Why you should build your GenAI/LLM apps using Hamilton

👉 Five reasons why

Stefan Krawczyk
CEO & Co-Founder
DAGWORKS (YCW23)

AICamp December 2023

historically:

Honda Research In

Some questions from me :)

Agenda

1. Challenges
2. Hamilton

1. Challenges

(1) Everything's new…

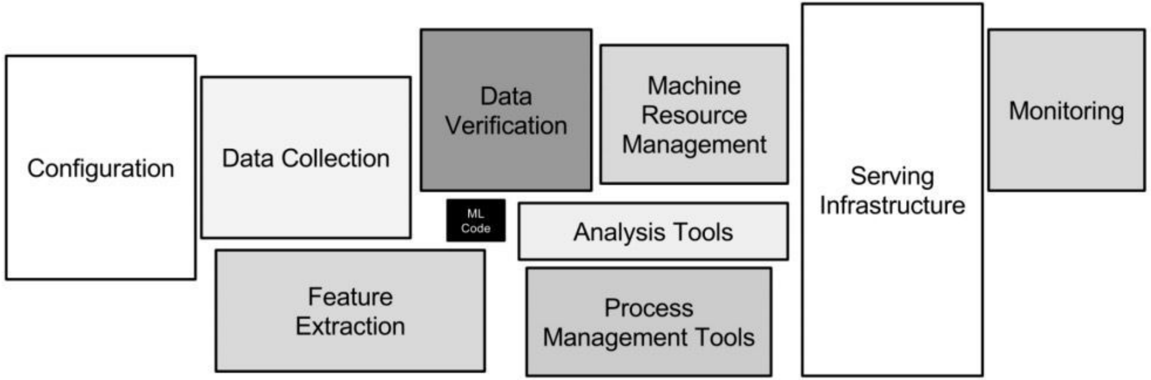
(2) Pace of change & development

(3) All this requires SWE skills

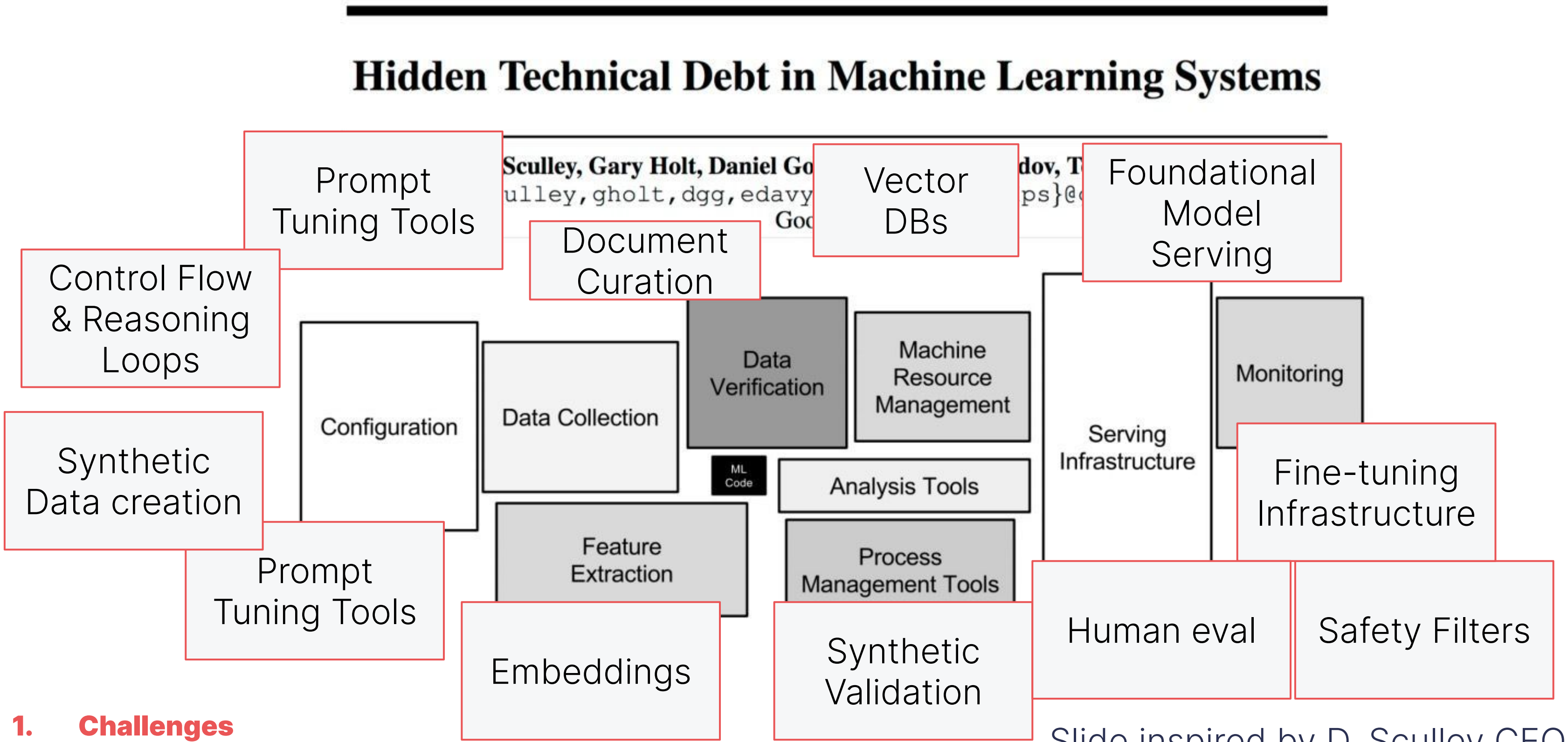
Anyone remember this?

