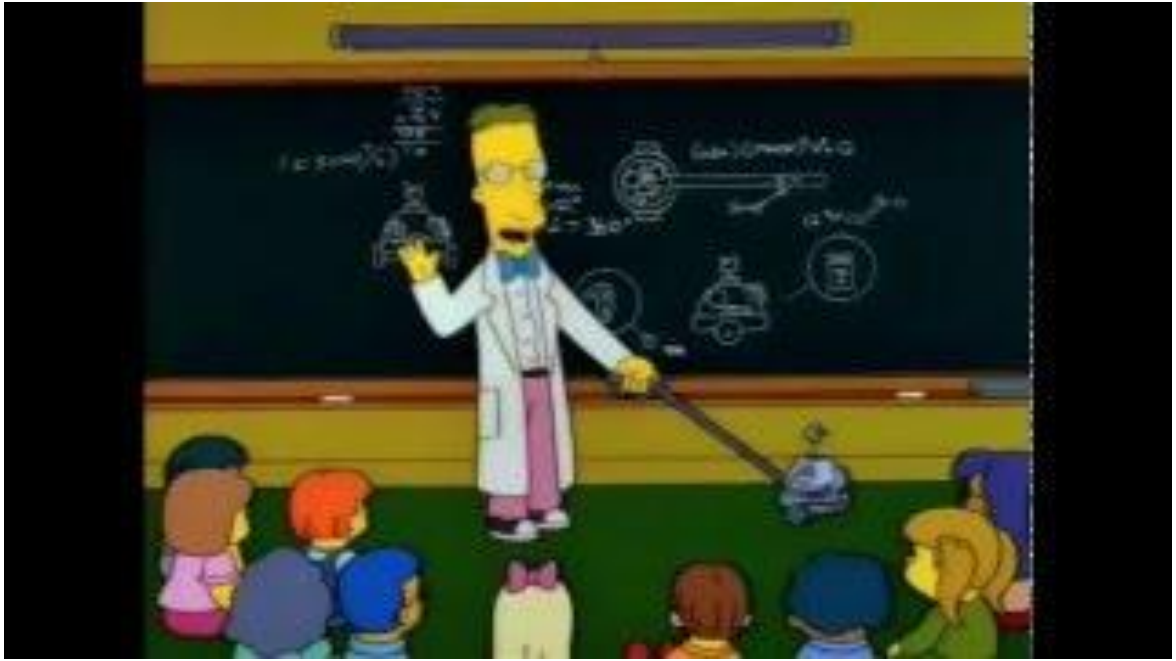
Documentation is Code (with agentic AI)

Allow billions of people to build billions of solutions

You won’t enjoy it on as many levels



What is DisC

- Documentation is Code(DisC)
- Any solution where instructions written using natural language are *executed* by an Agent

Working with Agentic AI

- What is an agent?
- High learning curve
- LLMs produce excessive output
- Inability to modify output
- No empathy, no mercy

Food for thought

- What if we are not looking for an answer?
- What if it is ok to make mistakes?
- Maybe agents are not supposed to solve every problem?

Evolution of Code

- Tubes, punch cards, assembly tapes
- Higher level languages to meta-frameworks
- No code
  - a GUI that builds things for you
  - Millions of products
- Documentation is Code
  - Just write the requirements and execute

Billions of people to build billions of solutions

- What about runtime performance?
- f(problem) = solution
  - The solution is a function of the problem
- Time to build is high, but immediate benefit is low
  - Proof of Concept
  - Personal projects
  - Internal tooling
- Uniquely personalized experiences

Demo

- Build a dashboard for our GitHub repo
  - The requirements keep changing
- Start with a traditional solution
- Then transform to a DisC solution

Requirements

- For the most prolific author of GitHub issues, give me the number of issues they have created.

- Given the author's issues, I want to know the number of comments that the same author has made.

