4 – Bringing AI into the Code

Reorientation

Where are we and where are we heading

Coding CAI Deep Dive So Far

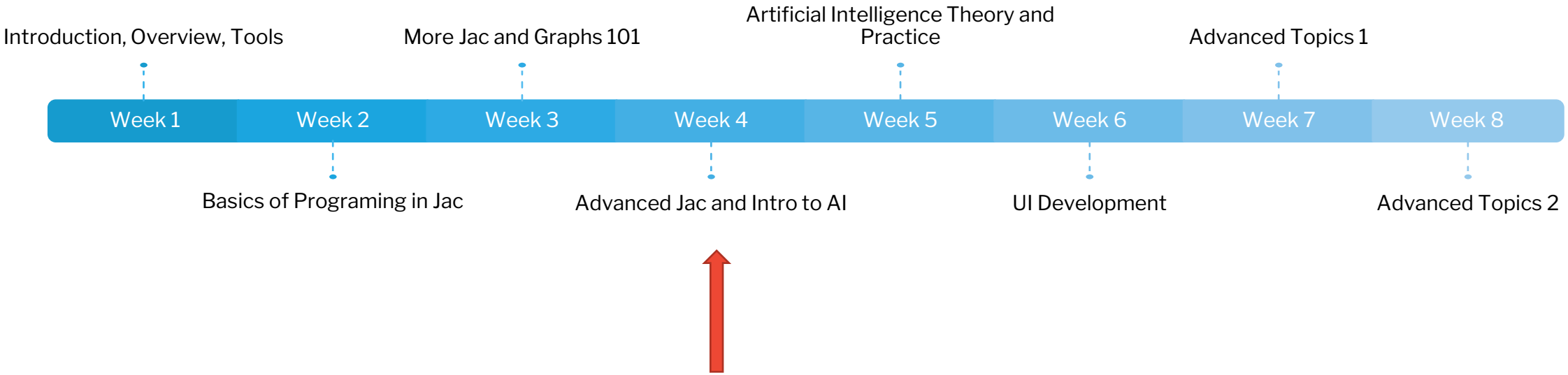
Toward our End to End bleeding edge conversational AI

Coding with AI

QA Scoring Like a ZSB

Getting Reoriented

An 8-week Journey To Master Building AI Products



Showcase and Prizes!!

- Showcase Coming
  - Big Event at the End
  - Everyone is invited
- Real Prizes
  - Best overall Coding! - $500 USD
  - Best coding Effort! - $250 USD
  - Special gift for All Coders (that committed code) - ?? Surprise
- It’s Time to Commit (to the Github repos ;-P)

