



Hamilton Global User Group June 2024 Meetup

What is Hamilton?

Hamilton helps data scientists and engineers define testable, modular, self-documenting dataflows, that encode lineage and metadata.
Runs and scales everywhere python does.

Icebreaker: Name and what you're using Hamilton for/looking for.



Agenda

1. The “news”
2. Deep Dive
3. Open 





The “News”

Some Recent Features Released

- Newer IPython Magic
- `.with_materializers()`
- Graceful Error Adapter
- Hamilton UI
- Kedro
- MLFlow

Some Recent Features Released

- Newer IPython Magic – see [this notebook](#) (or via [colab](#)).

```
▶ ?%cell_to_module
```

```
⇒ Docstring:  
::
```

```
%cell_to_module [-m [MODULE_NAME]] [-d [DISPLAY]] [-x [EXECUTE]]  
                [-b BUILDER] [-c CONFIG] [-i INPUTS] [-o OVERRIDES]  
                [--hide_results] [-w [WRITE_TO_FILE]]  
                [module_name]
```

Turn a notebook cell into a Hamilton module definition. This allows you to define and execute a dataflow from a single cell.

For example:

```
```\n%cell_to_module dataflow --display --execute\ndef A() -> int:\n    return 37\n\ndef B(A: int) -> bool:\n    return (A % 3) > 2\n```\n
```

```
▶ ?%cell_to_module
```

```
⇒ Docstring:
::
```

```
%cell_to_module [-m [MODULE_NAME]] [-d [DISPLAY]] [-x [EXECUTE]]
 [-b BUILDER] [-c CONFIG] [-i INPUTS] [-o OVERRIDES]
 [--hide_results] [-w [WRITE_TO_FILE]]
 [module_name]
```

Turn a notebook cell into a Hamilton module definition. This allows you to define and execute a dataflow from a single cell.

For example:

```
```\n%cell_to_module dataflow --display --execute\ndef A() -> int:\n    return 37\n\ndef B(A: int) -> bool:\n    return (A % 3) > 2\n```\n
```

positional arguments:

module_name Name for the module defined in this cell.

options:

```
-m <[MODULE_NAME]>, --module_name <[MODULE_NAME]>  
    Alias for positional argument 'module_name'  
-d <[DISPLAY]>, --display <[DISPLAY]>  
    Display the dataflow. The argument is the name of a dictionary of visualization kwargs  
-x <[EXECUTE]>, --execute <[EXECUTE]>  
    Execute the dataflow. The argument is the name of a list of nodes; else execute all nodes  
-b BUILDER, --builder BUILDER  
    Builder to which the module will be added to the dataflow. Allows to pass Config and Adapter to the builder  
-c CONFIG, --config CONFIG  
    Config to build a Driver. Passing -c/--config at the same time as a Builder -b/--builder will raise an exception.  
-i INPUTS, --inputs INPUTS  
    Execution inputs. The argument is the name of a dict of inputs; else {}.  
-o OVERRIDES, --overrides OVERRIDES  
    Execution overrides. The argument is the name of a dict of overrides; else {}.  
--hide_results  
    Hides the automatic display of execution results  
-w <[WRITE_TO_FILE]>, --write_to_file <[WRITE_TO_FILE]>  
    Write cell content to a file. The argument is the path; else write to {module_name}.py
```

File: ~/projects/dagworks/hamilton/hamilton/plugins/jupyter_

Some Recent Features Released

with_materializers()

```
from hamilton import driver
from hamilton.io.materialization import from_, to
import my_dataflow

loader = from_.parquet(target="raw_df", path="/my/raw_file.parquet")
saver = to.parquet(
    id="features__parquet",
    dependencies=["features_df"],
    path="/my/feature_file.parquet"
)

dr = (
    driver.Builder()
    .with_modules(my_dataflow)
    .with_materializers(loader, saver)
    .build()
)

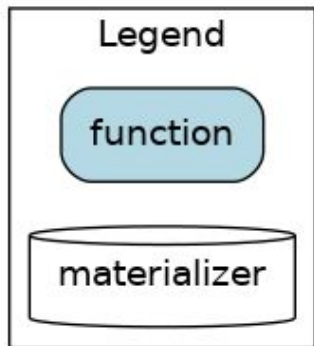
dr.execute(["features__parquet"])
```

Some Recent Features Released

with_materializers()

```
from hamilton import driver
from hamilton.io.materialization import from_, to_
import my_dataflow

loader = from_.parquet(target="raw_df", path="/my/raw_file.parquet")
save = to_.parquet()
```



```
dr.display_all_functions("dag.png")
```

Some Recent Features Released

- [GracefulErrorAdapter](#)

```
class DoNotProceed(Exception):
    pass

def wont_proceed() -> int:
    raise DoNotProceed()

def will_proceed() -> int:
    return 1

def never_reached(wont_proceed: int) -> int:
    return 1 # this should not be reached
```


Some Recent Features Released

- [GracefulErrorAdapter](#)

