# Voice Dev 1

## CS571: Building User Interfaces

Cole Nelson

## 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!

## Final Exam

- Friday, May 10th at 5:05 pm.
  - Fill out the conflict form by May 1st.
  - Graduating seniors may take the alternate exam on Thursday May 9th at 5:05 pm by completing the form above, just use some proof of upcoming graduation!
- SEC001 in Chamberlin 141, SEC002 in Chem S429.
  - Go by section in enroll.wisc.edu
  - Alternate exam location is still TBD.

## Final Exam

- 90 minutes for 50 MC questions.
  - 25 design and 25 implementation questions
- A single double-sided notesheet.
- Cumulative, with a heavier emphasis on content from the second-half of the semester.

## Other Announcements

- **All work** must be submitted by May 3rd at 11:59 pm.
  - HW11 up to 4 days late.
  - HW12 no late days, shorter submission period.
- 1 bonus point will be available...
  - +0.5 pts for completing the HCI Workshop Quiz.
  - +0.5 pts for completing the security CTF.
  - **There is no rounding of grades.**

## 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...

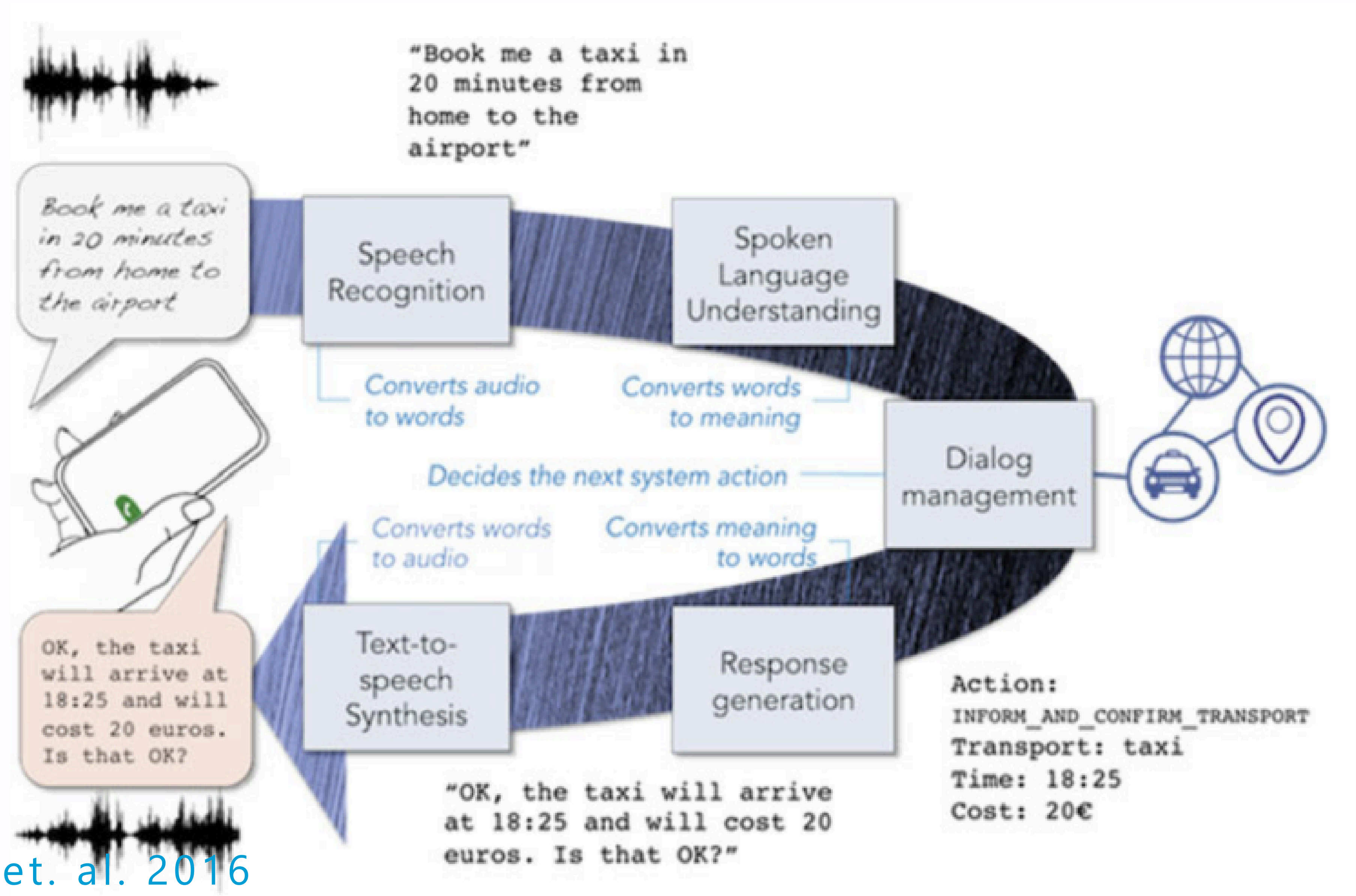
→ Address accessibility needs
→ Address context-specific problems (e.g. driving)
→ Augment the user experience

