Welcome

Build a Document Processing Pipeline for RAG Systems

Stefan Krawczyk

Co-creator of Hamilton & Burr
Co-founder & CEO DAGWorks Inc.

maven

⚡ Components - Code - Caveats

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

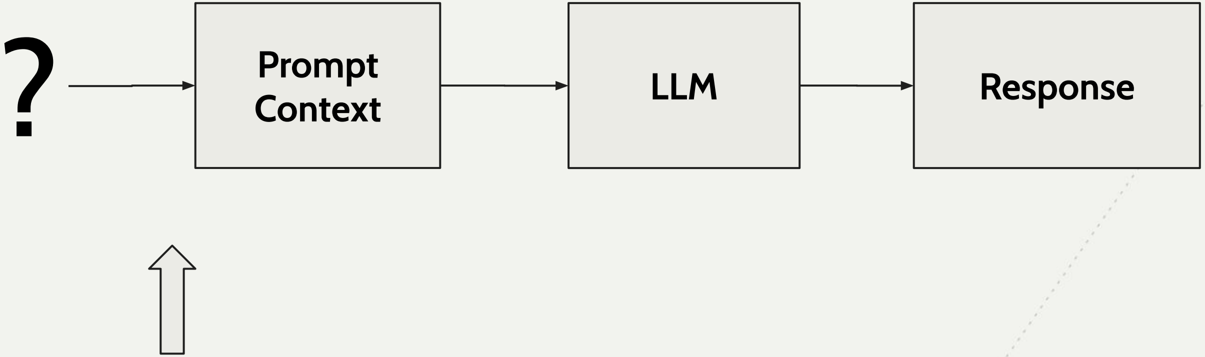
- Components
- Code
- Caveats

🌪️ In < 30 minutes!
🤯

Leave you with:

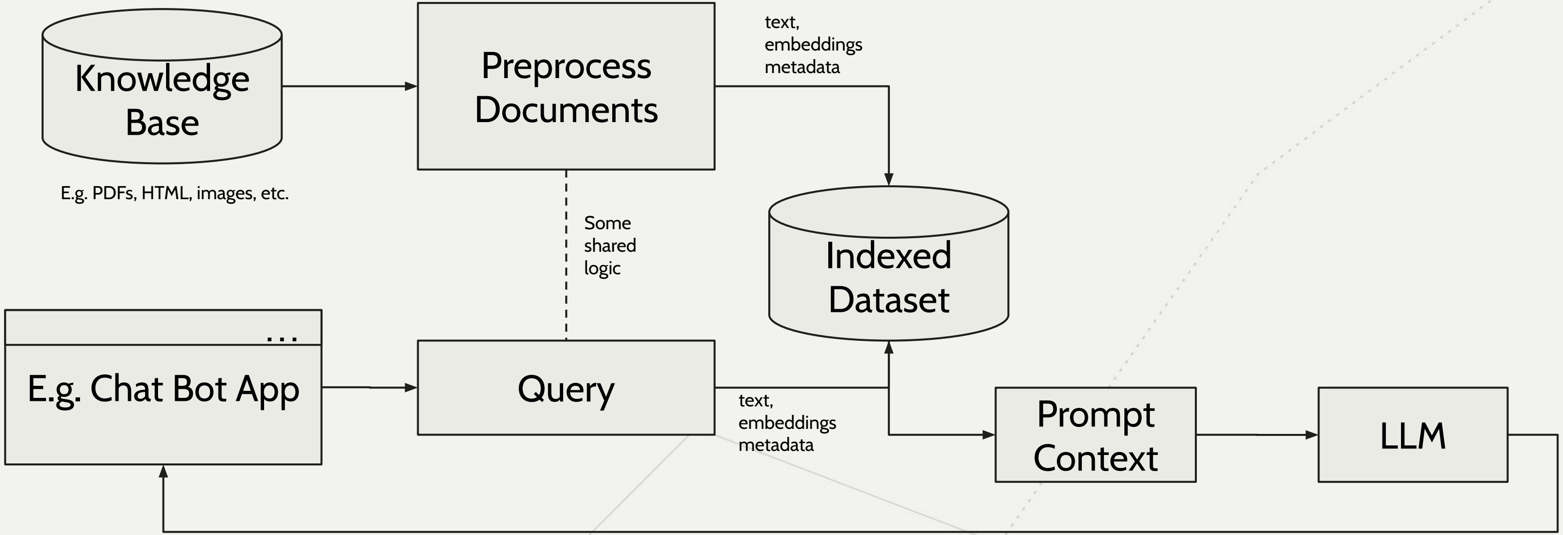
1. High level mental model of the process
2. Some code to help you get started
3. A sense for where caveats might lie on your journey