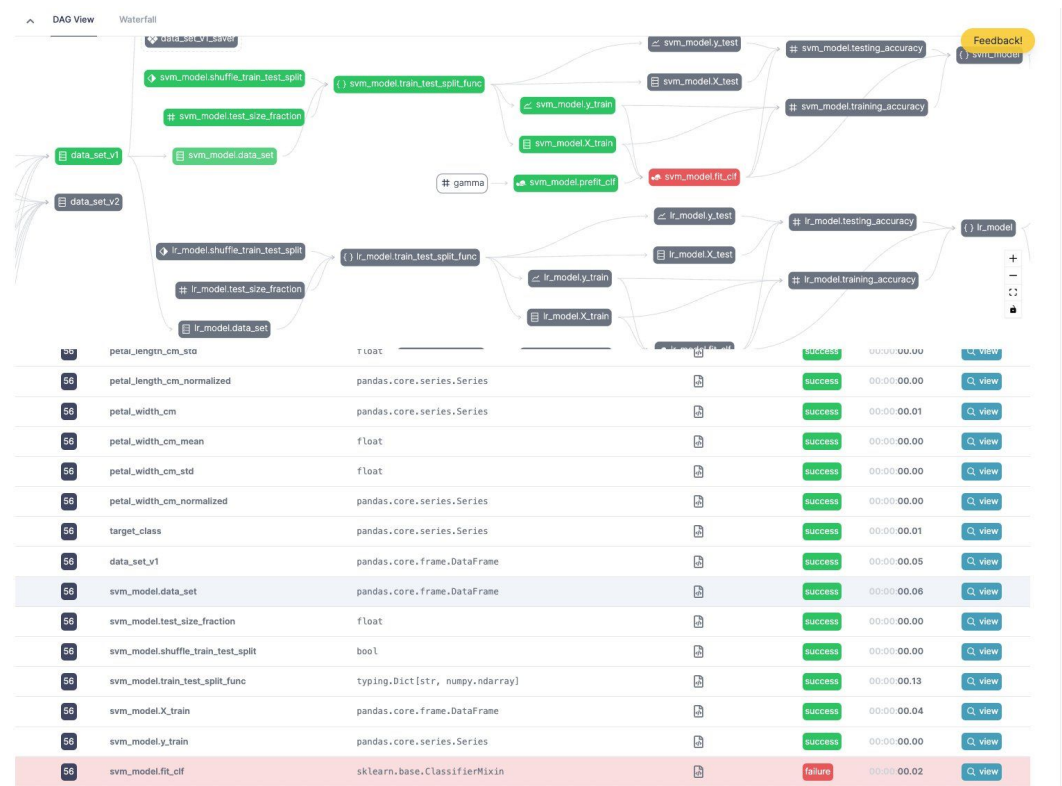
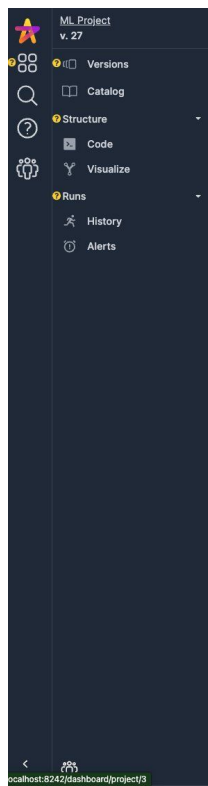
```
dr = (  
    driver.Builder()  
    .with_modules(my_module)  
    .with_adapters(  
        default.GracefulErrorAdapter(  
            error_to_catch=DoNotProceed,  
            sentinel_value=None  
        )  
    )  
    .build()  
)  
# will return {'will_proceed': 1, 'never_reached': None}  
dr.execute(["will_proceed", "never_reached"])
```

Some Recent Features Released

- Hamilton UI – pip installable + docker version

Key features:

- Visualize
- Version
- Catalog
- Telemetry



Some Recent Features Released

[Kedro](#) - [notebook](#)

```
from hamilton_sdk.adapters import HamiltonTracker
from hamilton import driver
from hamilton.plugins import h_kedro
from kedro_code.pipelines import data_processing,
data_science

# modify this as needed
tracker = HamiltonTracker(...)
builder = driver.Builder().with_adapters(tracker)

dr = h_kedro.kedro_pipeline_to_driver(
    data_processing.create_pipeline(),
    data_science.create_pipeline(),
    builder=builder
)
```

Some Recent Features Released

- [MLFlow](#)

Two use cases: – [video explainer](#)

- MLFlow data savers & loaders
- MLFlow Adapter to instrument Hamilton code to log to MLFlow

Pairs well with Hamilton UI

```
dr = (  
    driver.Builder()  
    .with_modules(model_training_2)  
    .with_adapters(  
        MLFlowTracker(  
            experiment_name=f"hamilton-project-{project_id}",  
            run_name=dag_name,  
        ),  
        HamiltonTracker(...)  
    )  
    .with_materializers(  
        to.mlflow(  
            id="trained_model__mlflow",  
            dependencies=["trained_model"],  
            register_as="my_new_model",  
        ),  
    )  
    .build()  
)
```

Deep Dive: RAG & Hamilton: Document processing



https://github.com/DAGWorks-Inc/hamilton/tree/main/examples/LLM_Workflows/RAG_document_extract_chunk_embed

<https://blog.dagworks.io/p/rag-ingestion-and-chunking-using>

<https://blog.dagworks.io/p/retrieval-augmented-generation-reference-arch>

Agenda: Build a Document Processing Pipeline for RAG Systems

- Components
- Code
- Caveats

Agenda: Build a Document Processing Pipeline for RAG Systems

- Components
- Code
- Caveats

Leave you with:

1. High level mental model of the process
2. Some code to help you get started
3. A sense for where caveats might lie on your journey

Agenda: Build a Document Processing Pipeline for RAG Systems

- Components
- Code
- Caveats

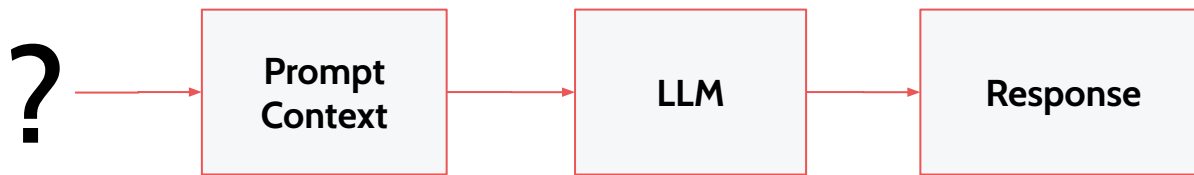


In < 30 minutes!