Hidden Technical Debt in Machine Learning Systems

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips
`{dsculley,gholt,dgg,edavydov,toddphillips}@google.com`
Google, Inc.



GenAI/LLM Apps are no different



Slide inspired by D. Sculley CEO of Kaggle

1. **Challenges**

SWE Development

is less this:

and more this:

(3) SWE challenges

Get it wrong:

1. IC: Tech debt & pipeline/workflow/code inheritance 😱
2. Business: High cost to change & slower to develop.

Get it right:

1. IC: Ship more & get faster promotions.
2. Business: higher ROI

(3) SWE challenges

Get it wrong:

Characteristics:

1. Change with confidence → testing
2. Swappable parts → modularity
3. Make tweaks/warm start → reusability
4. Layer on your concerns → portability, pluggability, & extensibility

2. Business: higher ROI

2. Hamilton

What is Hamilton?

Micro-orchestration framework
for defining dataflows
using declarative functions

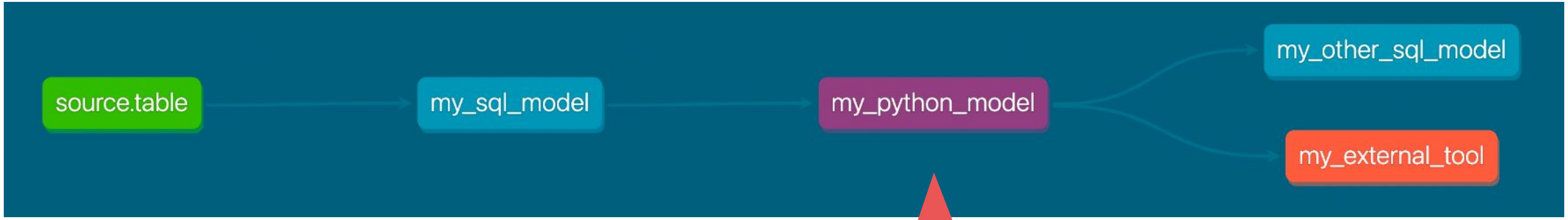
SWE best practices: ☑️ testing ☑️ documentation ☑️ modularity/reuse ☑️ iteration

`pip install sf-hamilton [came from Stitch Fix]`

www.tryhamilton.dev ← uses pyodide!

Micro-orchestration vs Macro-orchestration

Macro-orchestration is this whole thing (ETLs, web service requests, etc):

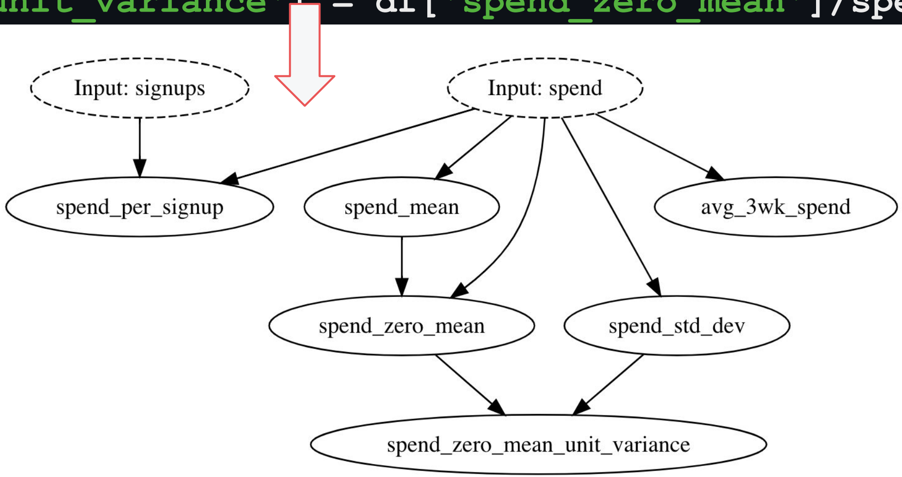


Micro-orchestration handles what happens within this step

What do you mean by dataflow?

Dataflows represent how your procedural code flows:

```
df['avg_3wk_spend'] = df['spend'].rolling(3).mean()
df['spend_per_signup'] = df['spend']/df['signups']
spend_mean = df['spend'].mean()
df['spend_zero_mean'] = df['spend'] - spend_mean
spend_std_dev = df['spend'].std()
df['spend_zero_mean_unit_variance'] = df['spend_zero_mean']/spend_std_dev
```



Declarative functions?