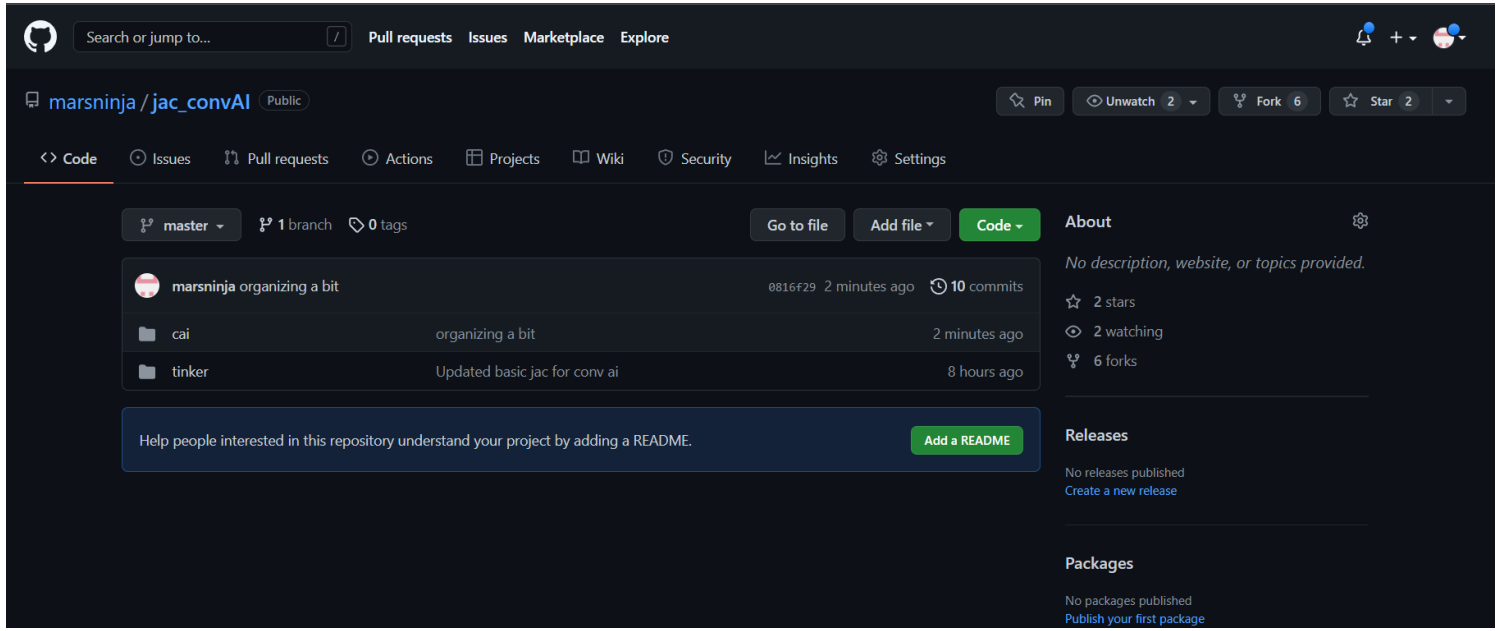
Visualizing the Journey

- Step 1: Understand CAI + Tinker
- Step 2: Integrate 3 AI Models
  - Question / Answering
  - Intent Classification
  - Named Entity Recognition
- Step 3: Plug in UI
- Step 4: Win!!

Coding and Graphs

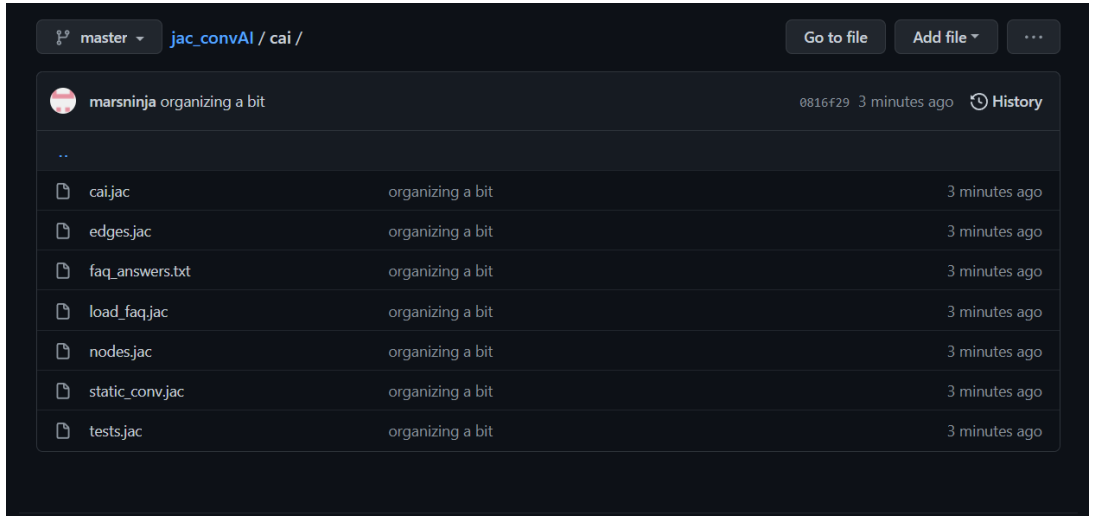
CAI

- A Conversational AI Example
  - Git clone https://github.com/marsninja/jac_convAI.git



How the project is organized

- Organized into 7 files
  - cai.jac
  - nodes.jac
  - edges.jac
  - static_conv.jac
  - load_faq.jac
  - test.jac
  - faq_answers.txt



Imports

- Imports enable multifile organization to code
  - Specify what you want to import
  - And you can code as if its available