Leave you with:

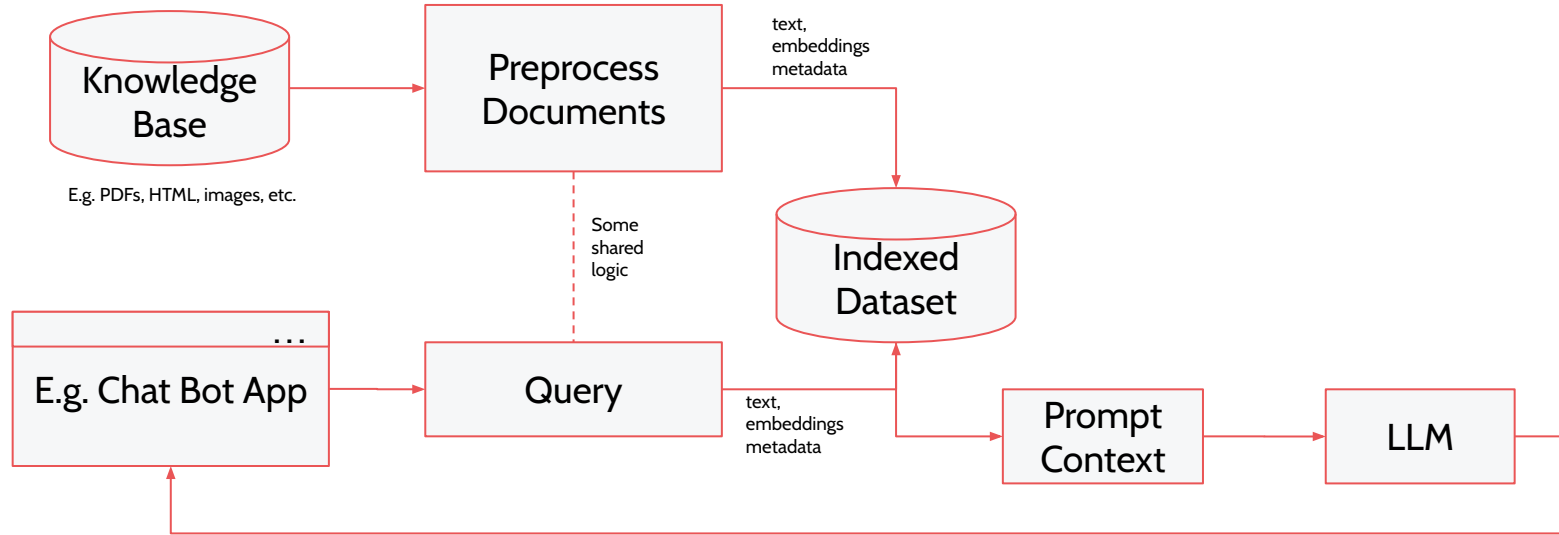
1. High level mental model of the process
2. Some code to help you get started
3. A sense for where caveats might lie on your journey