Retrieval Augmented Generation

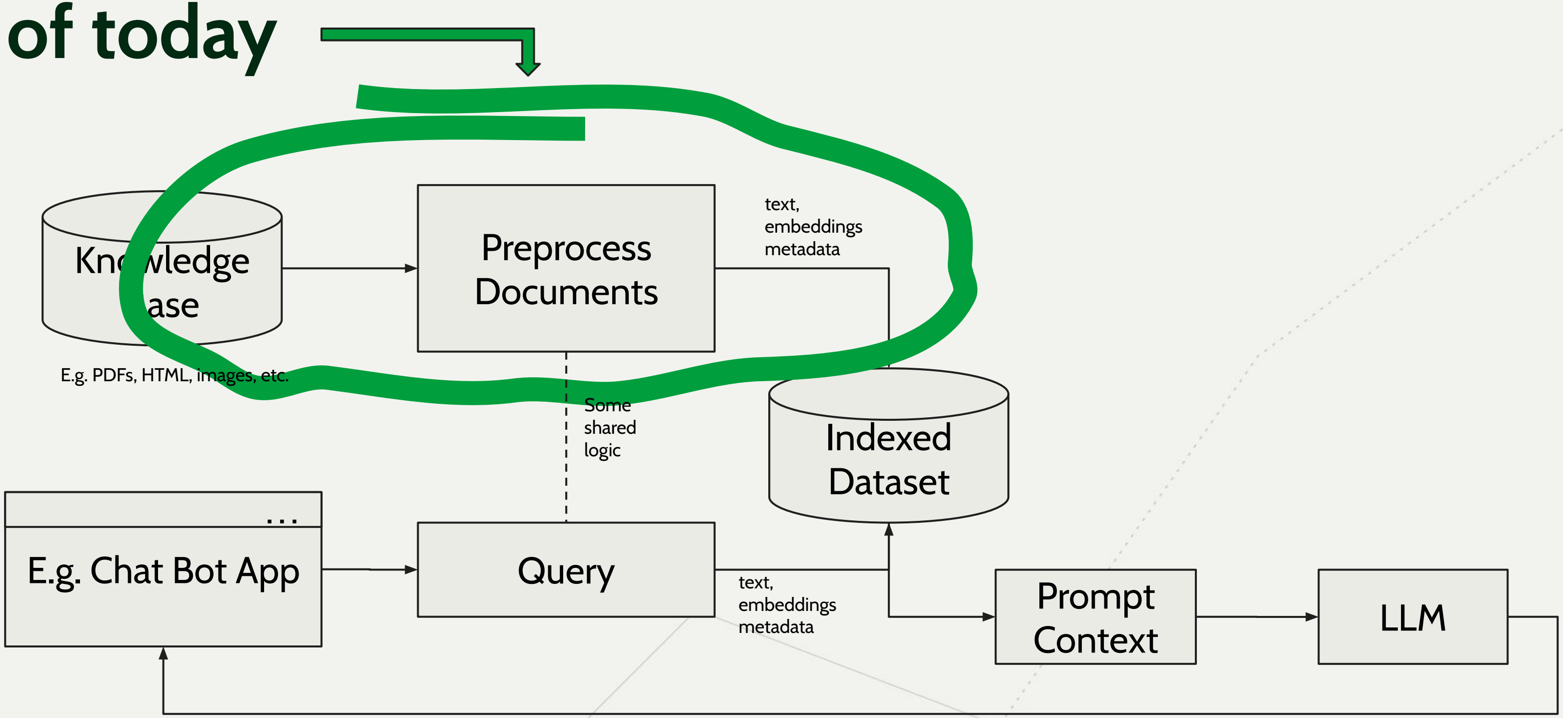


Purpose:
Build the right context for the LLM call

Retrieval Augmented Generation

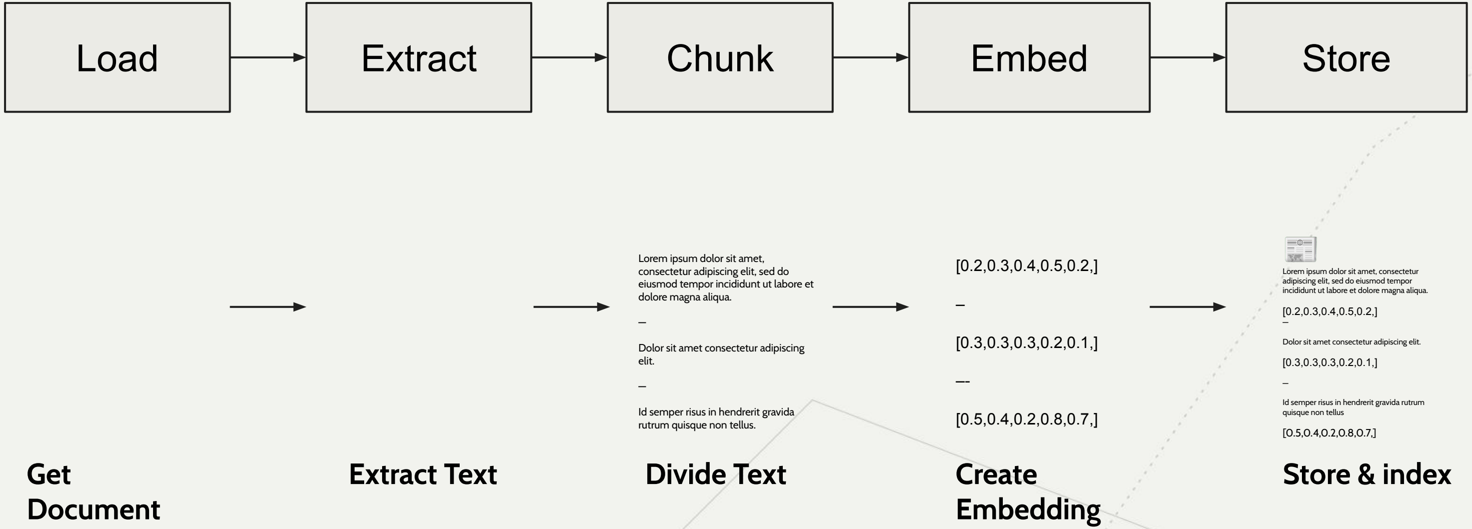


Focus of today



maven

Preprocessing Documents



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,]

Get Document

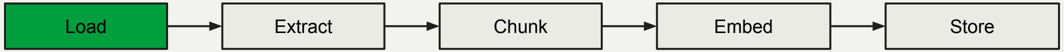
Extract Text

Divide Text

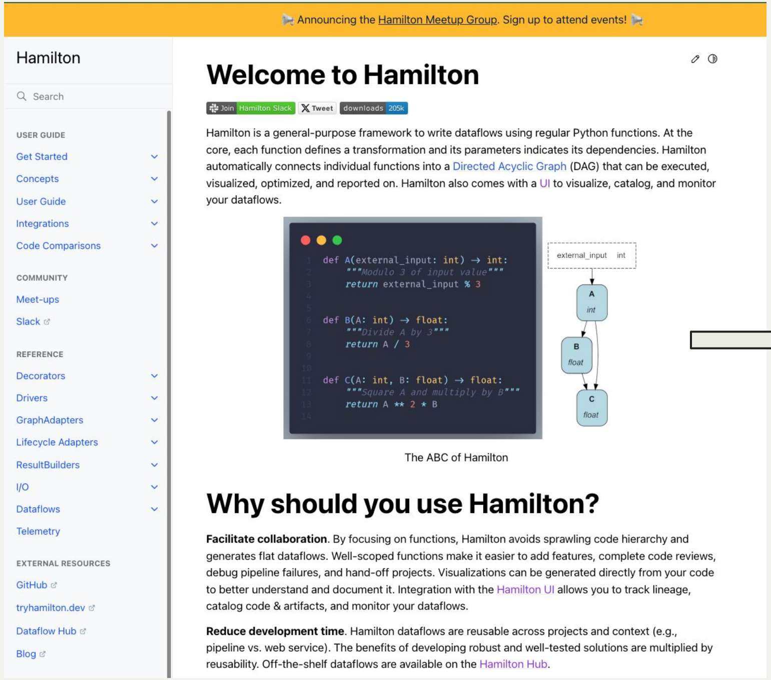
Create Embedding

Store & index

Note: 1 document → creates 1 or more embeddings



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

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.

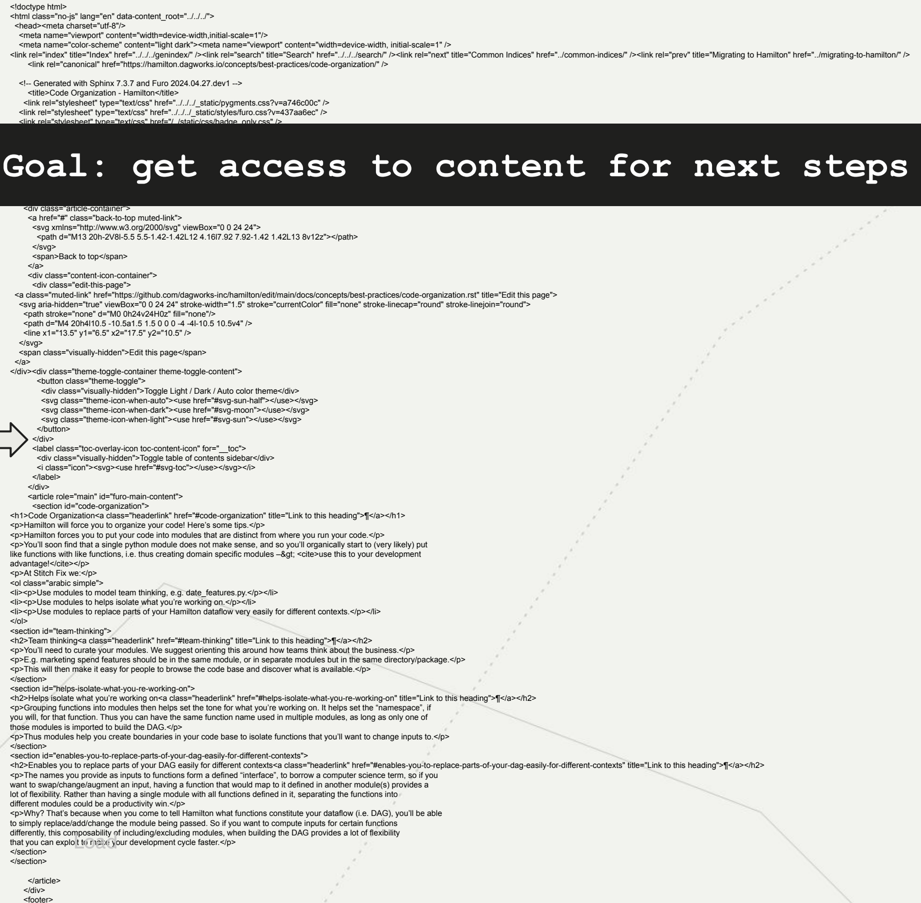
```python
def A(external_input: int) -> int:
    """Modulo 3 of input value"""
    return external_input % 3

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.

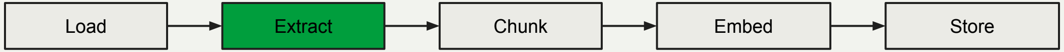