Functions *declare:*

- What they create in the dataflow.
- What dependencies are required for computation.

You don’t run the functions directly.

> When you read the function, you’ll understand what it does and what it needs.

Old Way vs. Hamilton Paradigm:

Instead of

```
c = f"Some prompt using {a} & {b}"
d = custom_logic(llm_api_call(c))
```

Outputs == Function Name **Inputs == Function Arguments**

You declare

```
def c(a: str, b: int) -> str:
    """Creates prompt"""
    return f"Some prompt using {a} & {b}"

def d(c: str) -> str:
    """Transform/send to LLM ..."""
    response = custom_logic(llm_api_call(c))
    return response
```

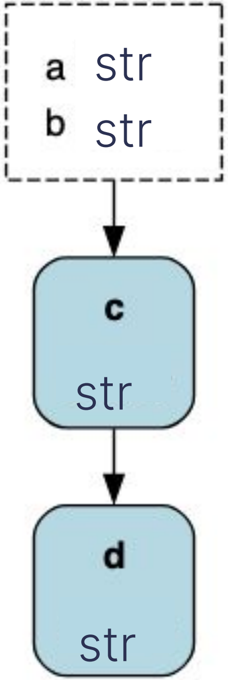
Full Hello World

(Note: works for any python object type)

Functions

```python
# llm_chain.py
def c(a: str, b: int) -> str:
    """Creates prompt"""
    return f"Some prompt using {a} & {b}"

def d(c: str) -> str:
    """Transform/send to LLM ..."""
    response = custom_logic(llm_api_call(c))
    return response
```



Driver says what/when to execute

```python
# run.py
from hamilton import driver
import llm_chain
dr = driver.Driver({'a': ..., 'b': ...}, llm_chain, adapter=...)
result = dr.execute(['c', 'd'])
print(result)
```

Full Hello World

(Note: works for any python object type)

Functions

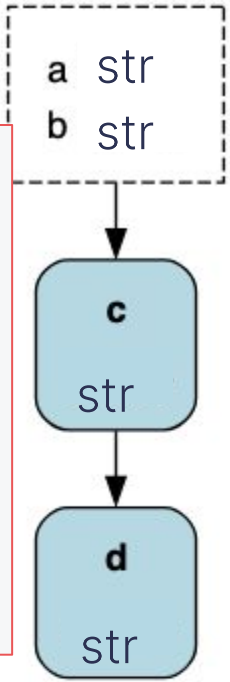
```
# llm_chain.py
def c(a: str, b: int) -> str:
    """Creates prompt"""
```

🤔 Yes, you can use it to replace (even use with):

Langchain

Llama Index

etc.



Driver says wh

```
# run
from 
import llm_chain
dr = driver.Driver({'a': ..., 'b': ...}, llm_chain, adapter=...)
result = dr.execute(['c', 'd'])
print(result)
```

Things to mention, but won't really cover:

We also have decorators that you add to functions that...

- `@tag` `# attach metadata`
- `@parameterize` `# curry + repeat a function`
- `@extract_columns` `# one dataframe -> multiple series`
- `@extract_outputs` `# one dict -> multiple outputs`
- `@check_output` `# data validation; very lightweight`
- `@config.when` `# conditional transforms`
- `@subdag` `# parameterize parts of your DAG`

Some Hamilton users we know of

STITCH FIX

BRITISH CYCLING

Joby AVIATION

TRANSFIX

IBM

ascena RETAIL GROUP INC.

AXIOM CLOUD

OAK RIDGE National Laboratory

HABITAT ENERGY

Government Digital Service

Pacific Northwest NATIONAL LABORATORY

LexisNexis RISK SOLUTIONS

Opendoor

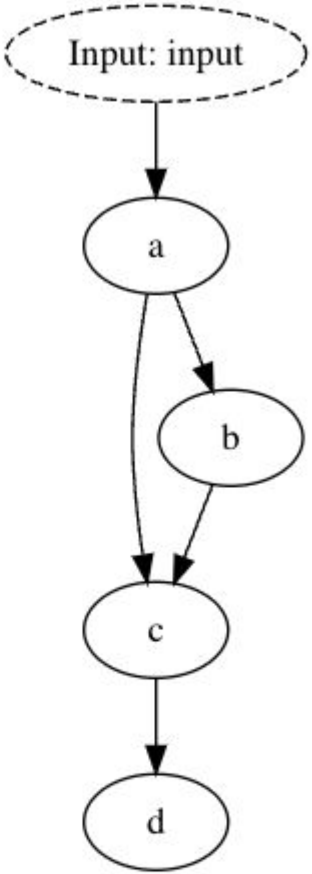
Five Reasons

1: One less tool to learn

With Hamilton you can describe & glue together:

1. Data processing
2. Feature engineering
3. Machine learning
4. GenAI/LLM
5. Web request
6. Etc

pipelines / workflows / dataflows / etc.



