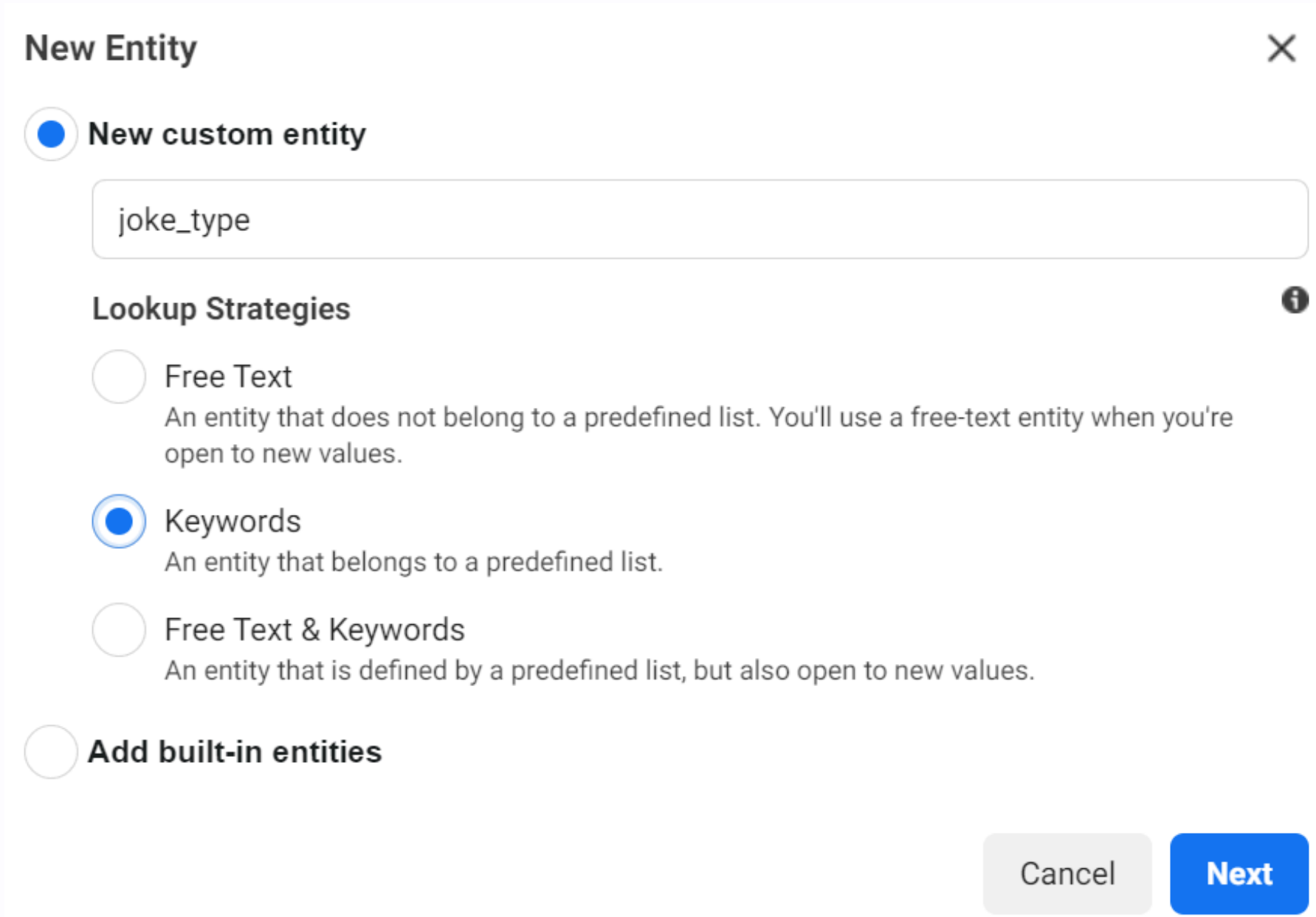
How can we handle this?