## 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


"Book me a taxi in 20 minutes from home to the airport"
Book me a taxi in 20 minutes from home to the airport
Speech Recognition
Spoken Language Understanding
Converts audio to words
Converts words to meaning
Decides the next system action
Dialog management
Converts words to audio
Converts meaning to words
OK, the taxi will arrive at 18:25 and will cost 20 euros. Is that OK?
Text-to-speech Synthesis
Response generation
Action: INFORM_AND_CONFIRM_TRANSPORT
Transport: taxi
Time: 18:25
Cost: 20€
"OK, the taxi will arrive at 18:25 and will cost 20 euros. Is that OK?"

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.
- **Entities:** Special attributes of an *intent*.

The goal of our **agent** is to extract the **intent** and any **entities** 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.

## 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!

## 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

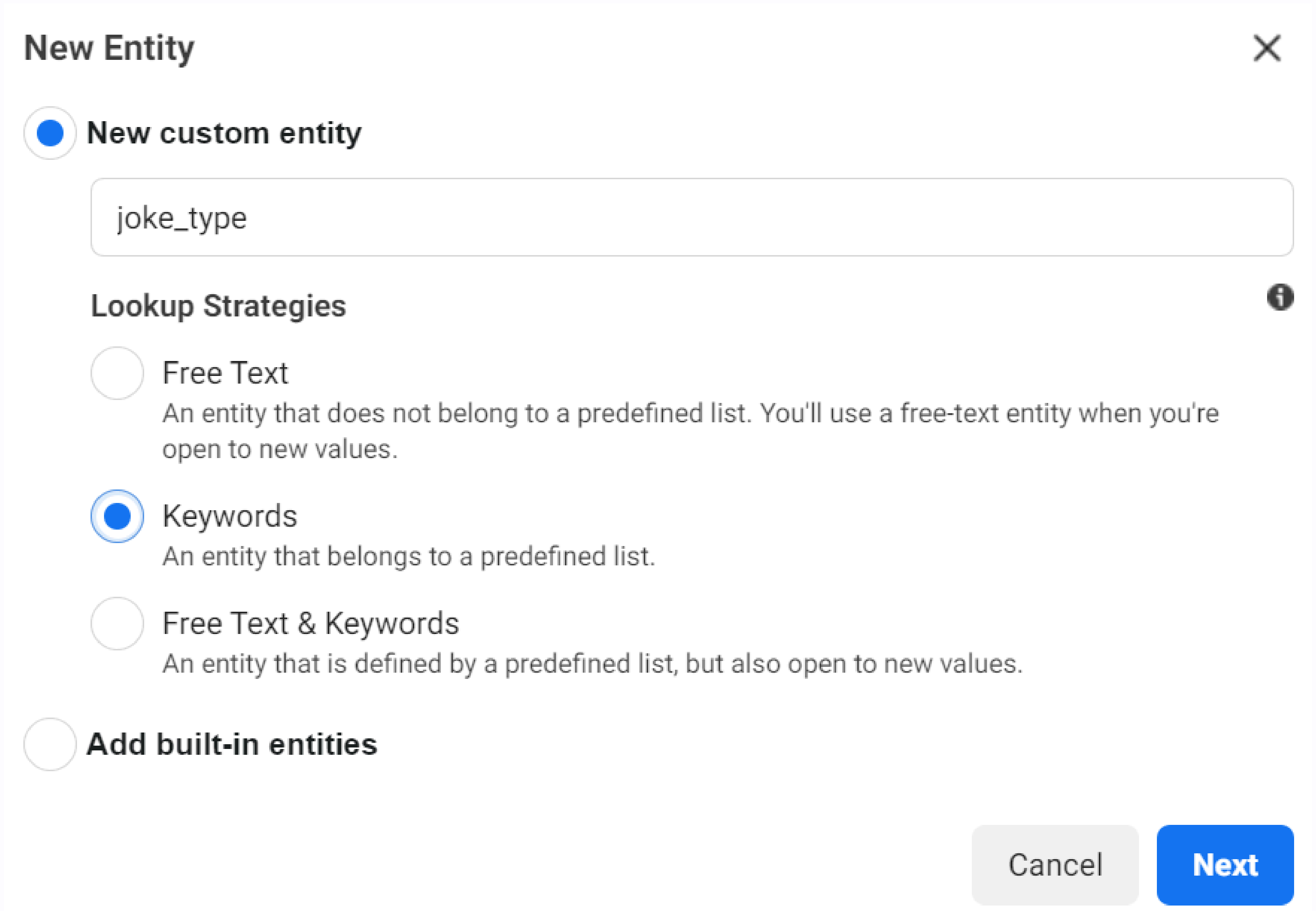
How can we handle this?