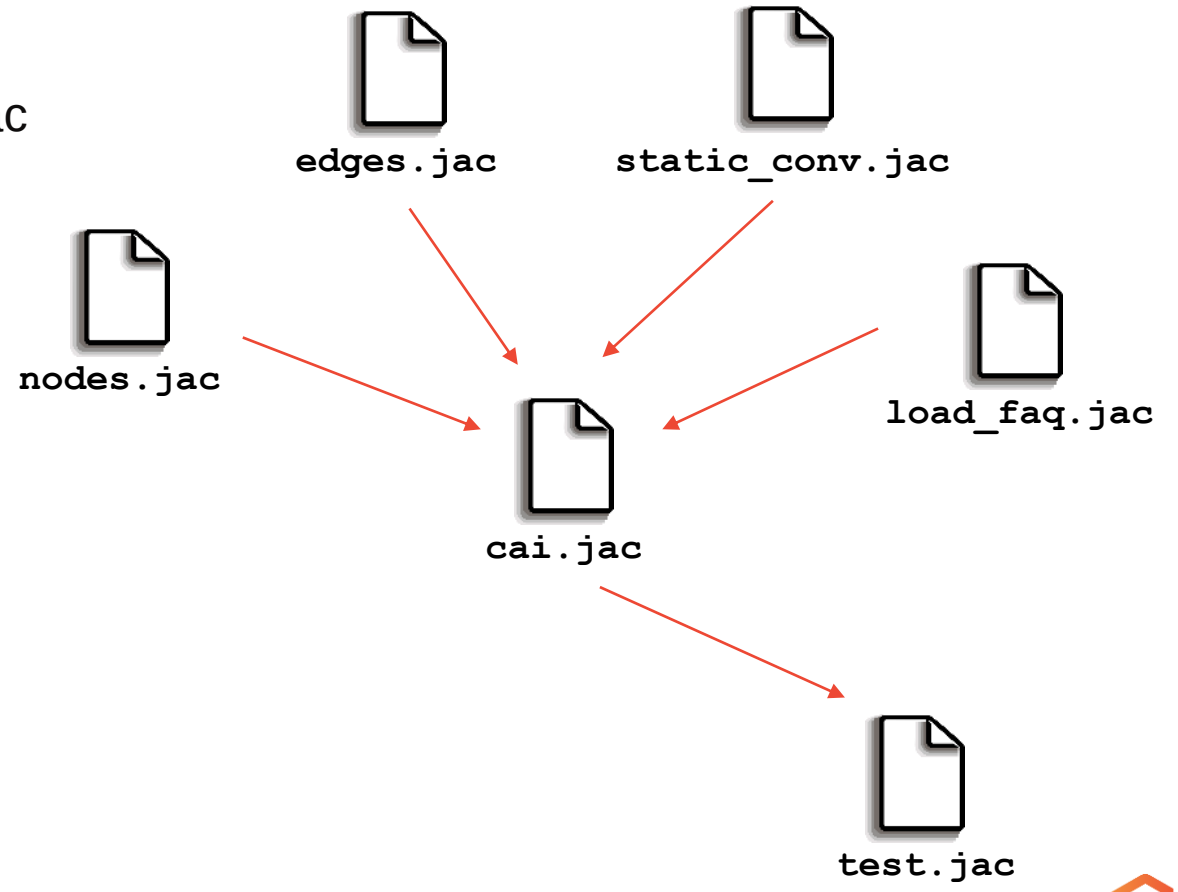
```
import {node::{state, hop_state}} with "./nodes.jac";
import {edge::{trans_ner, trans_intent, trans_qa}} with "./edges.jac";
import {graph::basic_gph} with "./static_conv.jac";
import {graph::faq_gph} with "./load_faq.jac";

walker init {
    root {
        spawn here --> graph::basic_gph;
        spawn -->[0] -[trans_intent(intent="about chat bots")]-> graph::faq_gph;
    }
    with exit {
        spawn -->[0] walker::talker;
    }
}

walker talker {
    has utterance="";
    has use_cmd = true, path = [];
    if(use_cmd and here.details['name'] != 'hop_state'):
        utterance = std.input("> ");
    take -->;
}
```