```html
<!doctype html>
<html class="no-js" lang="en" data-content_root="../../../">
  <head><meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width,initial-scale=1"/>
    <meta name="color-scheme" content="light dark"><meta name="viewport" content="width=device-width, initial-scale=1" />
<link rel="index" title="Index" href="../../../genindex/" /><link rel="search" title="Search" href="../../../search/" /><link rel="next" title="Common Indices" href="../common-indices/" /><link rel="prev" title="Migrating to Hamilton" href="../migrating-to-hamilton/" />
        <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="../../../_static/pygments.css?v=a746c00c" />
    <link rel="stylesheet" type="text/css" href="../../../_static/styles/furo.css?v=437aa6ec" />
    <link rel="stylesheet" type="text/css" href="/_static/css/badge_only.css" />
```

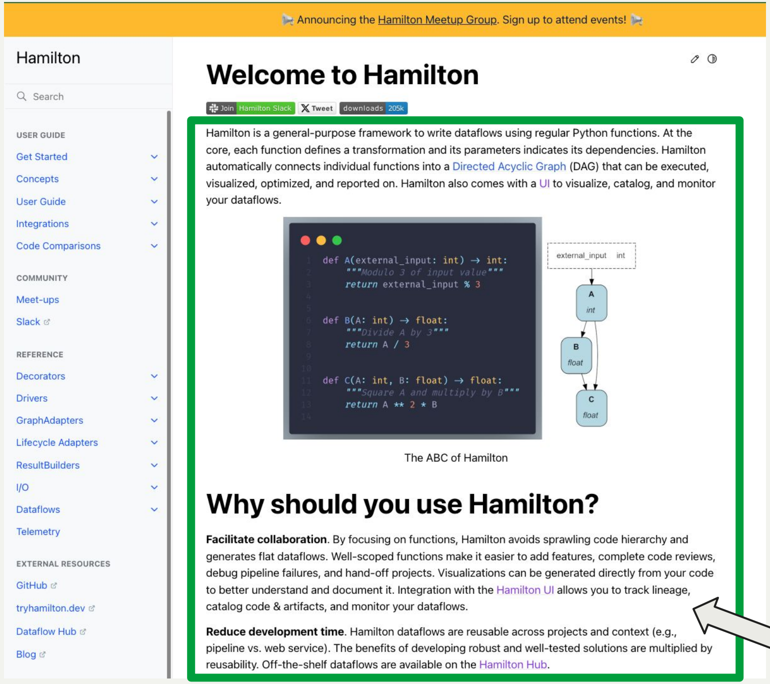
Goal: get access to content for next steps

```html
          <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-2V8l-5.5 5.5-1.42-1.42L12 4.16l7.92 7.92-1.42 1.42L13 8v12z"></path>
              </svg>
              <span>Back to top</span>
            </a>
            <div class="content-icon-container">
              <div class="edit-this-page">
  <a class="muted-link" href="https://github.com/dagworks-inc/hamilton/edit/main/docs/concepts/best-practices/code-organization.rst" title="Edit this page">
    <svg aria-hidden="true" viewBox="0 0 24 24" stroke-width="1.5" stroke="currentColor" fill="none" stroke-linecap="round" stroke-linejoin="round">
      <path stroke="none" d="M0 0h24v24H0z" fill="none"/>
      <path d="M4 20h4l10.5 -10.5a1.5 1.5 0 0 0 -4 -4l-10.5 10.5v4" />
      <line x1="13.5" y1="6.5" x2="17.5" y2="10.5" />
    </svg>
    <span class="visually-hidden">Edit this page</span>
  </a>
</div><div class="theme-toggle-container theme-toggle-content">
            <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="__toc">
            <div class="visually-hidden">Toggle table of contents sidebar</div>
            <i class="icon"><svg><use href="#svg-toc"></use></svg></i>
          </label>
        </div>
        <article role="main" id="furo-main-content">
          <section id="code-organization">
<h1>Code Organization<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; <cite>use this to your development
advantage!</cite></p>
<p>At Stitch Fix we:</p>
<ol class="arabic simple">
<li><p>Use modules to model team thinking, e.g. date_features.py.</p></li>
<li><p>Use modules to helps isolate what you’re working on.</p></li>
<li><p>Use modules to replace parts of your Hamilton dataflow very easily for different contexts.</p></li>
</ol>
<section id="team-thinking">
<h2>Team thinking<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="helps-isolate-what-you-re-working-on">
<h2>Helps isolate what you’re working on<a class="headerlink" href="#helps-isolate-what-you-re-working-on" title="Link to this heading">¶</a></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<a class="headerlink" href="#enables-you-to-replace-parts-of-your-dag-easily-for-different-contexts" title="Link to this heading">¶</a></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/augment 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/add/change the module being passed. So if you want to compute inputs for certain functions
differently, this composability of including/excluding modules, when building the DAG provides a lot of flexibility
that you can exploit to make your development cycle faster.</p>
</section>
</section>