## 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

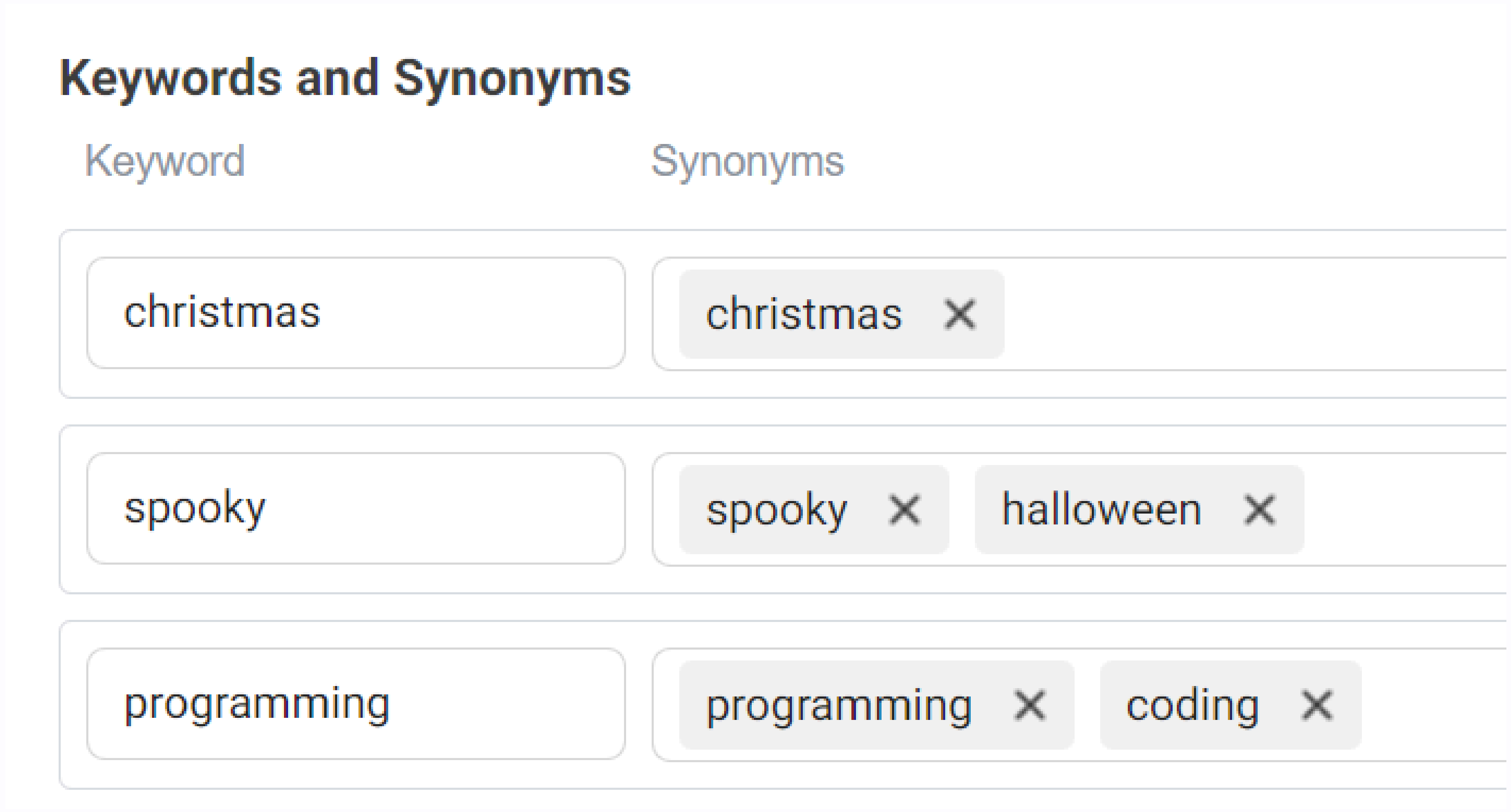
Lookup Strategies

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



## 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` - maybe next time? 😊😔

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 an agent to handle some jokes.

## A Review of Async

**Definition:** not happening or done at the same time.

We `promise` something will happen in the future.

If we fulfill our promise, `then` we do something.

Otherwise, we `catch` and handle the error.

## `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
- every `async` function returns a `Promise`
- a synchronous function may spawn `async` behavior
- an `async` function always happens asynchronously

## Equivilant Handling!

```
function getLogos() {
  fetch("https://www.example.com/logos?amount=20")
  .then(res => res.json())
  .then((newLogos) => {
      setLogos(newLogos);
  })
}
```

```
async function getLogos() {
  const resp = await fetch("https://www.example.com/logos?amount=20");
  const newLogos = await resp.json();
  setLogos(newLogos);
}
```

## Equivilant Handling - Errors

```
function getLogos() {
  fetch("https://www.example.com/logos?amount=20")
  .then(res => res.json())
  .then((newLogos) => {
      setLogos(newLogos);
  })
  .catch(e => alert("Something went wrong!"))
}
```

## Equivilant Handling - Errors

```
async function getLogos() {
  try {
    const resp = await fetch("https://www.example.com/logos?amount=20");
    const newLogos = await resp.json();
    setLogos(newLogos);
  } catch (e) {
    alert("Something went wrong!")
  }
}
```