2: Portable, Pluggable & Extensible

Your code is **portable** & runs **& scales** anywhere python runs:

RAY dask APACHE Spark

FastAPI Flask

jupyter

👀 VLDB Workshop Papers

Hamilton: a modular open source declarative paradigm for high level modeling of dataflows

Stefan Krawczyk
skrawczyk@stitchfix.com
stefank@cs.stanford.edu
Stitch Fix
San Francisco, California, USA

Elijah ben Izzy
elijah.benizzy@stitchfix.com
Stitch Fix
San Francisco, California, USA

ABSTRACT

As the role of data in industry has grown, the need for specific data management tooling has followed. While a hello world example for a typical machine learning workflow might look trivial, once one layers in industry concerns such as data & computational lineage, data quality/observability, scalability, unit testing, code base maintenance and documentation, this melange of specific tooling often results in a poor end to end user experience with high engineering effort. A
solve a subset of
simplifying the us
that the paradigm
unified interface f
way that facilitates modularity of data management system tooling by forcing a clear decoupling of concerns. It does this by requiring a programming paradigm change on part of the user that enables easy specification and execution of dataflow graphs. Hamilton therefore represents a novel high level approach to modeling dataflows, and presents an industry pragmatic avenue for building a simpler user experience that can easily integrate with existing data management tooling in a modular fashion. Hamilton is available as open source code.

PVLDB Reference Format:
Stefan Krawczyk and Elijah ben Izzy. Hamilton: a modular open source declarative paradigm for high level modeling of dataflows. PVLDB, 14(1): XXX-XXX, 2022.
doi:XX.XX/XXX.XX

1 INTRODUCTION

An industry trend that we have lived through at Stitch Fix is the shift to "Full Stack Data Science"[1], where data scientists are expected to not only do data science, but also engineer and manage data pipelines for their production machine learning models. This approach places additional burdens on data scientists, who no longer hand off their ideas off to a software engineering team for implementation and maintenance. Previously, hand-offs allowed data scientists to focus on a specific domain and set of tooling to accomplish their work. They did not have to worry about such production concerns as, lineage, scalability, or data quality. All they had to do was build a model and prescribe the recipe for an engineering team to implement. In a "full stack" model, however, the data scientist has to pick up the engineering work and understand the complexities of implementing a production pipeline. This has made it all the more important to build streamlined experiences that reduce the complexity of their engineering work, while still enabling them to move quickly and adjust their pipelines as the business requires.

At Stitch Fix, the Hamilton framework[5] was conceived to miti-
ific
m's
en-
vith
ate,
maintain, and execute code for data transformations, especially in the case of highly complex data transformation dependency chains. Hamilton does this by deriving a directed acyclic graph (DAG) of dependencies from specially defined declarative Python functions that describe the user's intended dataflow. Altogether, Hamilton makes incremental development, code reuse, unit testing, lineage tracking, data quality checks, and code documentation natural and straightforward. Furthermore, its modularity provides avenues to quickly and easily scale computation onto various distributed frameworks, e.g. Ray[4]/Spark[11]/Dask[7], as well as extend the platform to integrate with other data management tools, e.g. lineage/governance and data quality. Hamilton has enabled data science teams at Stitch Fix to scale modeling dataflows to support 4000+ data transformations without impacting team and user productivity.

We will first ground ourselves with a basic extract, transform, load (ETL) approach to machine learning, then explain the requirements that guided Hamilton, and finally spend the rest of this paper diving into Hamilton's programming paradigm. We will show the benefits this paradigm brings, briefly discuss evaluation, propose future extensions, and finish with a summary.

