



KeppyLab

WorldEnder.ai

By James Dominguez

Who am I?

James Dominguez

- Founder at KeppyLab
 - AI for biomedical rare disease understanding
 - Language models that can tokenize ancient Greek
 - WorldEnder.ai (this talk)
-

Can LLMs replace
backends?

—

The Dark Ages: un-typed prompting

In the beginning, all we had was ChatGPT

To get structured outputs, we would do something like...

```
resp = client.chat.completions.create(  
    model="gpt-3.5-turbo",  
    messages=[  
        {"role": "user", "content": "Please give me character properties who were in the order of the phoenix as a json object ```json\n"},  
    ],  
    n=5,  
    temperature=1,  
)
```

The Dark Ages: un-typed prompting example #1

```
...  
  {  
    "name": "Neville Longbottom",  
    "house": "Gryffindor",  
    "role": "Member of Dumbledore's Army",  
    "wand": "Cherry and unicorn hair"  
  }  
...  
  {  
    "name": "Sirius Black",  
    "house": "Gryffindor",  
    "role": "Member of the Order of the Phoenix",  
    "wand": "Unknown",  
    "patronus": "Unknown"  
  }
```

The Dark Ages: un-typed prompting example #2

```
...  
  {  
    "name": "Molly Weasley",  
    "age": "",  
    "role": "Member of the Order of the Phoenix"  
  }  
...  
  {  
    "name": "Severus Snape",  
    "house": "Slytherin",  
    "role": "Head of Slytherin House, member of the Order of the Phoenix",  
    "animagus": false,  
    "patronus": "doe"  
  }
```

Too much ambiguity

Three issues with un-typed prompt engineering

1. Random dictionary structures
 2. No type guarantees
 3. Conceptual ambiguity
-

Some sort of data
validation would
be nice.

Finding Our Way; Pydantic

Pydantic Evolution:

- Evolved into a robust data validation library

Impact of PEP 593 (2019):


- Introduced 'Annotated' to Python for enhanced type annotations
- Allowed attaching runtime metadata to types without altering type checker interpretation
- Pydantic leveraged this feature to further improve data schema definitions and validation

Finding Our Way; Pydantic




Why are these kids
obsessed with Pydantic?

Finding Our Way; Pydantic

✓ 0s  `from pydantic import BaseModel`

```
class Wizard(BaseModel):  
    """  
    A non-muggle character from the Harry Potter universe  
    """  
    name: str  
    age: int  
    house: str  
    role: str  
  
harry = Wizard(name="Harry Potter", age="15", house="Gryffindor", role="Dumbledore's Army")  
print(harry)
```

 `name='Harry Potter' age=15 house='Gryffindor' role="Dumbledore's Army"`

Pydantic is smart enough to cast “15” to 15

Finding Our Way; Pydantic

✓
0s

```
[6] harry.model_json_schema()
```

```
➞ {'description': 'A non-muggle character from the Harry Potter universe',  
   'properties': {'name': {'title': 'Name', 'type': 'string'},  
                  'age': {'title': 'Age', 'type': 'integer'},  
                  'house': {'title': 'House', 'type': 'string'},  
                  'role': {'title': 'Role', 'type': 'string'}},  
   'required': ['name', 'age', 'house', 'role'],  
   'title': 'Wizard',  
   'type': 'object'}
```

What can we do with a
JSON schema?