Retrieval Augmented Generation

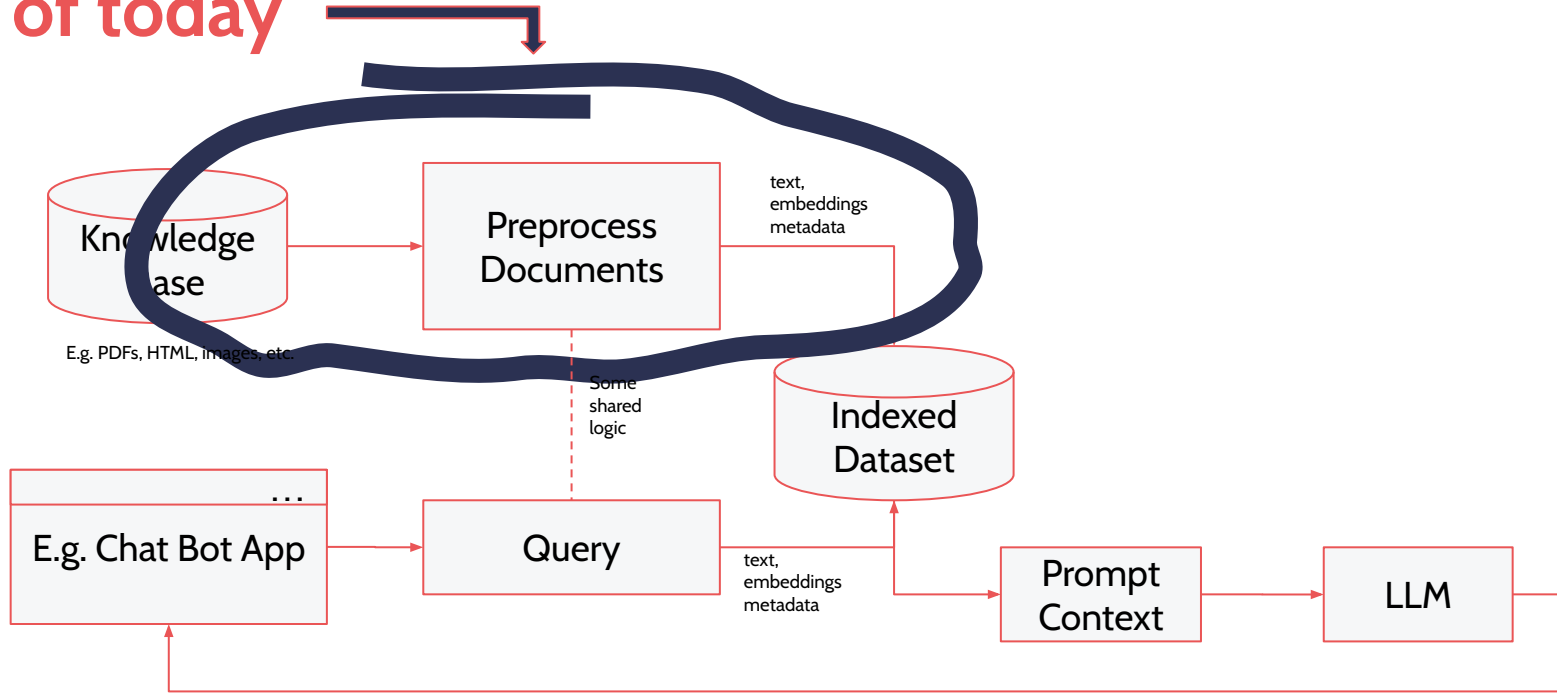


Purpose:
Build the right context
for the LLM call

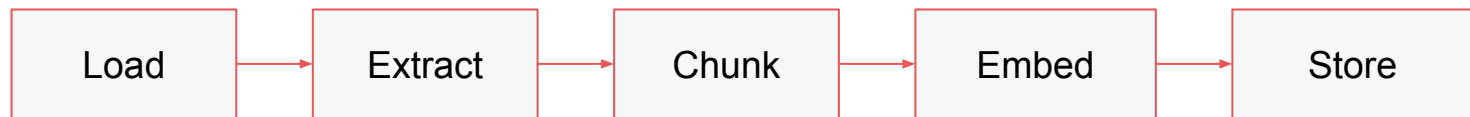
Retrieval Augmented Generation



Focus of today



Preprocessing Documents



Get Document



Extract Text



Divide Text



Create Embedding



Store & index

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

—
Dolor sit amet consectetur adipiscing elit.

—
Id semper risus in hendrerit gravida rutrum quisque non tellus.

[0.2,0.3,0.4,0.5,0.2,]

—

[0.3,0.3,0.3,0.2,0.1,]

—

[0.5,0.4,0.2,0.8,0.7,]



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

[0.2,0.3,0.4,0.5,0.2,]

—

Dolor sit amet consectetur adipiscing elit.

[0.3,0.3,0.3,0.2,0.1,]

—

Id semper risus in hendrerit gravida rutrum quisque non tellus.

[0.5,0.4,0.2,0.8,0.7,]

Note: 1 document → creates 1 or more embeddings

Preprocessing Documents

Load

Announcing the Hamilton Meetup Group. Sign up to attend events!

Hamilton

Search

USER GUIDE

Get Started

Concepts

User Guide

Integrations

Code Comparisons

COMMUNITY

Meet-ups

Slack of

REFERENCE

Decorators

Drivers

GraphAdapters

Lifecycle Adapters

ResultBuilders

I/O

Dataflows

Telemetry

EXTERNAL RESOURCES

GitHub of

tryhamilton.dev of

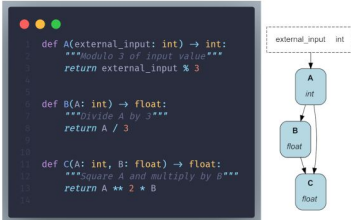
Dataflow Hub of

Blog of

Welcome to Hamilton

Join Hamilton Slack | Tweet | Downloads 200k

Hamilton is a general-purpose framework to write dataflows using regular Python functions. At the core, each function defines a transformation and its parameters indicates its dependencies. Hamilton automatically connects individual functions into a **Directed Acyclic Graph (DAG)** that can be executed, visualized, optimized, and reported on. Hamilton also comes with a UI to visualize, catalog, and monitor your dataflows.

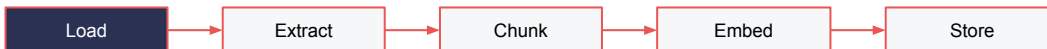


The ABC of Hamilton

Why should you use Hamilton?

Facilitate collaboration. By focusing on functions, Hamilton avoids sprawling code hierarchy and generates flat dataflows. Well-scoped functions make it easier to add features, complete code reviews, debug pipeline failures, and hand-off projects. Visualizations can be generated directly from your code to better understand and document it. Integration with the **Hamilton UI** allows you to track lineage, catalog code & artifacts, and monitor your dataflows.

Reduce development time. Hamilton dataflows are reusable across projects and context (e.g., pipeline vs. web service). The benefits of developing robust and well-tested solutions are multiplied by reusability. Off-the-shelf dataflows are available on the **Hamilton Hub**.