2 CURRENT ETL APPROACHES

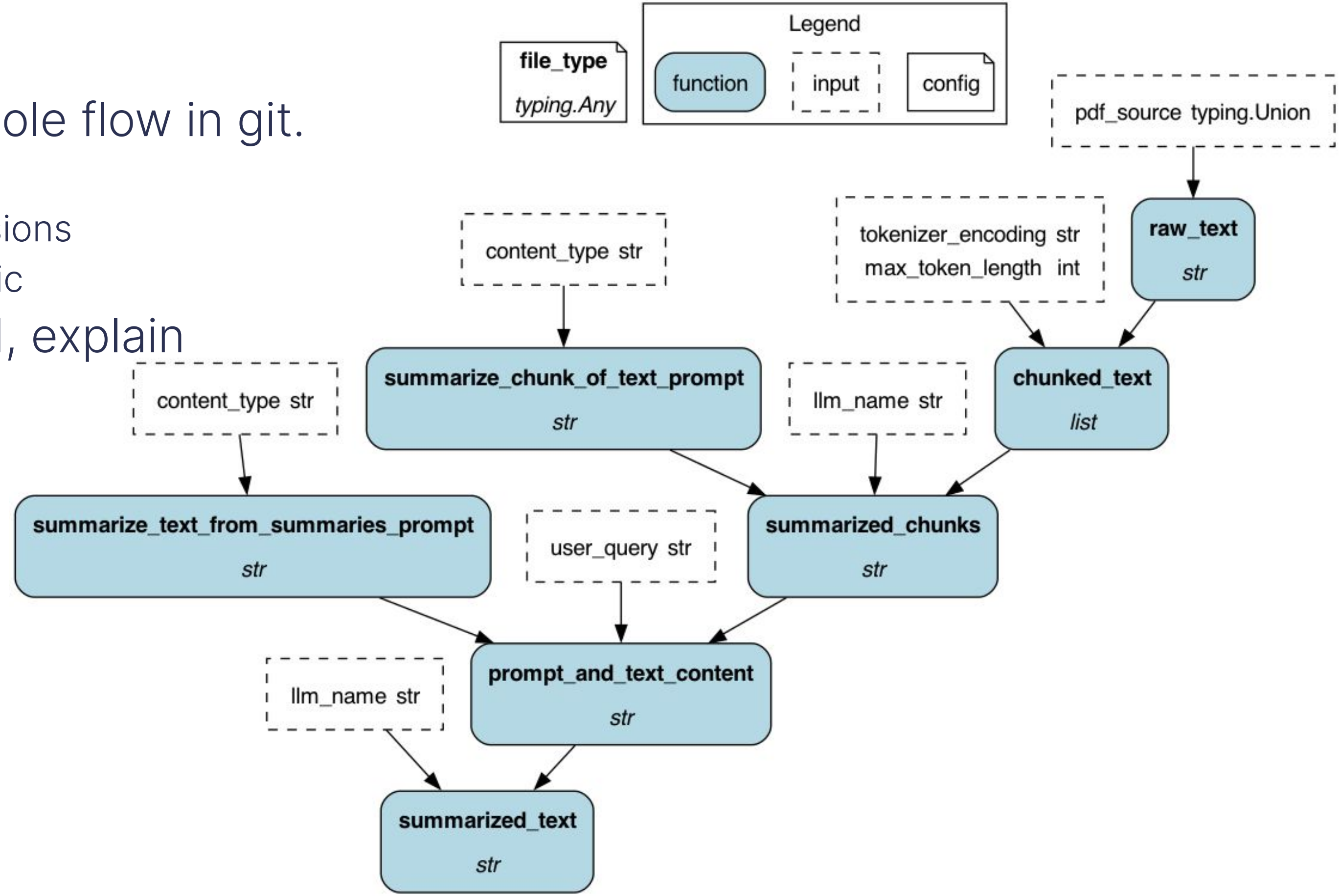
Bringing a machine learning model to production at Stitch Fix requires building an ETL workflow. One has to extract data (SQL or Python), transform it for input into a model (SQL or Python), transform it into a model (Python), transform data with the help of the model (Python), and finally load the results somewhere to connect it back with the business (SQL or Python). Furthermore, this has to be run on a cadence. If modeled as discrete steps then data/artifacts have to be materialized between them. An orchestration system, e.g. [6, 10], is responsible for scheduling and executing these steps.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

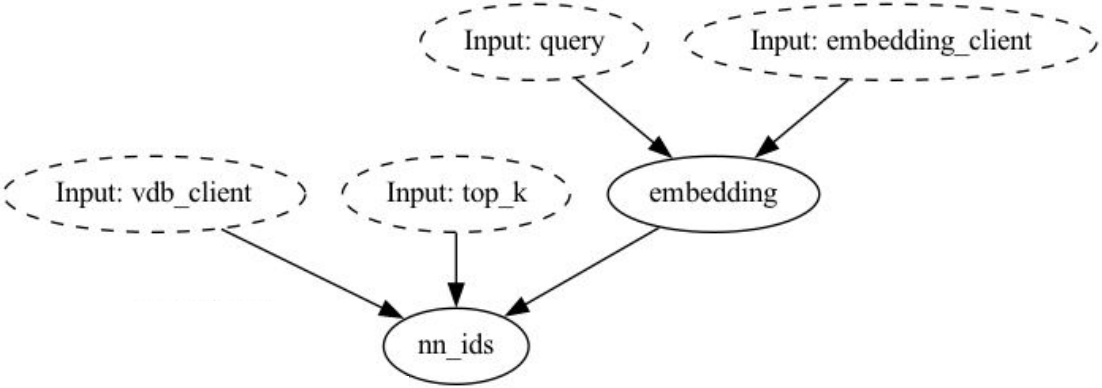
3: Lineage as Code

1. Version your whole flow in git.
   a. Prompts
   b. Model/API versions
   c. Processing logic
2. Debug, onboard, explain faster.