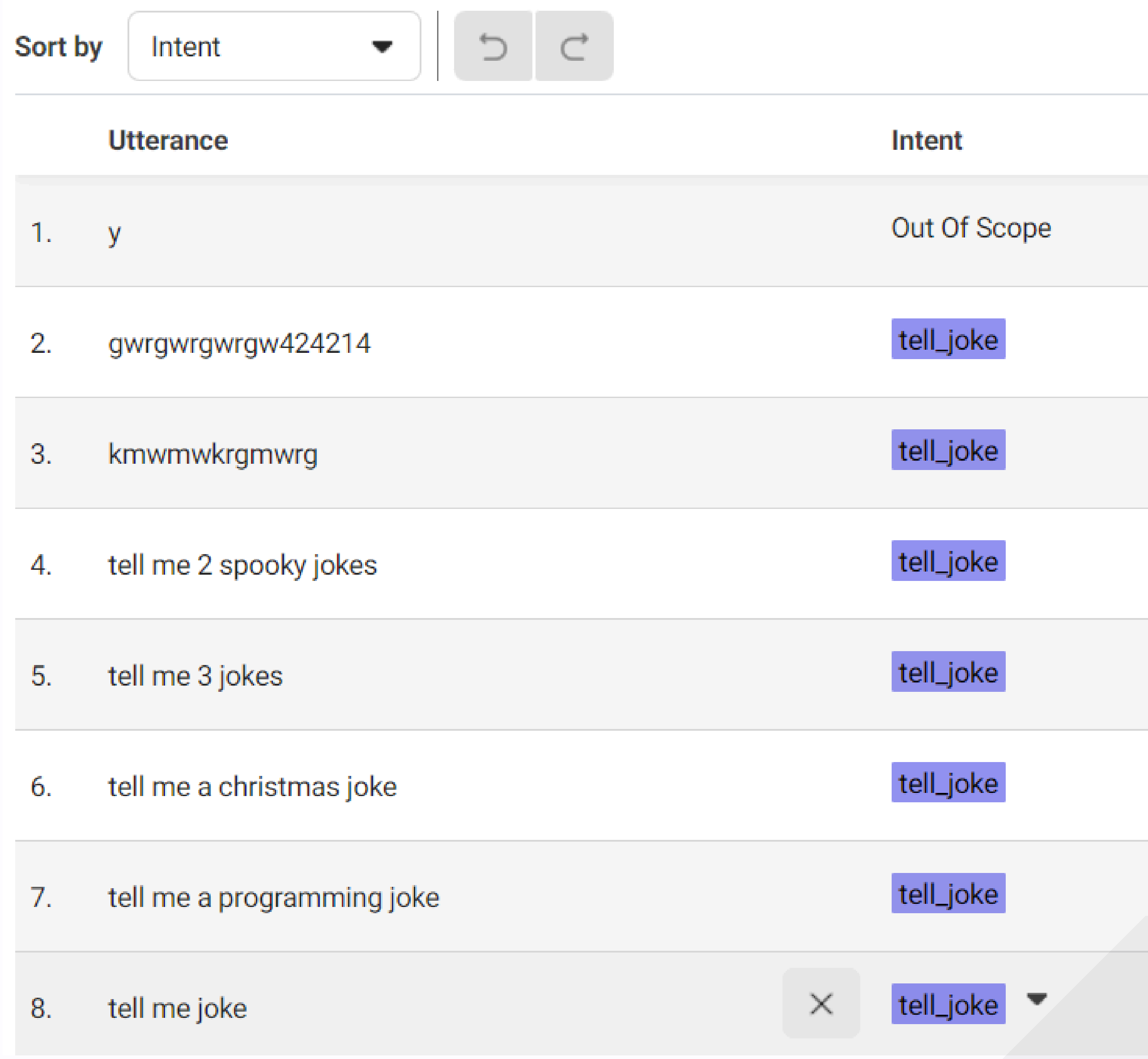
## Your Turn!

Let's build a better agent using `async` / `await` ...

# Reminder: AI!

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

Use as many utterances as you can!



Sort by Intent

| | Utterance | Intent |
|---|---|---|
| 1. | y | Out Of Scope |
| 2. | gwrgwrgwrgw424214 | tell_joke |
| 3. | kmwmwkrgmwrg | tell_joke |
| 4. | tell me 2 spooky jokes | tell_joke |
| 5. | tell me 3 jokes | tell_joke |
| 6. | tell me a christmas joke | tell_joke |
| 7. | tell me a programming joke | tell_joke |
| 8. | tell me joke | tell_joke |

## Questions?