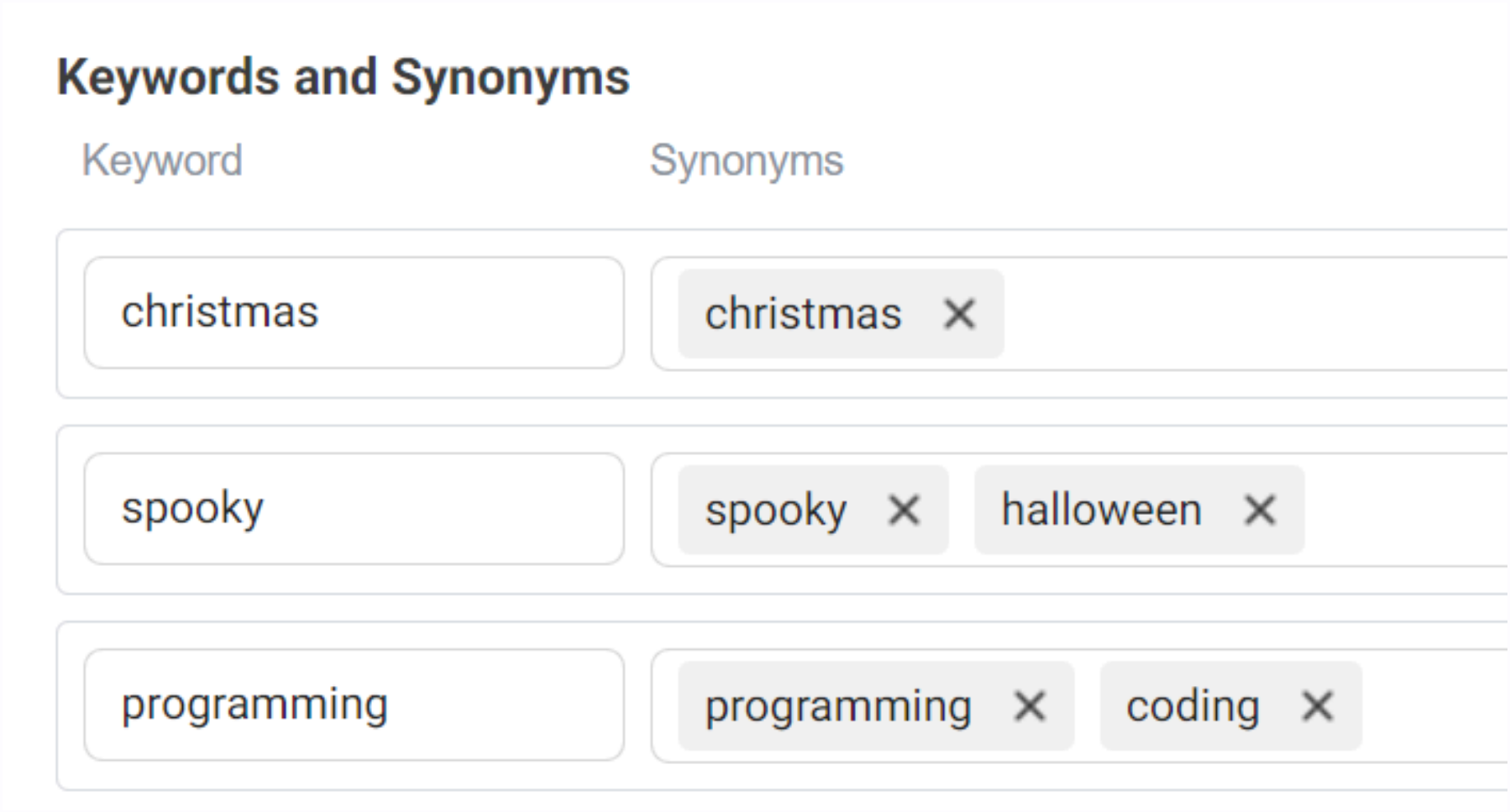
Entities

We could create the following intents...

- `tell_pun_joke`
- `tell_spooky_joke`
- `tell_programming_joke`
- ...

... or make an entity parameter, e.g. `joke_type`!


New Entity
New custom entity
joke_type
Lookup Strategies
Free Text
An entity that does not belong to a predefined list. You'll use a free-text entity when you're open to new values.
Keywords
An entity that belongs to a predefined list.
Free Text & Keywords
An entity that is defined by a predefined list, but also open to new values.
Add built-in entities
Cancel
Next


Keywords and Synonyms
Keyword
Synonyms
christmas
christmas
spooky
spooky
halloween
programming
programming
coding

```
{
    "entities": {
        "joke_type:joke_type": [
            {
                "body": "halloween",
                "confidence": 1,
                "end": 19,
                "entities": {},
                "id": "948991563175475",
                "name": "joke_type",
                "role": "joke_type",
                "start": 10,
                "type": "value",
                "value": "spooky"
            }
        ]
    },
    "intents": [
        {
            "confidence": 0.992232100272184,
            "id": "2117974348567211",
            "name": "tell_joke"
        }
    ],
    "text": "tell me a halloween joke",
    "traits": {}
}
```

Intents & Entities

The JSON response body consists of...