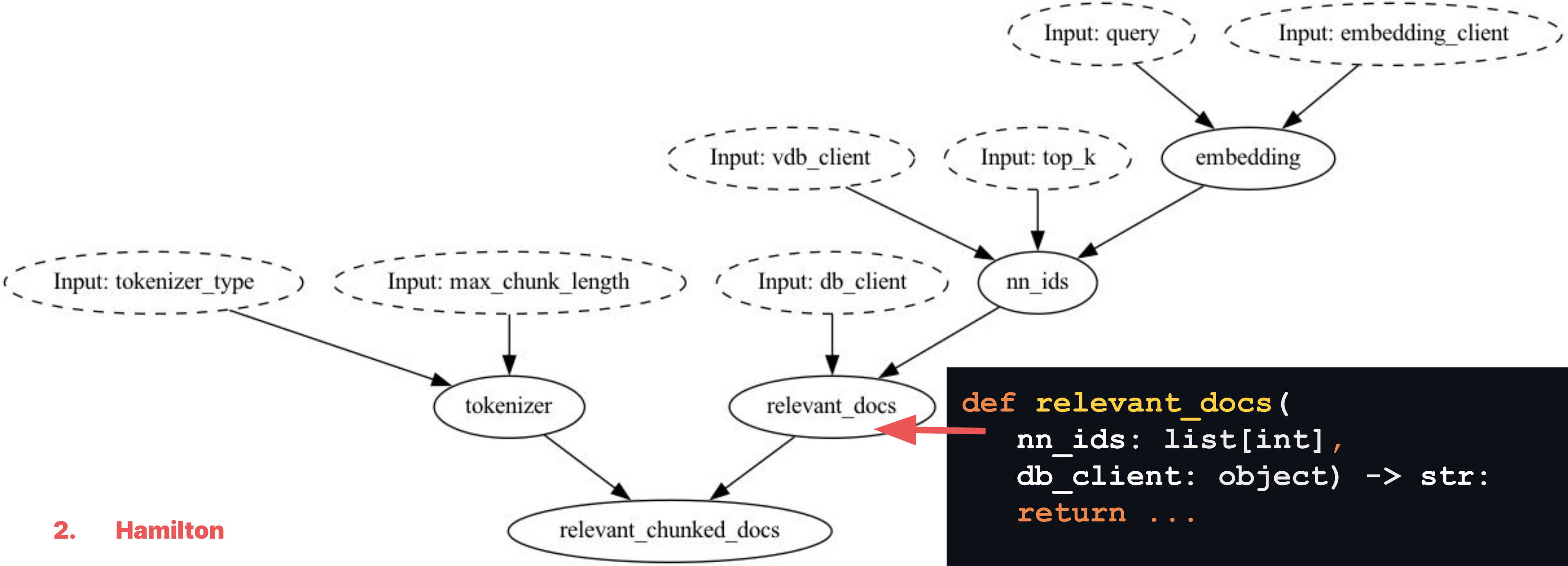
2. Hamilton

4: Modularity & Reuse



4: Modularity & Reuse

1. Straightforward to compose & reuse flows.

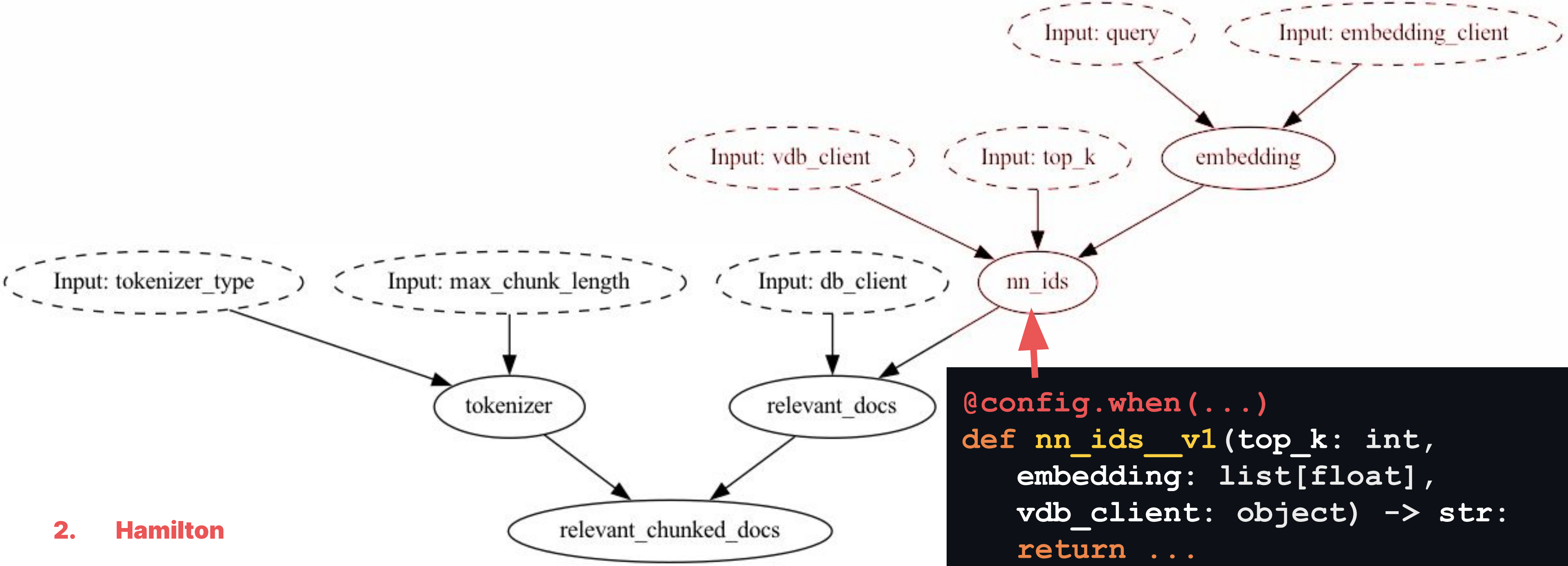



```python
def relevant_docs(
    nn_ids: list[int],
    db_client: object) -> str:
    return ...
```