        </article>
      </div>
      <footer>
```

maven



Extract



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.

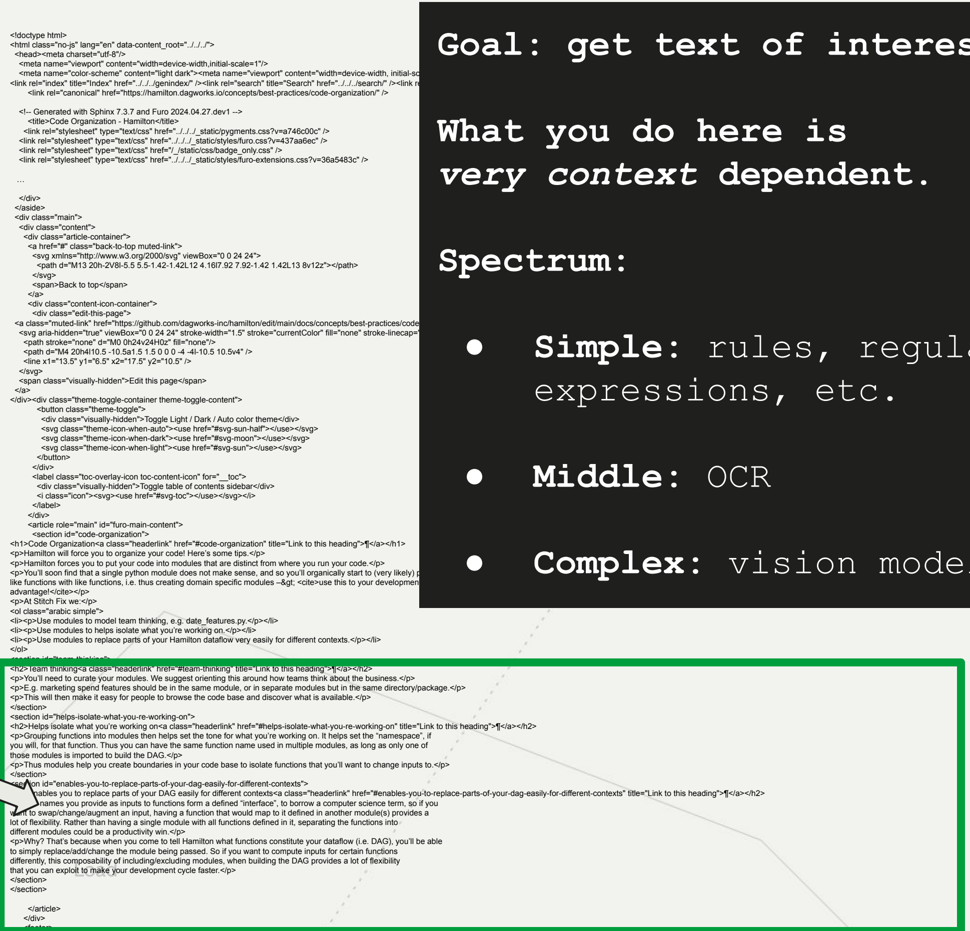
```python
def A(external_input: int) -> int:
    """Modulo 3 of input value"""
    return external_input % 3

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.



```html
<!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-sc
<link rel="index" title="Index" href="../../../genindex/" /><link rel="search" title="Search" href="../../../search/" /><link re
      <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="../../../_static/pygments.css?v=a746c00c" />
    <link rel="stylesheet" type="text/css" href="../../../_static/styles/furo.css?v=437aa6ec" />
    <link rel="stylesheet" type="text/css" href="/_/static/css/badge_only.css" />
    <link rel="stylesheet" type="text/css" href="../../../_static/styles/furo-extensions.css?v=36a5483c" />

   ...

    </div>
  </aside>
  <div class="main">
    <div class="content">
     <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-2V8l-5.5 5.5-1.42-1.42L12 4.16l7.92 7.92-1.42 1.42L13 8v12z"></path>
       </svg>
       <span>Back to top</span>
      </a>
      <div class="content-icon-container">
       <div class="edit-this-page">
  <a class="muted-link" href="https://github.com/dagworks-inc/hamilton/edit/main/docs/concepts/best-practices/code
    <svg aria-hidden="true" viewBox="0 0 24 24" stroke-width="1.5" stroke="currentColor" fill="none" stroke-linecap="
     <path stroke="none" d="M0 0h24v24H0z" fill="none"/>
     <path d="M4 20h4l10.5 -10.5a1.5 1.5 0 0 0 -4 -4l-10.5 10.5v4" />
     <line x1="13.5" y1="6.5" x2="17.5" y2="10.5" />
    </svg>
    <span class="visually-hidden">Edit this page</span>
  </a>
</div><div class="theme-toggle-container theme-toggle-content">
         <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="__toc">
         <div class="visually-hidden">Toggle table of contents sidebar</div>
         <i class="icon"><svg><use href="#svg-toc"></use></svg></i>
        </label>
      </div>
      <article role="main" id="furo-main-content">
        <section id="code-organization">
<h1>Code Organization<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) p
like functions with like functions, i.e. thus creating domain specific modules –&gt; <cite>use this to your developmen
advantage!</cite></p>
<p>At Stitch Fix we:</p>
<ol class="arabic simple">
<li><p>Use modules to model team thinking, e.g. date_features.py.</p></li>
<li><p>Use modules to helps isolate what you’re working on.</p></li>
<li><p>Use modules to replace parts of your Hamilton dataflow very easily for different contexts.</p></li>
</ol>
<section id="team-thinking">
<h2>Team thinking<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="helps-isolate-what-you-re-working-on">
<h2>Helps isolate what you’re working on<a class="headerlink" href="#helps-isolate-what-you-re-working-on" title="Link to this heading">¶</a></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<a class="headerlink" href="#enables-you-to-replace-parts-of-your-dag-easily-for-different-contexts" title="Link to this heading">¶</a></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/augment 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/add/change the module being passed. So if you want to compute inputs for certain functions
differently, this composability of including/excluding modules, when building the DAG provides a lot of flexibility
that you can exploit to make your development cycle faster.</p>
</section>
</section>

    </article>
   </div>
   <footer>
```

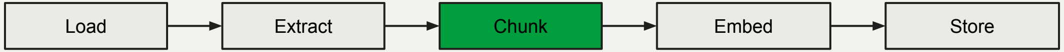
Goal: get text of interest

What you do here is *very context* dependent.

Spectrum:

- **Simple:** rules, regular expressions, etc.
- **Middle:** OCR
- **Complex:** vision models

maven



Load → Extract → Chunk → Embed → Store

Chunk

Goal:

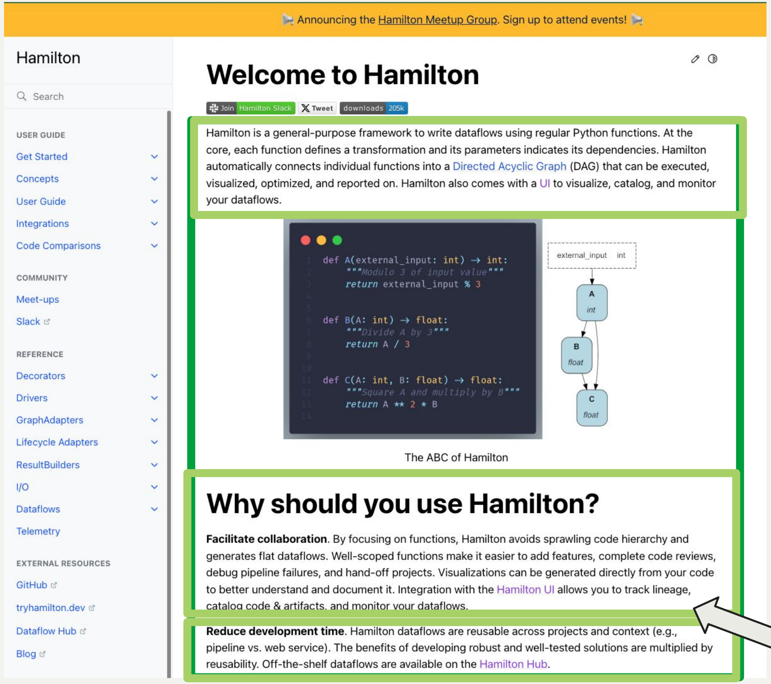
- Segment for your use case

Decisions:

- How big are the chunks
- Is there overlap

You'll use this:

- to help find & build context for the LLM call.