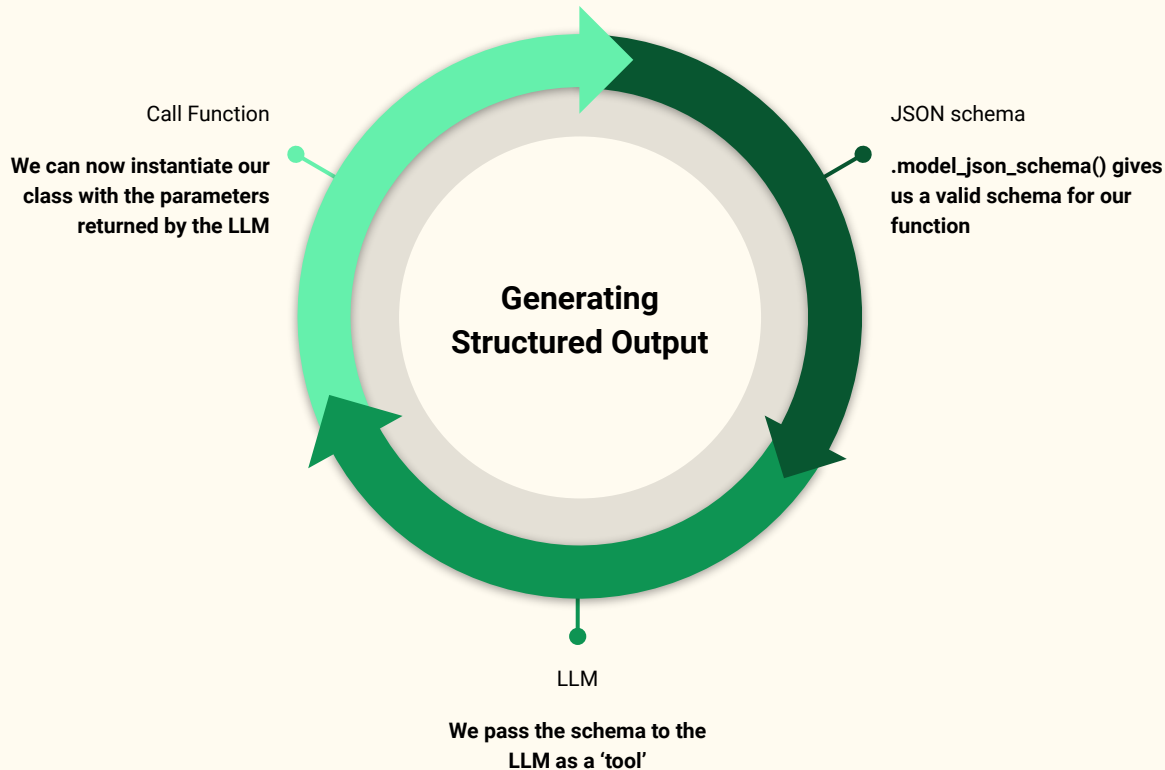
—

Seeing the Light: Function Calling

Function Calling allows us to describe functions a model, which is fine tuned to generate a JSON object containing arguments to call those functions.

Python functions
are objects

Seeing the Light; Function Calling



instructor

Python library that handles Function Calling using Pydantic types for us

It enhances the OpenAI API with a new parameter: **response_model**

Other tools exist for this.

What does this have to
do with simulations?

—

Reference Papers

- **Learning Interactive Real-World Simulators** <https://arxiv.org/abs/2310.06114>
- **Large Language Models as Gaming Agents** <https://openreview.net/forum?id=iS3fQooCaa>
- **Generative AI in Mafia-like Game Simulation** <https://arxiv.org/abs/2309.11672>
- **PANGeA: Procedural Artificial Narrative using Generative AI for Turn-Based Video Games** <https://arxiv.org/abs/2404.19721>

Simulation Algorithm

1. Generate an **Event** from a text player query that has a list of possible **Outcomes**
2. The player sees *only* a list of 3-5 choices per **Outcome**—predicted to cause each **Outcome**
3. The player selects *one* choice—the string is fed back into the **Event** generator (1.) as a new player query
4. After x cycles of player multiple choice prompting, the final model outputs a **WorldEnd** prediction

Let me show you
in Colab...

https://github.com/keppy/WorldEnder.ai/blob/master/WorldEnder_ai.ipynb

Simulation Algorithm Support

The data output from the experiment is repeatable & observable...

We found that when we followed the flow of data through the algorithm; the player's choices used as future prompts impacted the final outcome in a novel way.

Interaction between *player*, *generator*, and *context* enables diverse open ended simulation.

Variables that may affect the outcome...

- The context
- The player's bias
- The player's written english capabilities
- The capabilities of the specific model used

Conclusion

We can build robust, novel simulations with structured output, clever algorithms, and context. More experimentation and testing is needed to explore how far this can go (i.e. RAG + KG).



github.com/keppy/WorldEnder.ai



www.keppylab.com

Other projects

Research to the People and Stanford Medicine's Rare Disease AI Hackathon:

- Disease Lab: knowledge graph informed AI for disease research
- Hypophosphatasia RAG: hackathon open source team prototype

Talk to me if you're interested!

Thank you!