2. **Hamilton**

4: Modularity & Reuse

1. Straightforward to compose & reuse flows.
2. Easy to switch between multiple “implementations”




```python
@config.when(...)
def nn_ids__v1(top_k: int,
    embedding: list[float],
    vdb_client: object) -> str:
    return ...
```

5: Testing & Documentation

```python
# use_case.py

def example_system_prompt(a: str, b: int) -> str:
    """More documentation would go here"""
    return f"Some prompt using {a} & {b}"
```

Testing: easier to unit & integration test (e.g. evals in CI/CD)

```python
# test_use_case.py

def test_example_system_prompt():
    actual = example_system_prompt("some input", 2.0)
    expected = f"Some prompt using some input & 2.0"
    assert actual == expected
```

5: Testing & Documentation

```python
# use_case.py

@check_output(data_type=str, some_property=value)
def example_system_prompt(a: str, b: int) -> str:
    """More documentation would go here"""
    return f"Some prompt using {a} & {b}"
```

Testing: easier to unit & integration test (e.g. evals in CI/CD)

Data Quality Tests: runtime checks via annotation*; Pandera supported. Pydantic on roadmap.

5: Testing & Documentation

```python
# use_case.py

@tag(owner='Data-Science', pii='False')
@check_output(data_type=str, some_property=value)
def example_system_prompt(a: str, b: int) -> str:
    """More documentation would go here"""
    return f"Some prompt using {a} & {b}"
```

Testing: easier to unit & integration test (e.g. evals in CI/CD)

Data Quality Tests: runtime checks via annotation*; Pandera supported. Pydantic on roadmap.

Self-documenting: naming, doc strings, annotations, & visualization

5: Testing & Documentation

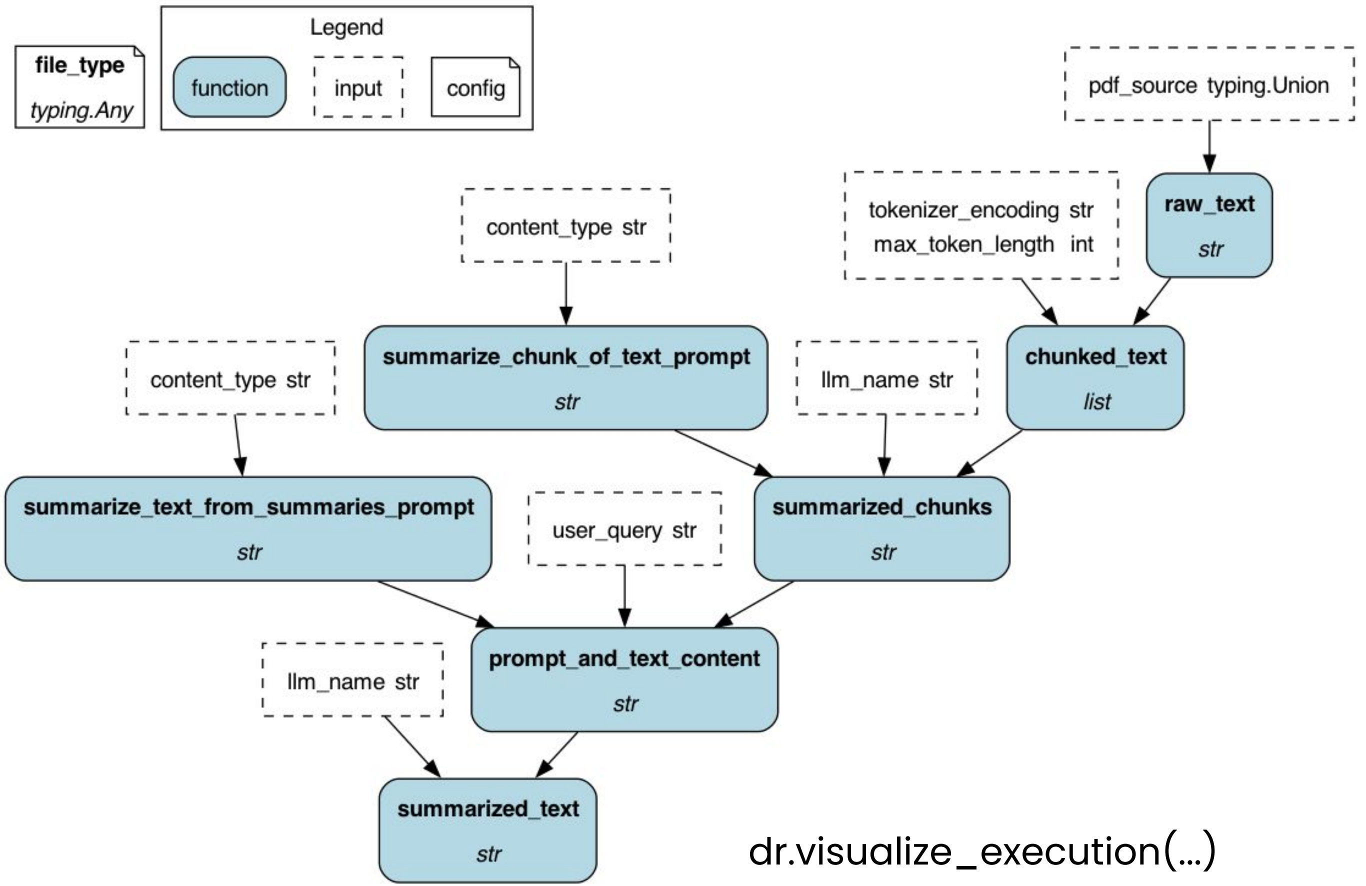
```
# use_case.py

@tag(owner='Da
@check_output
def example_sy
    """More do
    return f"S
```