Frontend

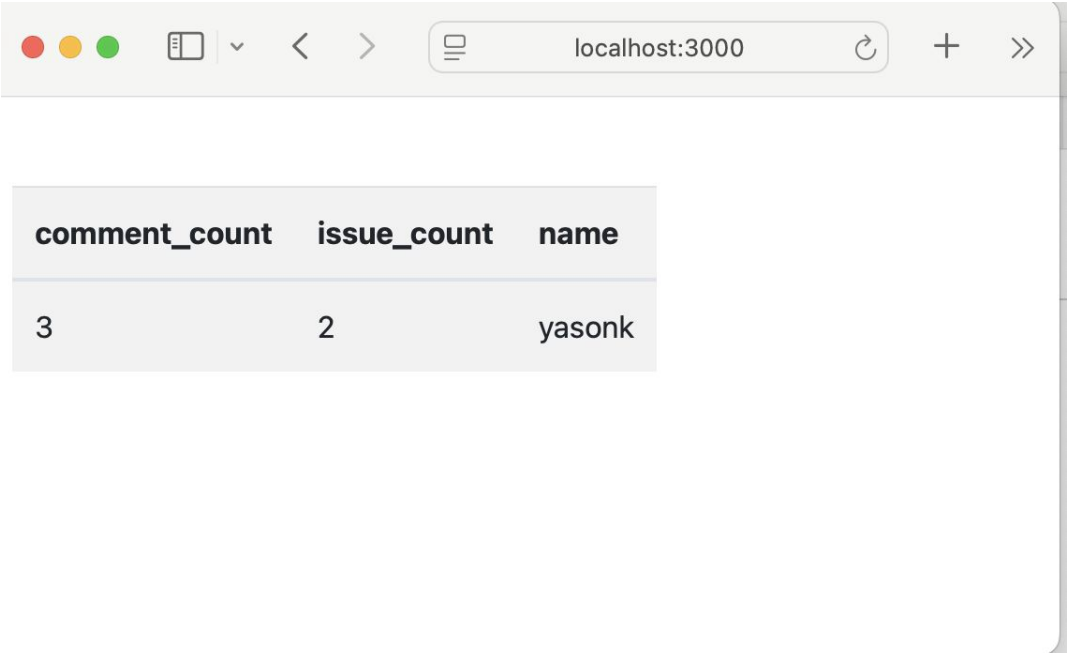
```
...

<div class="container mt-5">
        <table class="table table-striped">
            <tbody id="dataTable">
                <!-- Data will be populated here -->
            </tbody>
        </table>
    </div>

    <script>
        async function loadData() {
            const response = await fetch('http://localhost:5001/data');

...
```

comment_count	issue_count	name
3	2	yasonk

Backend

```
for issue in issues:
  author = issue["user"]["login"]
  author_count[author] = author_count.get(author,0) + 1

author = max(author_count, key=author_count.get, default=None)

issue_count = author_count.get(author, 0)

comments_count = sum(1 for comment in comments if comment["user"]["login"] == author)


result = {
    "data": [
        {"name": author, "issue_count": issue_count, "comment_count": comment_count}
    ]
}
return jsonify(result)
```

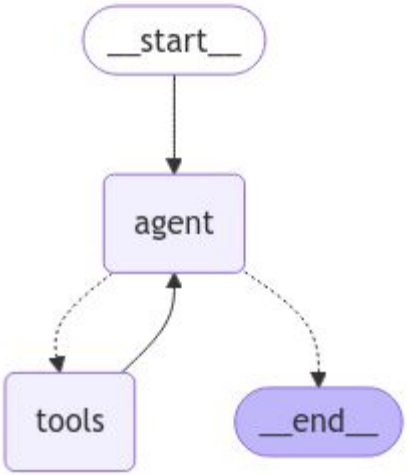
Result



comment_count	issue_count	name
3	2	yasonk

ReAct Agent

- Reason - Act - Observe - Iterate
- ReAct : Synergizing Reasoning And Acting In Language Models https://arxiv.org/pdf/2210.03629
- Tool calling support
- Prebuilt LangGraph ReAct agent



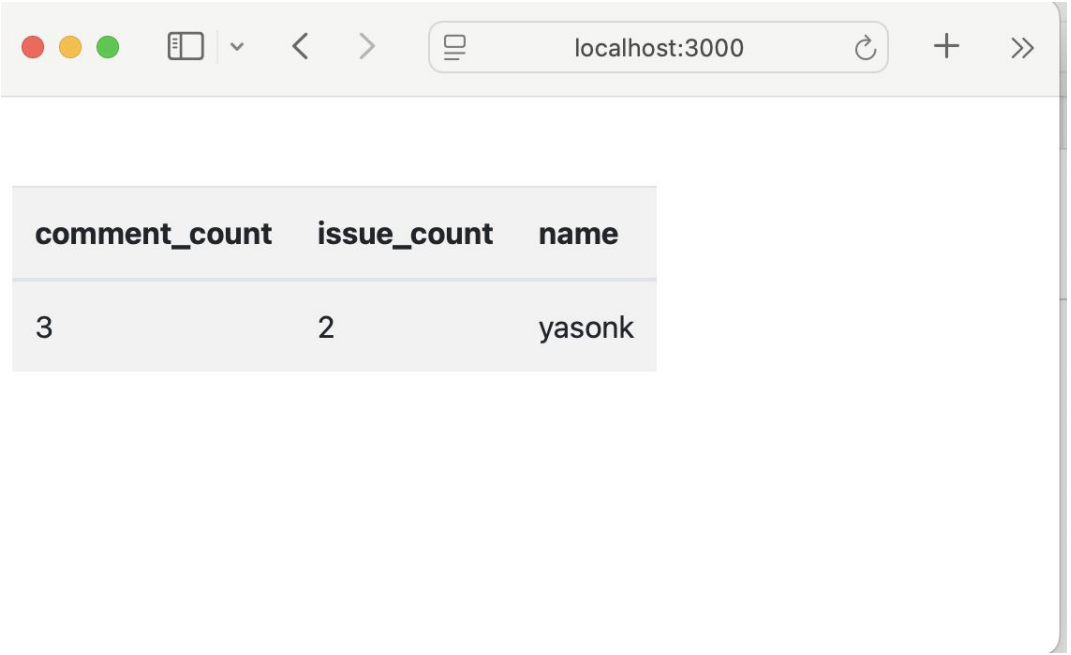
Endpoint Implementation

```
class Author(BaseModel):
    name: str
    issue_count: int
    comment_count: int


@app.route('/data', methods=['GET'])
def get_data():
    query = ("""
Given my repo: https://github.com/breba-apps/TempRepo
For the most prolific author of github issues, give me the number of issues they have created.
Given the author's issues, I want to know the number of comments that same author has made.
""")

    return ai(query, response_format=Author)
```