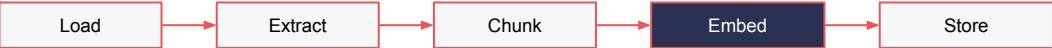
```
<doctype html>
<html class="no_js" lang="en" data-content_root="...">
<head><meta charset="utf-8">
<meta name="viewport" content="width=device-width,initial-scale=1">
<meta name="color-scheme" content="light dark"><meta name="viewport" content="width=device-width,initial-scale=1">
<link rel="index" title="Index" href="..."><link rel="search" title="Search" href="...">
<link rel="canonical" href="https://hamilton.dagworks.io/concepts/best-practices/code-organization/">

<!-- Generated with Sphinx 7.3.7 and Furo 2024.04.27.dev1 -->
<title>Code Organization - Hamilton</title>
<link rel="stylesheet" type="text/css" href="...">
<link rel="stylesheet" type="text/css" href="...">
<link rel="stylesheet" type="text/css" href="...">
```

Goal: get access to content for next steps

```
<div class="article-container">
  <a href="#" class="back-to-top muted-link">
    <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24">
      <path d="M13 20h-5.5l-5.5-1.42L12 4.12 17.92 7.92 14.12 13 12 17.92" />
    </svg>
    <span>Back to top</span>
  </a>
  <div class="content-icon-container">
    <div class="edit-this-page">
      <a href="#" class="muted-link" href="https://github.com/dagworks/hamilton/edit/main/docs/concepts/best-practices/code-organization.rst">
        <svg aria-hidden="true" viewBox="0 0 24 24" stroke-width="1.5" stroke="currentColor" fill="none" stroke-linecap="round" stroke-join="round">
          <path stroke="none" d="M0 0h24v24H0z" />
          <path d="M4 20h16" />
          <path d="M4 16h16" />
          <path d="M4 12h16" />
          <path d="M4 8h16" />
          <path d="M4 4h16" />
        </svg>
      </a>
      <span class="visually-hidden">Edit this page</span>
    </div>
    <div><div class="theme-toggle-container theme-toggle-container">
      <button class="theme-toggle">
        <div class="visually-hidden">Toggle Light / Dark / Auto color theme</div>
        <svg class="theme-icon when-auto"><use href="#svg-sun-half"></use></svg>
        <svg class="theme-icon when-dark"><use href="#svg-moon"></use></svg>
        <svg class="theme-icon when-light"><use href="#svg-sun"></use></svg>
      </button>
      <div>
        <label class="toc-overlay-icon toc-content-icon" for="..._loc">
          <div class="visually-hidden">Toggle table of contents sidebar</div>
          <div class="icon"><svg><use href="#svg-loc"></use></div>
          <div>
            <article role="main" id="uro-main-content">
              <section id="code-organization">
                <h1>Code Organization</h1><a class="headerlink" href="#code-organization" title="Link to this heading"></a><h1>
                <p>Hamilton will force you to organize your code! Here's some tips.</p>
                <p>Hamilton forces you to put your code into modules that are distinct from where you run your code.</p>
                <p>You'll soon find that a single python module does not make sense, and so you'll organically start to (very likely) put like functions with like functions, i.e. thus creating domain specific modules -&gt;. </p>
                <p>This is the way to your development advantage!</p>
                <p>Ah Sitch Fix we </p>
                <ol>
                  <li>Use modules to model team thinking, e.g. <code>features.py</code></li>
                  <li>Use modules to help isolate what you're working on.</li>
                  <li>Use modules to replace parts of your Hamilton dataflow very easily for different contexts.</li>
                </ol>
                <section id="team-thinking">
                  <h2>Team thinking</h2>
                  <p>Team thinking is a class="headerlink" href="#team-thinking" title="Link to this heading"></a><h2>
                  <p>You'll need to curate your modules. We suggest orienting this around how teams think about the business.</p>
                  <p>E.g. marketing spend features should be in the same module, or in separate modules but in the same directory/package.</p>
                  <p>This will then make it easy for people to browse the code base and discover what is available.</p>
                </section>
                <section id="help-isolate-what-you-re-working-on">
                  <h2>Help isolate what you're working on</h2>
                  <p>Grouping functions into modules then helps set the tone for what you're working on. It helps set the namespace, if you will, for that function. Thus you can have the same function name used in multiple modules, as long as only one of those modules is imported to build the DAG.</p>
                  <p>Thus modules help you create boundaries in your code base to isolate functions that you'll want to change inputs to.</p>
                </section>
                <section id="enables-you-to-replace-parts-of-your-dag-easily-for-different-contexts">
                  <h2>Enables you to replace parts of your DAG easily for different contexts</h2>
                  <p>The names you provide as inputs to functions form a defined "interface": to borrow a computer science term, so if you want to swap/change/alignment an input, having a function that would map to it defined in another module(s) provides a lot of flexibility. Rather than having a single module with all functions defined in it, separating the functions into different modules could be a productivity win.</p>
                  <p>Why? That's because when you come to tell Hamilton what functions constitute your dataflow (i.e. DAG), you'll be able to simply replace/change the module being passed. So if you want to compute inputs for certain functions differently, this is possible by including/excluding modules, when building the DAG provides a lot of flexibility that you can exploit to make your development cycle faster.</p>
                </section>
              </article>
            </div>
          </div>
        </div>
      </div>
    </div>
```

Load



Embed

Announcing the [Hamilton Meetup Group](#). Sign up to attend events!