📢 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

Join Hamilton Slack | Tweet | downloads 205k

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.

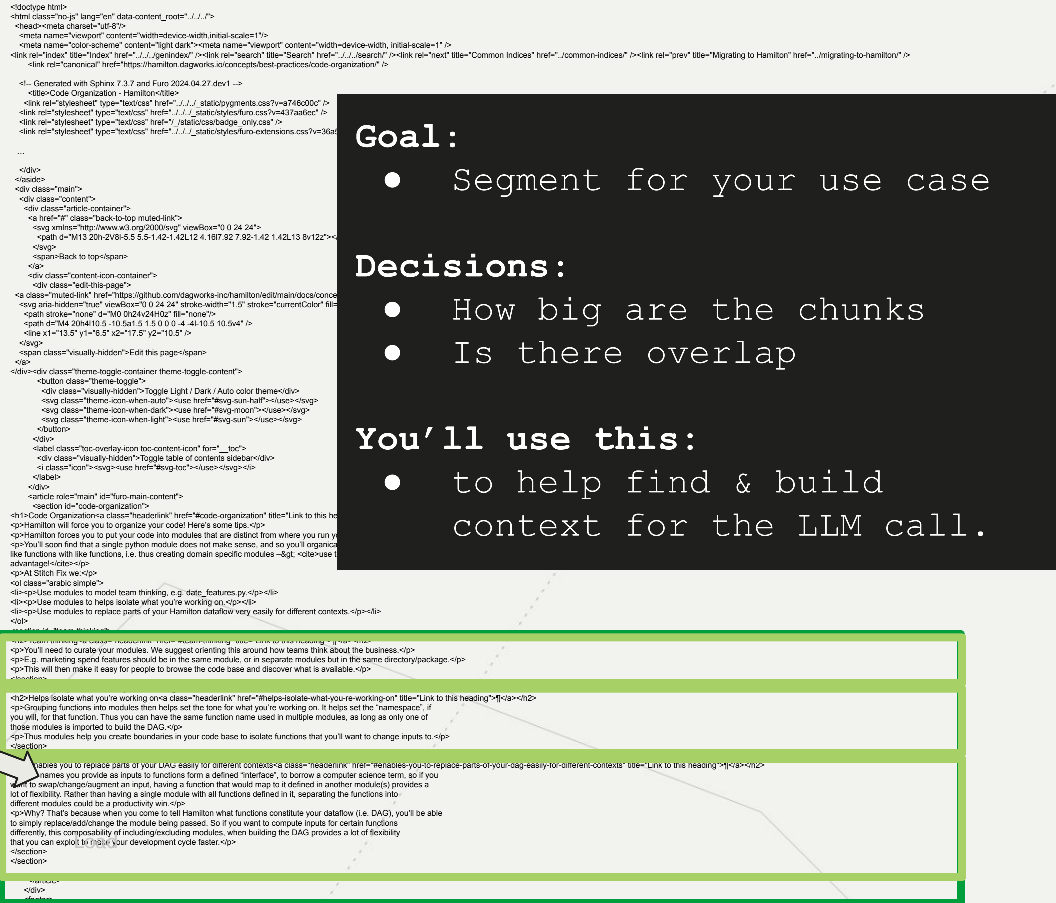
```python
def A(external_input: int) -> int:
    """Modulo 3 of input value"""
    return external_input % 3

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?

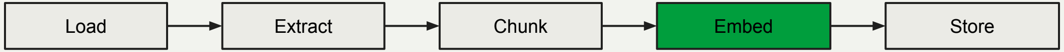
Facilitate collaboration. By focusing on functions, Hamilton avoids sprawling code hierarchy and generates flat dataflows. Well-scoped functions make it easier to add features, complete code reviews, debug pipeline failures, and hand-off projects. Visualizations can be generated directly from your code 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.



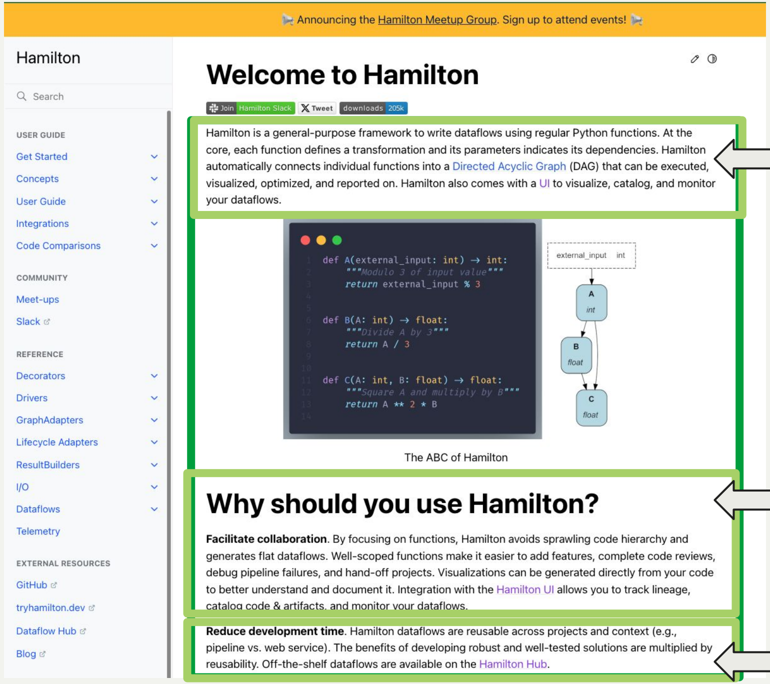
```html
<!doctype html>
<html class="no-js" lang="en" data-content_root="../../../">
 <head><meta charset="utf-8"/>
  <meta name="viewport" content="width=device-width,initial-scale=1"/>
  <meta name="color-scheme" content="light dark"><meta name="viewport" content="width=device-width, initial-scale=1" />
<link rel="index" title="Index" href="../../../genindex/" /><link rel="search" title="Search" href="../../../search/" /><link rel="next" title="Common Indices" href="../common-indices/" /><link rel="prev" title="Migrating to Hamilton" href="../migrating-to-hamilton/" />
    <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="../../../_static/pygments.css?v=a746c00c" />
  <link rel="stylesheet" type="text/css" href="../../../_static/styles/furo.css?v=437aa6ec" />
  <link rel="stylesheet" type="text/css" href="/_/static/css/badge_only.css" />
  <link rel="stylesheet" type="text/css" href="../../../_static/styles/furo-extensions.css?v=36a5

  ...