Import Structure of CAI Jac Files

- Cai.jac is the central hub
  - Other files have various assets that cai.jac depends on
  - Call jac run on cai.jac and everything works
- Test.jac imports from CAI
  - Only to test functionality as you code
  - Call jac test on test.jac to run tests



cai.jac

- Init walker
  - Creates static graph for conversational flow
  - Pulls in FAQs to add to graph
  - Runs talker walker on exit
- Talker walker
  - Grabs input from standard in
  - Walks entire graph (for now)

```
import {node::{state, hop_state}} with "./nodes.jac";
import {edge::{trans_ner, trans_intent, trans_qa}} with "./edges.jac";
import {graph::basic_gph} with "./static_conv.jac";
import {graph::faq_gph} with "./load_faq.jac";

walker init {
    root {
        spawn here --> graph::basic_gph;
        spawn -->[0] -[trans_intent(intent="about chat bots")]->
graph::faq_gph;
    }
    with exit {
        spawn -->[0] walker::talker;
    }
}

walker talker {
    has utterance="";
    has use_cmd = true, path = [];
    if(use_cmd and here.details['name'] != 'hop_state'):
        utterance = std.input("> ");
    take -->;
}
```

nodes.jac

- State node
  - Has placeholder functionality
  - Can record user utterances
  - Speak is triggered by any walker
  - Listen only happens for talker walkers
  - Special test ability for demo purposes
- Hop_state node
  - Connects conversational subgraphs

```
node state {
    has name = rand.word();
    has response="I'm a silly bot.";
    has user_utter;

    can speak with entry {
        std.out(response + " I'm current on "+name+" node");
    }

    can listen with talker exit {
        user_utter = visitor.utterance;
        visitor.path.l::append(&here);
        std.out("I heard "+user_utter+".");
    }

    can test_path with get_states entry {
        visitor.path.l::append(&here);
    }
}

node hop_state {
    has name;
    can log with exit {
        std.log("A walker is walking right over me.");
    }
}
```

edges.jac

- Three types of edges with data in them that we'll use to trigger AI functionality
  - NER
  - Intent
  - QA

```
edge trans_ner { has entities; }
edge trans_intent { has intent; }
edge trans_qa { has embed; }
```

static_conv.jac

- A statically connected graph
- Used for quick prototyping of our conversational flow
- Anchor root node is whats returned to connecting Edge

```
import {edge::{trans_ner, trans_intent, trans_qa}} with "./edges.jac";
import {node::{state, hop_state}} with "./nodes.jac";

graph basic_gph {
    has anchor conv_root;
    spawn {
        conv_root = spawn node::state(name="Conv Root");

        appt = spawn conv_root -[trans_intent(intent="appointment")]->
            node::hop_state(name="Appointments");

        spawn appt -[trans_intent(intent="create")]->
            node::state(name="Create an appoitnment");
        spawn appt -[trans_intent(intent="cancel")]->
            node::state(name="Cancel an appoitnment");
        spawn appt -[trans_intent(intent="reschedule")]->
            node::state(name="Reschedule an appoitnment");

        service = spawn conv_root -[trans_intent(intent="service info")]->
            node::hop_state(name="Services");

        spawn service -[trans_intent(intent="manicures")]->
            node::state(name="About manicures");
        spawn service -[trans_intent(intent="haircuts")]->
            node::state(name="About haircuts");
        spawn service -[trans_intent(intent="makeup")]->
            node::state(name="About makeup");
    }

}
```

load_faq.jac

- This is a similar static graph builder
  - Uses file I/O to programmatically build out edges

```
import {edge::{trans_ner, trans_intent, trans_qa}}
with "./edges.jac";
import {node::{state, hop_state}} with "./nodes.jac";

graph faq_gph {
    has anchor faq_root;
    spawn {
        faq_root = spawn node::state(name="Faq Root");

        answers =
file.load_str('./faq_answers.txt').str::split('&&&');

        for i in answers:
            spawn faq_root -[trans_qa]->
node::state(response=i);
    }

}
```

test.jac

- Example test capabilities
- Tests are simple
  - Run existing walkers on static graphs
  - Assert the functionality you expect

```
import {*} with "./cai.jac";

walker get_states {
    has anchor path = [];
    take -->;
}

test "Travesal touches all nodes"
with graph::basic_gph by
walker::get_states {
    std.out(path.length);
    assert(path.length==7);
}
```