ReAct Agent Implementation

```python
llm = ChatOpenAI(model="gpt-4.1", temperature=0)

def ai(query: str, response_format: Type[BaseModel]):
    async with github_mcp() as tools:
        agent_executor = create_react_agent(llm, tools,response_format=response_format)
        events = agent_executor.stream({"messages": [("user", query)]}, stream_mode="values")

        event = {}
        for event in events:
            event["messages"][-1].pretty_print()

        structured_response = event.get("structured_response")
        return json.dumps({"data": [structured_response.model_dump()]})
```

Result



comment_count	issue_count	name
3	2	yasonk

Endpoint Implementation

```
class Author(BaseModel):
    name: str
    issue_count: int
    comment_count: int


@app.route('/data', methods=['GET'])
def get_data():
    query = ("""
Given my repo: https://github.com/breba-apps/TempRepo
For the most prolific author of github issues, give me the number of issues they have created.
Given the author's issues, I want to know the number of comments that same author has made.
""")

    return ai(query, response_format=Author)
```

New Requirements

- Find the most prolific author of github issues, and I want to see all of their issues and comments
- I will need to see only the author, issue title, and comments.

ReAct Agent Implementation

```
...
...
events = agent_executor.astream(
    {"messages": [("system",
                   "You are a web server, serving HTML web pages. Do not use markdown for formatting."
                   "All output must be in HTML format and will be displayed to an end user."
                   "You need to start with doctype and html tags and provide the entire page.")
                   ("user", query)]},
    stream_mode="values")
...
```

Endpoint Implementation

```
@app.route('/', methods=['GET'])
def get_data():
    return asyncio.run(ai(Path("my_app-step-3-mcp.txt").read_text()))
    # Where is all the code?


# Given my repo: https://github.com/breba-apps/TempRepo
# Find the most prolific author of github issues, and I want to see all of their issues and comments

# I will need to see ONLY the author, title, and comments.
# HTML elements can be styled using bootstrap css.
```

Result

Most Prolific Issue Author

yasonk

First issue here you go

Comments:

No comments by yasonk on this issue.

This is my second issue

Comments:

yasonk: Comment 1, because I'm building a community

yasonk: Comment 2, Because need to keep it going

yasonk: Comment 3, because I'm awesome

Note: Only issues and comments authored by **yasonk** are shown.

New Requirements

- Below the comments, include a link to the issue on GitHub.
- Use accordion style display to make it easier scroll through the entries

  (Open Jupyter Notebook)

Limitations

- Tools still need to be coded
- Must learn agent specializations and limitations
- Runtime performance is slow
- Inconsistent outputs
  - Blasts out overreaching solutions
  - Incremental refinement is challenging
- Engineering rigor
  - Security
  - Scalability
  - Debugging

Next Steps

- Tools to be dynamic without requiring new code
  - Simply drop in documentation for a tool
  - Provide URL to a new MCP server
- Stateful agents
  - Checkpointing and persistence
  - Iterative development
  - Output consistency
- Improve human-in-the-loop processes
  - Human in the loop is unavoidable
  - Assist with debugging

Website Building DisC Agent

- Less than 90 line of code
- Unique perspective into the internet
- No more ad-driven platforms
- No more attention harvesting platforms
- No more opaque algorithms keeping us siloed

Conclusion

- With DisC, users will write their specification and the agent will run it
- Solve of problems rarely solved before
  - Developer setup instructions
  - A small website for your party
  - A tennis league at work
  - A behavior tracking app for your child
- Billions of people will be able to solve billions of highly personalized problems

presentation

LinkedIn