  </div>
 </aside>
 <div class="main">
  <div class="content">
   <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-2V8l-5.5 5.5-1.42-1.42L12 4.16l7.92 7.92-1.42 1.42L13 8v12z"></
     </svg>
     <span>Back to top</span>
    </a>
    <div class="content-icon-container">
     <div class="edit-this-page">
 <a class="muted-link" href="https://github.com/dagworks-inc/hamilton/edit/main/docs/conce
  <svg aria-hidden="true" viewBox="0 0 24 24" stroke-width="1.5" stroke="currentColor" fill=
   <path stroke="none" d="M0 0h24v24H0z" fill="none"/>
   <path d="M4 20h4l10.5 -10.5a1.5 1.5 0 0 0 -4 -4l-10.5 10.5v4" />
   <line x1="13.5" y1="6.5" x2="17.5" y2="10.5" />
  </svg>
  <span class="visually-hidden">Edit this page</span>
 </a>
</div><div class="theme-toggle-container theme-toggle-content">
      <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="__toc">
      <div class="visually-hidden">Toggle table of contents sidebar</div>
      <i class="icon"><svg><use href="#svg-toc"></use></svg></i>
     </label>
    </div>
    <article role="main" id="furo-main-content">
     <section id="code-organization">
<h1>Code Organization<a class="headerlink" href="#code-organization" title="Link to this he
<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 yo
<p>You'll soon find that a single python module does not make sense, and so you'll organica
like functions with like functions, i.e. thus creating domain specific modules –&gt; <cite>use t
advantage!</cite></p>
<p>At Stitch Fix we:</p>
<ol class="arabic simple">
<li><p>Use modules to model team thinking, e.g. date_features.py.</p></li>
<li><p>Use modules to helps isolate what you're working on.</p></li>
<li><p>Use modules to replace parts of your Hamilton dataflow very easily for different contexts.</p></li>
</ol>
<section id="team-thinking">
<h2>Team thinking<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>

<h2>Helps isolate what you're working on<a class="headerlink" href="#helps-isolate-what-you-re-working-on" title="Link to this heading">¶</a></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>

   nables you to replace parts of your DAG easily for different contexts<a class="headerlink" href="#enables-you-to-replace-parts-of-your-dag-easily-for-different-contexts" title="Link to this heading">¶</a></h2>
   names you provide as inputs to functions form a defined "interface", to borrow a computer science term, so if you
want to swap/change/augment 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/add/change the module being passed. So if you want to compute inputs for certain functions
differently, this composability of including/excluding modules, when building the DAG provides a lot of flexibility
that you can exploit to make your development cycle faster.</p>
</section>
</section>
     </article>
    </div>
    <footer>
```

maven



Load → Extract → Chunk → Embed → Store

Embed



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.

[0.384,0.417,0.122,0.17

[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.

Load

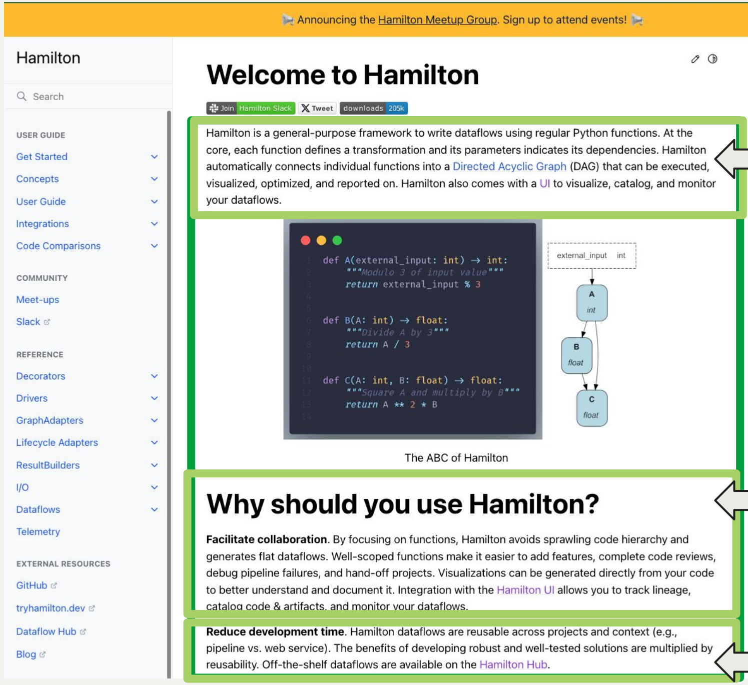
Load → Extract → Chunk → Embed → Store

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

Join Hamilton Slack | Tweet | downloads 205k

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
```

external_input int

A int

B float

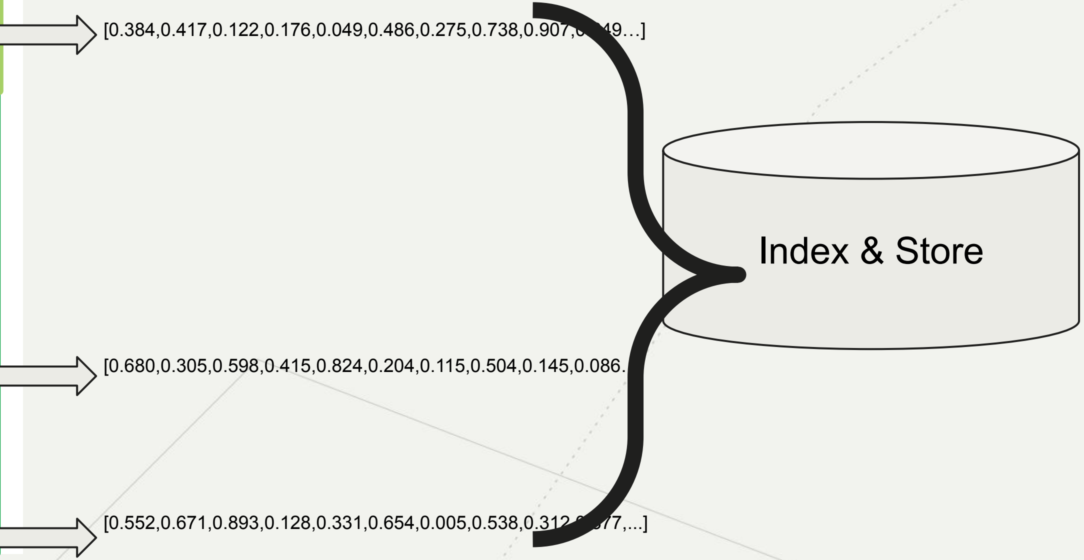
C float

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.



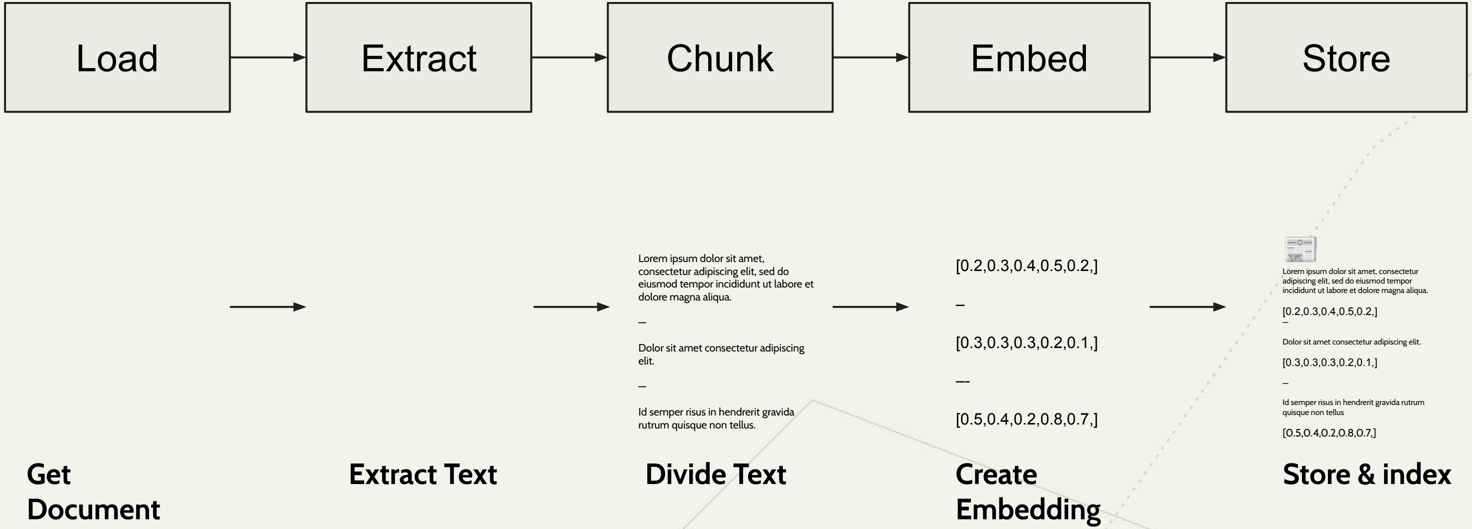
Example: Processing Hamilton’s Documentation