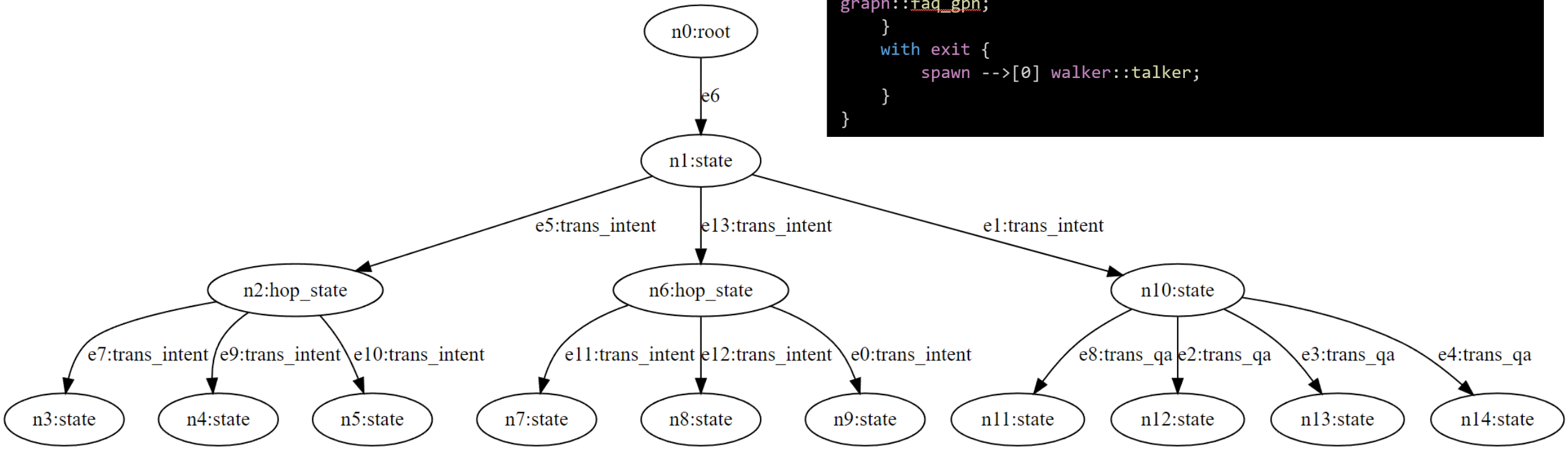
faq_answers.txt

- Flat file with answers for FAQ engine
- &&& used as delimiter (separator) by loader

```
A chatbot is an artificial intelligence (AI) based computer program that can interact with a human either via voice or text through messaging applications, websites, mobile apps or through the telephone.
&&&
Conversational chatbots have been around for decades now. In the past, there have been many unsuccessful attempts to build a chatbot that successfully mimics human conversation. However, not thats solved with the creation of me!
&&&
During the chatbot design process, it is important to keep your user in mind as it will help you define the right chatbot features, functionality and build human-like interactions.
&&&
In order for a chatbot to function properly, it is crucial for the program to access your knowledge base, website, internal databases, existing documents, or other sources of information.
```

The Conversational Graph (atm)