Hamilton

Q Search

USER GUIDE
Get Started
Concepts
User Guide
Integrations
Code Comparisons

COMMUNITY
Meet-ups
Slack

REFERENCE
Decorators
Drivers
GraphAdapters
Lifecycle Adapters
ResultBuilders
I/O
Dataflows
Telemetry

EXTERNAL RESOURCES
GitHub
tryHamilton.dev
Dataflow Hub
Blog

Welcome to Hamilton

Hamilton is a general-purpose framework to write dataflows using regular Python functions. At the core, each function defines a transformation and its parameters indicates its dependencies. Hamilton automatically connects individual functions into a **Directed Acyclic Graph (DAG)** that can be executed, visualized, optimized, and reported on. Hamilton also comes with a UI to visualize, catalog, and monitor your dataflows.

```
def A(external_input: int) -> int:
    """Modulo 3 of input value"""
    return external_input % 3

def B(A: int) -> float:
    """Divide A by 3"""
    return A / 3

def C(A: int, B: float) -> float:
    """Square A and multiply by 3"""
    return A ** 2 * B
```

The ABC of Hamilton

Why should you use Hamilton?

Facilitate collaboration. By focusing on functions, Hamilton avoids sprawling code hierarchy and generates flat dataflows. Well-scoped functions make it easier to add features, complete code reviews, debug pipeline failures, and hand-off projects. Visualizations can be generated directly from your code to better understand and document it. Integration with the [Hamilton UI](#) allows you to track lineage, catalog code & artifacts, and monitor your dataflows.

Reduce development time. Hamilton dataflows are reusable across projects and context (e.g., pipeline vs. web service). The benefits of developing robust and well-tested solutions are multiplied by reusability. [Off-the-shelf dataflows are available on the Hamilton Hub](#).

[0.384,0.417,0.122,0.176]

[0.680,0.305,0.598,0.415,0.824,0.204,0.115,0.504,0.145,0.086...]

[0.552,0.671,0.893,0.128,0.331,0.654,0.005,0.538,0.312,0.877,...]

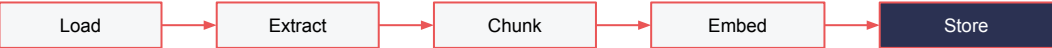
Goal:

- Capture semantic meaning
- Used for finding similar text

Decisions:

- Implementation
 - E.g. openai, etc.
- Size
- Fine tuning or not.

Embed



Store

Goal: Store for retrieval

What: text, embeddings, metadata

Decisions: where do you store it?

Announcing the [Hamilton Meetup Group](#). Sign up to attend events!

Hamilton

Search

- USER GUIDE
 - Get Started
 - Concepts
 - User Guide
 - Integrations
 - Code Comparisons
- COMMUNITY
 - Meet-ups
 - Slack
- REFERENCE
 - Decorators
 - Drivers
 - GraphAdapters
 - Lifecycle Adapters
 - ResultBuilders
 - I/O
 - Dataflows
 - Telemetry
- EXTERNAL RESOURCES
 - GitHub
 - tryhamilton.dev
 - Dataflow Hub
 - Blog

Welcome to Hamilton

Hamilton is a general-purpose framework to write dataflows using regular Python functions. At the core, each function defines a transformation and its parameters indicates its dependencies. Hamilton automatically connects individual functions into a **Directed Acyclic Graph (DAG)** that can be executed, visualized, optimized, and reported on. Hamilton also comes with a UI to visualize, catalog, and monitor your dataflows.

```

def A(external_input: int) -> int:
    """Modulo 3 of input value"""
    return external_input % 3

def B(A: int) -> float:
    """Divide A by 3"""
    return A / 3

def C(A: int, B: float) -> float:
    """Square A and multiply by B"""
    return A ** 2 * B
  
```

The ABC of Hamilton

Why should you use Hamilton?

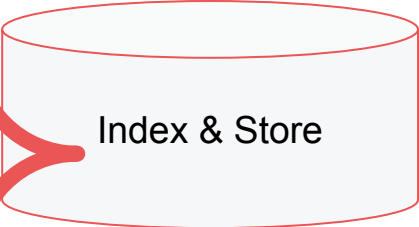
Facilitate collaboration. By focusing on functions, Hamilton avoids sprawling code hierarchy and generates flat dataflows. Well-scoped functions make it easier to add features, complete code reviews, debug pipeline failures, and hand-off projects. Visualizations can be generated directly from your code to better understand and document it. Integration with the [Hamilton UI](#) allows you to track lineage, catalog code & artifacts, and monitor your dataflows.

Reduce development time. Hamilton dataflows are reusable across projects and context (e.g., pipeline vs. web service). The benefits of developing robust and well-tested solutions are multiplied by reusability. [Off-the-shelf dataflows are available on the Hamilton Hub.](#)

[0.384,0.417,0.122,0.176,0.049,0.486,0.275,0.738,0.907,0.149,...]

[0.680,0.305,0.598,0.415,0.824,0.204,0.115,0.504,0.145,0.086...

[0.552,0.671,0.893,0.128,0.331,0.654,0.005,0.538,0.312,0.177,...]



Store

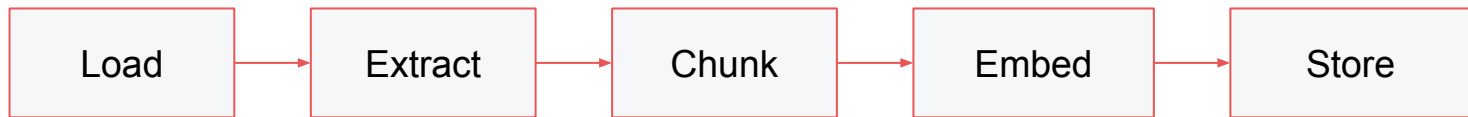
Example: Processing Hamilton's Documentation

Simple Pipeline Notebook (open in google collab)

Caveats on the road to production

Two main dimensions:

- Domain specific
- Execution related



Get Document



Extract Text



Divide Text



Create Embedding



Store & index

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

-

Dolor sit amet consectetur adipiscing elit.