Simple Pipeline Notebook (open in google collab)

Caveats on the road to production

Two main dimensions:

- Domain specific
- Execution related



Load → Extract → Chunk → Embed → Store

Get Document

Extract Text

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.

Divide Text

[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,]

Create Embedding

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

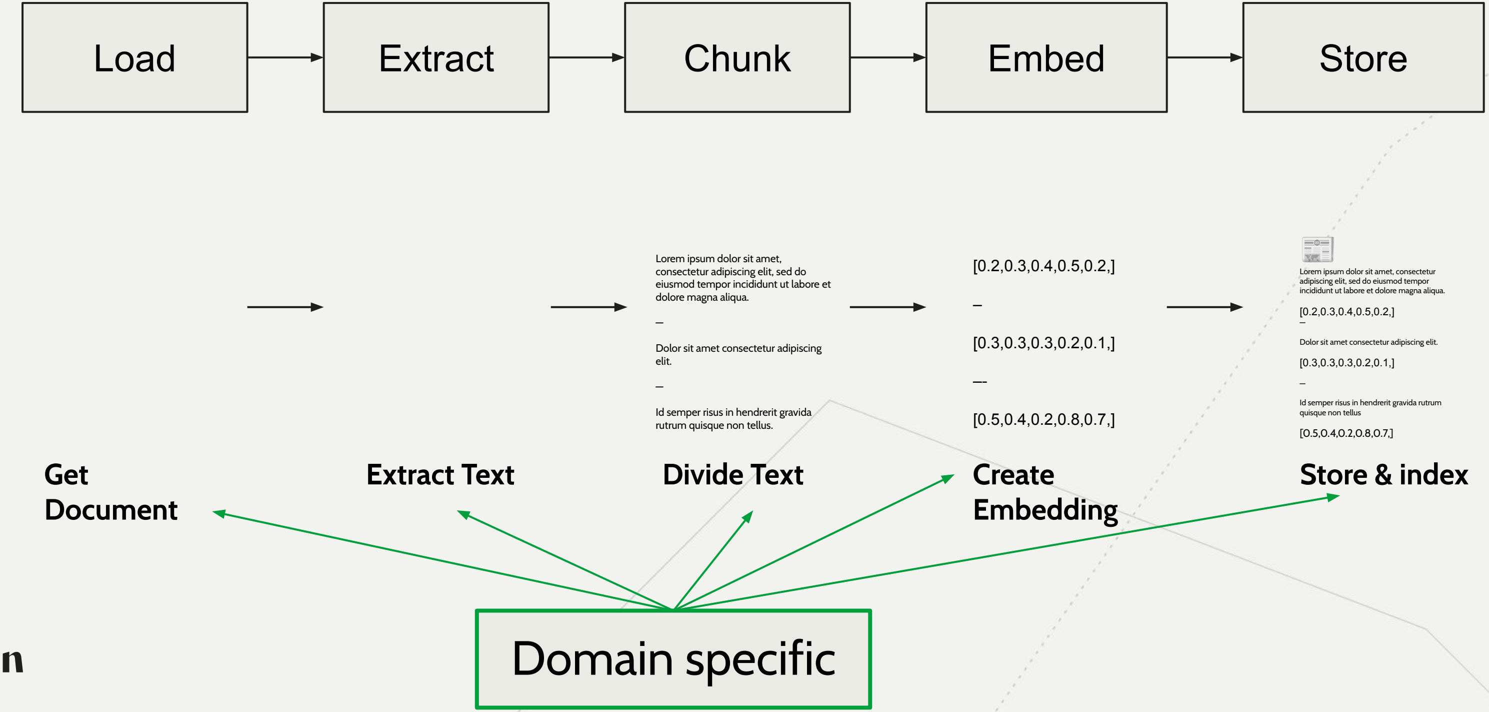
[0.5,0.4,0.2,0.8,0.7,]

Store & index

Caveats on the road to production

Two main dimensions:

- Domain specific
- Execution related



Load → Extract → Chunk → Embed → Store

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,]

Get Document

Extract Text

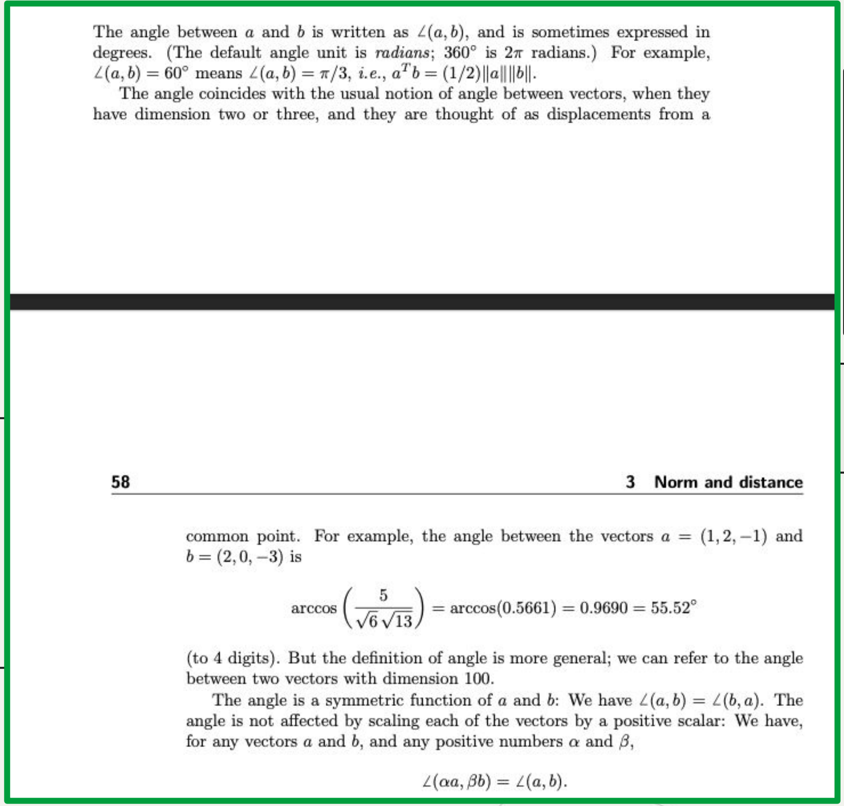
Divide Text

Create Embedding

Store & index

Domain specific

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, $\angle(a, b) = 60°$ 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