- `text` - exactly what the user said
- `intents` - a *list* of likely matches
- `entities` - an *object* of likely matches
- `traits` - 😊😔

Read the Wit.AI Docs

Intents

```
"intents": [
    {
        "confidence": 0.992232100272184,
        "id": "2117974348567211",
        "name": "tell_joke"
    }
]
```

Each intent consists of a `confidence` from 0 to 1, as well as an `id` linked to the `name` of the intent.

The 0th intent is the best match.

Entities

```
"entities": {
    "joke_type:joke_type": [
        {
            "body": "halloween",
            ...
            "value": "spooky"
        }
    ]
}
```

Entities map a specific type to a list of `body` (what was actually written) and `value` (what it was resolved to)

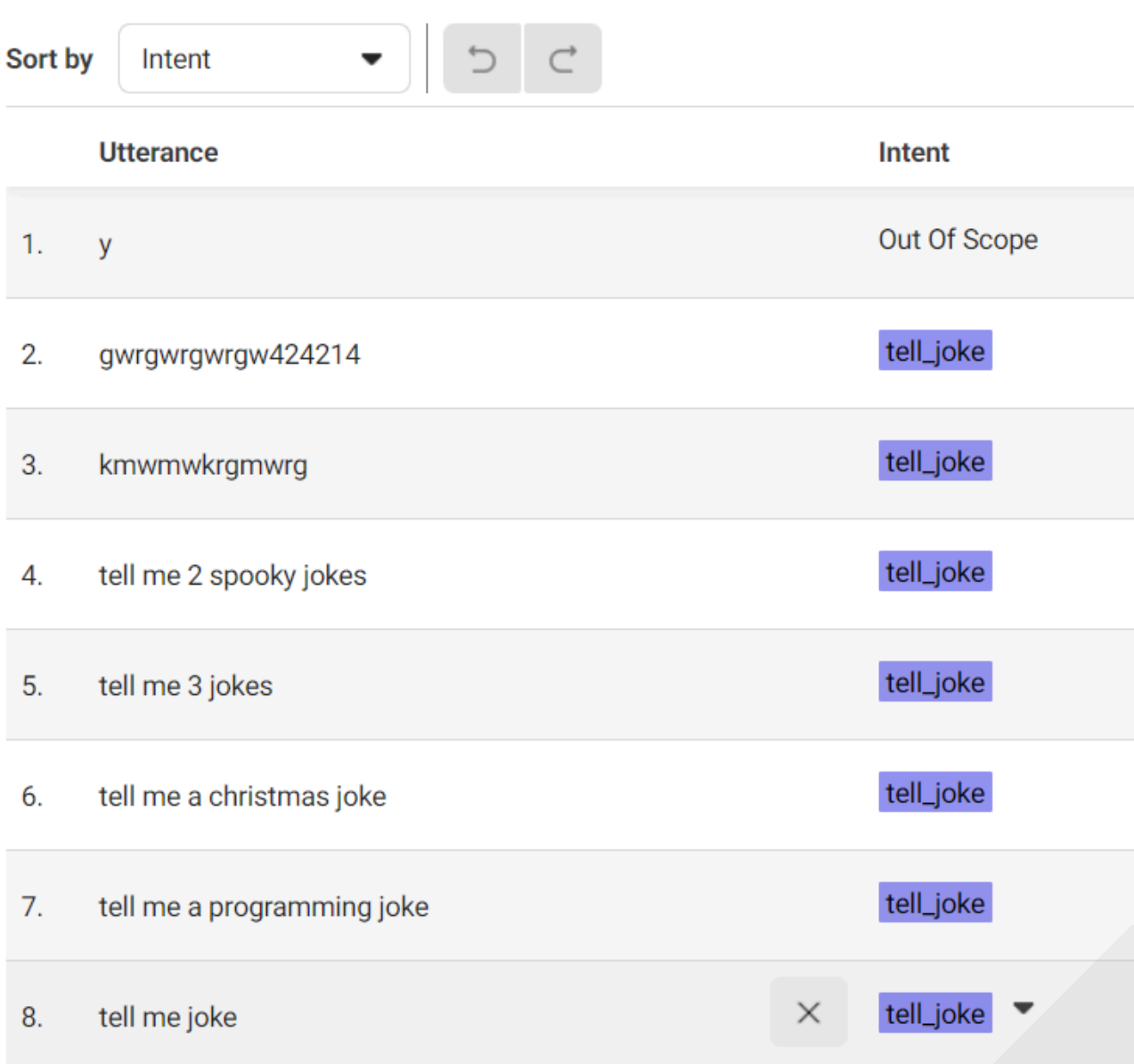
Your Turn!

Let's build a better comedian that takes joke requests.

Reminder: AI!

There are no guarantees. Intent matching is still emerging technology.

Use as many utterances as you can!



Questions?