-

Id semper risus in hendrerit gravida rutrum quisque non tellus.

[0.2,0.3,0.4,0.5,0.2,]

-

[0.3,0.3,0.3,0.2,0.1,]

-

[0.5,0.4,0.2,0.8,0.7,]



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

[0.2,0.3,0.4,0.5,0.2,]

-

Dolor sit amet consectetur adipiscing elit.

[0.3,0.3,0.3,0.2,0.1,]

-

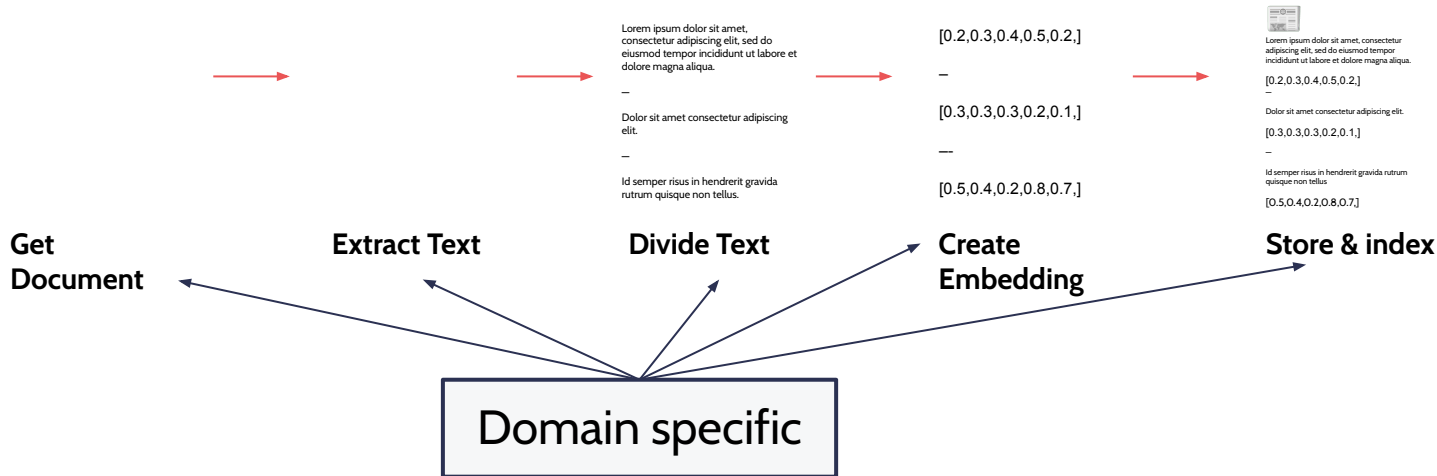
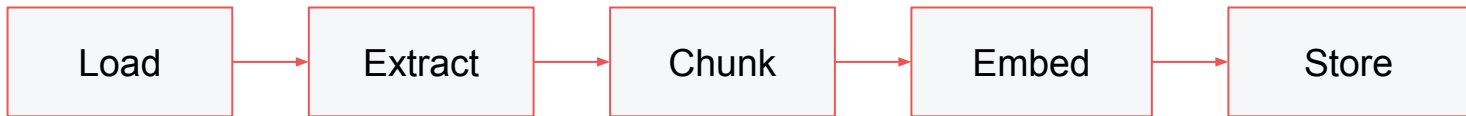
Id semper risus in hendrerit gravida rutrum quisque non tellus.

[0.5,0.4,0.2,0.8,0.7,]

Caveats on the road to production

Two main dimensions:

- Domain specific
- Execution related



Caveats on

The angle between a and b is written as $\angle(a, b)$, and is sometimes expressed in degrees. (The default angle unit is *radians*; 360° is 2π radians.) For example, $\angle(a, b) = 60^\circ$ means $\angle(a, b) = \pi/3$, i.e., $a^T b = (1/2)\|a\|\|b\|$.
The angle coincides with the usual notion of angle between vectors, when they have dimension two or three, and they are thought of as displacements from a

- ## Two main dimensions:
- Domain specific
 - Execution related

Load

58 3 Norm and distance

common point. For example, the angle between the vectors $a = (1, 2, -1)$ and $b = (2, 0, -3)$ is


$$\arccos\left(\frac{5}{\sqrt{6}\sqrt{13}}\right) = \arccos(0.5661) = 0.9690 = 55.52^\circ$$

(to 4 digits). But the definition of angle is more general; we can refer to the angle between two vectors with dimension 100.

The angle is a symmetric function of a and b : We have $\angle(a, b) = \angle(b, a)$. The angle is not affected by scaling each of the vectors by a positive scalar: We have, for any vectors a and b , and any positive numbers α and β ,

$$\angle(\alpha a, \beta b) = \angle(a, b).$$

Store



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

[0.2,0.3,0.4,0.5,0.2]

-

Dolor sit amet consectetur adipiscing elit.