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

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

(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,

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

Two main dimensions:

- Domain specific
- Execution related

Load

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,]

–

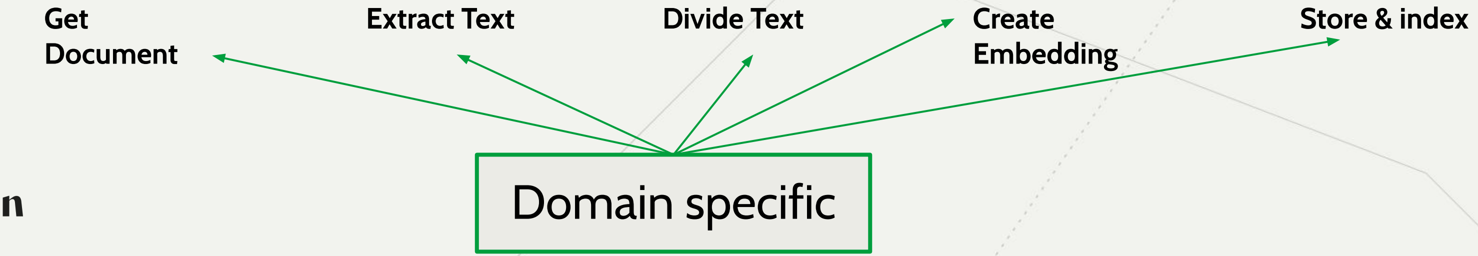
Dolor sit amet consectetur adipiscing elit.

[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,]

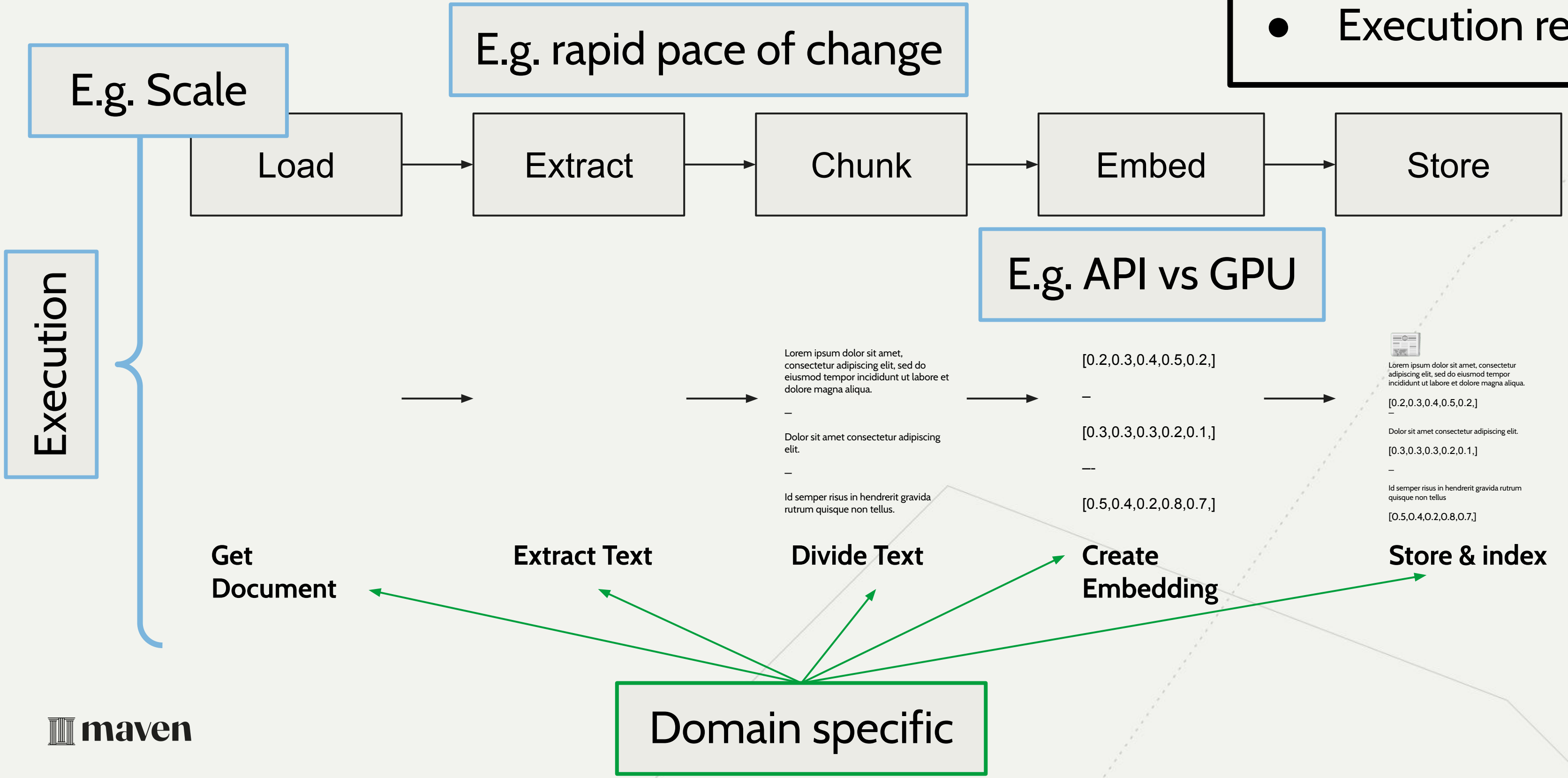


maven

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

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,]

Get Document

Extract Text

Divide Text

Create Embedding

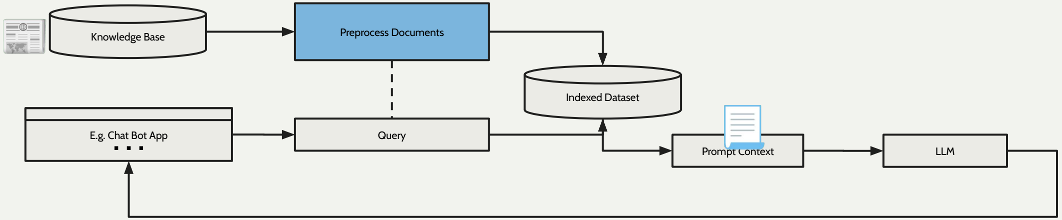
Store & index

Domain specific

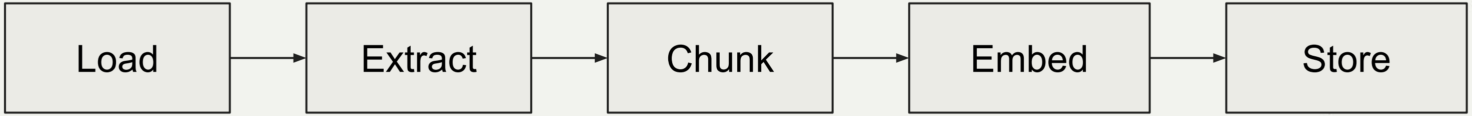
maven

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



Thanks for listening!

Would you like a course on this? Please fill out https://maven.com/forms/49388e

Connect: https://www.linkedin.com/in/skrawczyk/ (mention maven)