Testing: ea

Data Qualit
Pydantic or

Self-docun



dr.visualize_execution(...)

2. **Hamilton**

Hamilton: build your GenAI/LLM apps on Hamilton

Problem:

- Pace of change & iteration → need good SWE practices to not 💥

With Hamilton → 🏎️ :

1. One tool – for data, web request, ML, and GenAI/LLM work.
2. You can port, plug and extend your code and the framework.
3. Version, debug & understand faster with *lineage as code*.
4. Naturally have modular and reusable, without much 🤔.
5. Never complain again about testing & documentation.

Want the “langsmith” equivalent but for Hamilton?

(1) Stop by our table for a demo
(2) Come see a toy GenAI app built with Hamilton

DAGWORKS

(3) 📣 we’re looking for a GenAI/LLM partner

www.dagworks.io
Versioning, Lineage, Catalog, Observability
[Free trial]

Get started:

`pip install sf-hamilton`

▶: tryhamilton.dev ← runs 🐍 in the browser!

▶: hub.dagworks.io ← our bank of dataflows to get started in 3 lines

🤓: blog.dagworks.io ← various posts e.g. RAG, prompts, etc.

⭐: https://github.com/dagworks-inc/hamilton (see examples/)

📣: Join us on slack

DAGWORKS

Questions?

HAMILTON

Get started:

```
pip install sf-hamilton
```

▶: tryhamilton.dev ← runs 🐍 in the browser!

▶: hub.dagworks.io ← our bank of dataflows to get started in 3 lines

🤓: blog.dagworks.io ← various posts e.g. RAG, prompts, etc.

⭐: https://github.com/dagworks-inc/hamilton (see examples/)

📣: Join us on slack

DAGWORKS

Questions?

Get started:

`pip install sf-hamilton`

▶: tryhamilton.dev ← runs 🐍 in the browser!

▶: hub.dagworks.io ← our bank of dataflows to get started in 3 lines

🤓: blog.dagworks.io ← various posts e.g. RAG, prompts, etc.

⭐: https://github.com/dagworks-inc/hamilton (see examples/)

📣: Join us on slack

DAGWORKS

Questions?

Get started:

`pip install sf-hamilton`

▶: tryhamilton.dev ← runs 🐍 in the browser!

▶: hub.dagworks.io ← our bank of dataflows to get started in 3 lines

🤓: blog.dagworks.io ← various posts e.g. RAG, prompts, etc.

⭐: https://github.com/dagworks-inc/hamilton (see examples/)

📣: Join us on slack

DAGWORKS

Questions?

Get started:

```
pip install sf-hamilton
```

▶: tryhamilton.dev ← runs 🐍 in the browser!

▶: hub.dagworks.io ← our bank of dataflows to get started in 3 lines

🤓: blog.dagworks.io ← various posts e.g. RAG, prompts, etc.

⭐: https://github.com/dagworks-inc/hamilton (see examples/)

📣: Join us on slack

DAGWORKS

Questions?

Get started:

`pip install sf-hamilton`

▶: tryhamilton.dev ← runs 🐍 in the browser!

▶: hub.dagworks.io ← our bank of dataflows to get started in 3 lines

🤓: blog.dagworks.io ← various posts e.g. RAG, prompts, etc.

⭐: https://github.com/dagworks-inc/hamilton (see examples/)

📣: Join us on slack

DAGWORKS

Questions?