[0.3,0.3,0.3,0.2,0.1]

-

Id semper risus in hendrerit gravida rutrum quisque non tellus

[0.5,0.4,0.2,0.8,0.7]

Get Document

Extract Text

Divide Text

Create Embedding

Store & index

Domain specific

Caveats on the road to production

Two main dimensions:

- Domain specific
- Execution related

E.g. Scale

E.g. rapid pace of change

Load

Extract

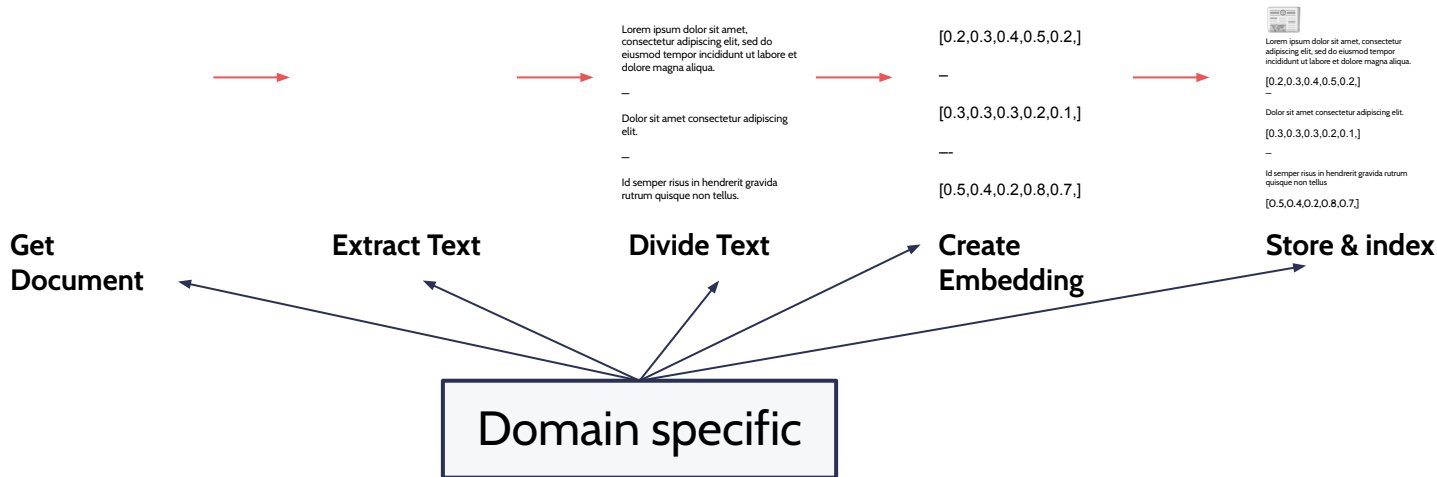
Chunk

Embed

Store

E.g. API vs GPU

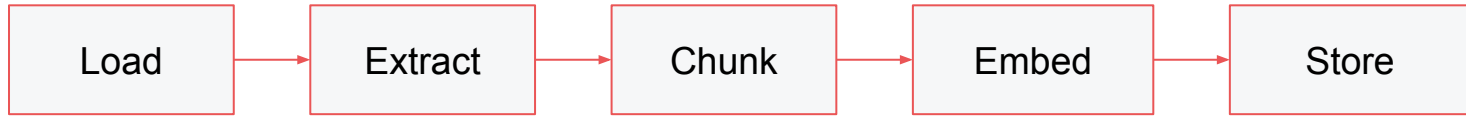
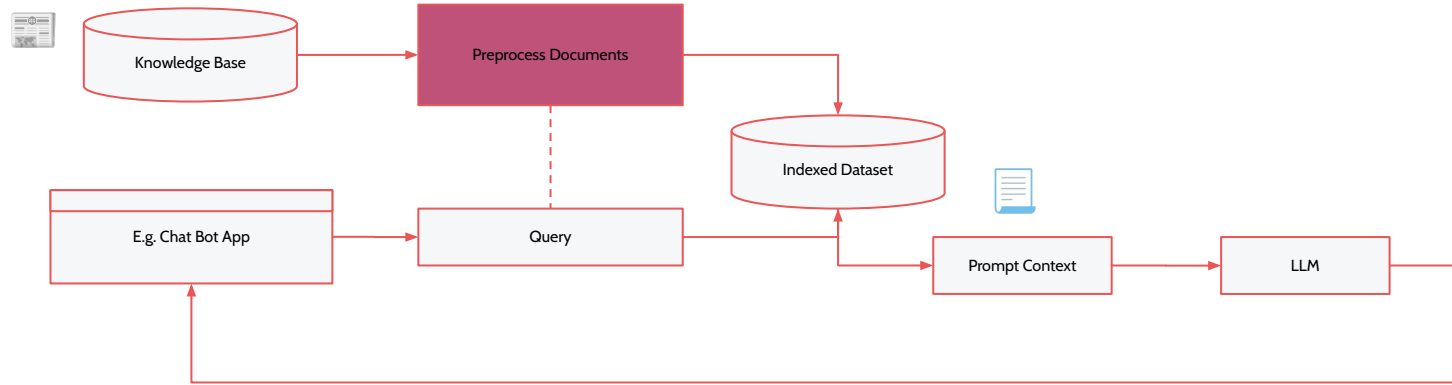
Execution



Links / “Keywords” Slide

Structuring Code	Hamilton (tryhamilton.dev) , Burr
Extracting Text	unstructured , OCR , LangChain , LlamaIndex , etc.
Chunking Text	unstructured , LangChain , LlamaIndex , etc.
Embedding Text	OpenAI , Anthropic , HuggingFace , etc.
Storage & Indexing	Files (e.g. parquet), numpy , PGVector , LanceDB , Marqo , etc
Scaling Processing	Ray , PySpark
More on embeddings	High-level article , More technical Google video

To finish





Next meet-up - August:

Speaker 1: Gilad Rubin



Open Mic.

FIN. Thanks for coming!