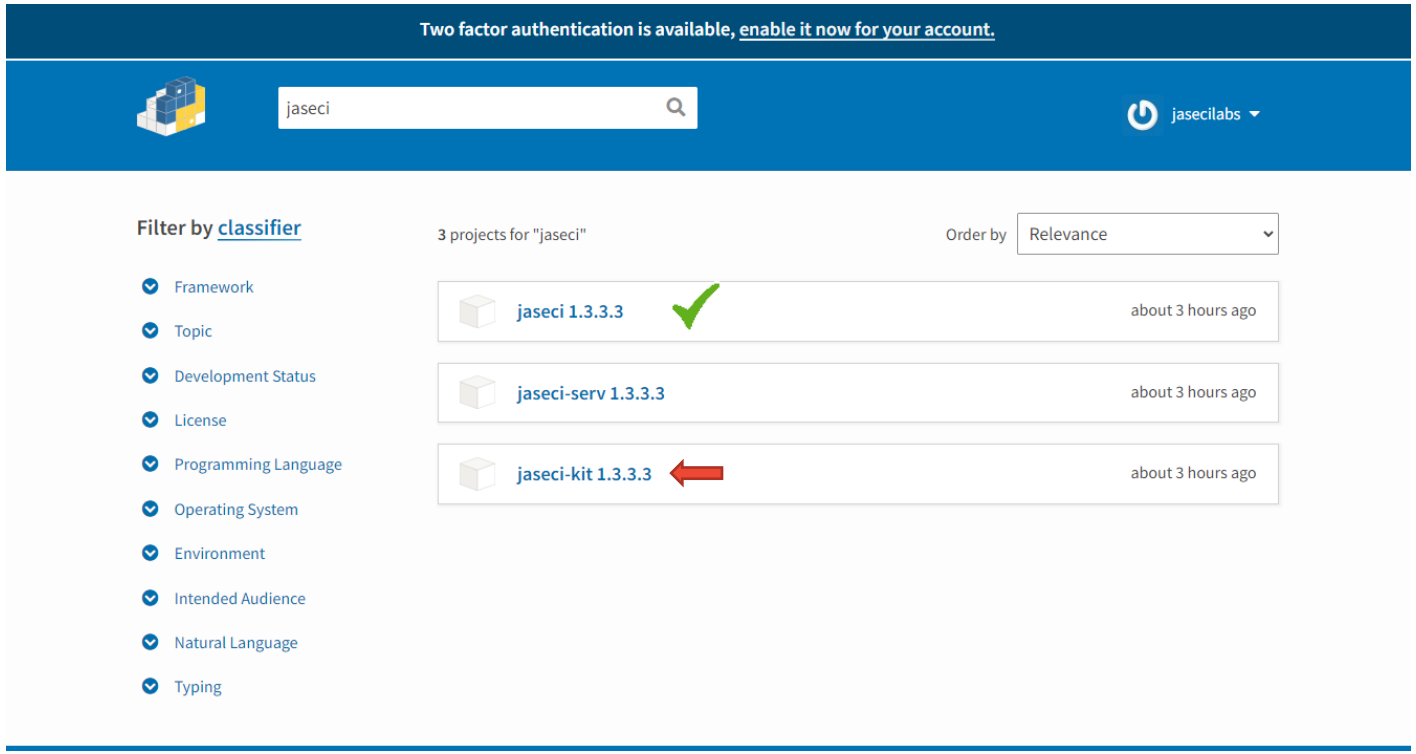
```
walker init {
    root {
        spawn here --> graph::basic_gph;
        spawn -->[0] -[trans_intent(intent="about chat bots")]->
graph::faq_gph;
    }
    with exit {
        spawn -->[0] walker::talker;
    }
}
```



Bringing AI In!

New Pipy Package to install

- The Jaseci Kit
  - Filled with the absolute state of the art AI Models
  - We'll be using USE QA now
- pip install jaseci-kit



Loading AI so Jaseci Knows its There

- In jsctl
  - Simply run `actions load module …` to load AI modules from jaseci-kit
- You validate with `actions list`
  - Shows all available actions your jac code can have

```
ninja@DESKTOP-VO9IVBR:~/jac_convAI/tinker$ jsctl -m
Starting Jaseci Shell...
jaseci > actions load module jaseci_kit.use_qa.use_qa
```

```
jaseci > actions list
[
  "net.max",
  "net.min",
  "net.root",
  "rand.seed",
```

…

```
  "date.quantize_to_day",
  "date.date_day_diff",
  "use.question_encode",
  "use.enc_question",
  "use.answer_encode",
  "use.enc_answer",
  "use.cos_sim_score",
  "use.dist_score",
  "use.qa_score"
]
jaseci >
```

Using USE QA

- Let’s have some fun
- AI is simple and magic

```
walker init {
    can use.enc_question, use.enc_answer;

    answers = ['I am 20 years old', 'My dog is hungry', 'My TV is
broken'];
    question = "If I wanted to fix something what should I fix?";

    q_enc = use.enc_question(question);
    a_enc = use.enc_answer(answers); # can take lists or single
strings

    a_scores=[];

    for i in a_enc:
        a_scores.l::append(vector.cosine_sim(q_enc, i));

    report a_scores;
}
```