Control Flow for Agents - Burr + Hamilton combo

Tooling dedicated to capturing interactions with LLMs as state machines.

Version: 0.3 Dated: 2024-10-21 Authored by: CM

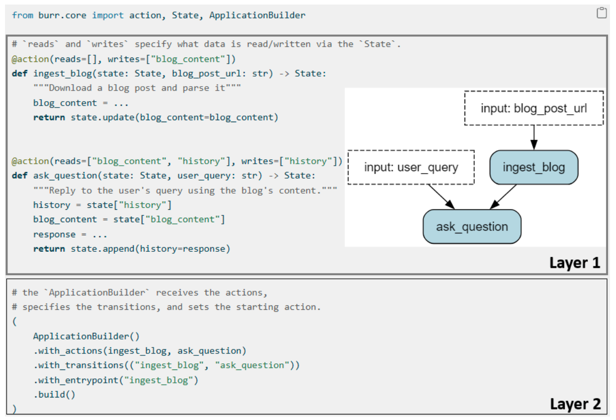
colum.mccoole@analect.com
colum-mccoole-746b946a
analect/technical-docs-hierarchy

Topic Navigation

Better Flow Controls When Developing LLM Apps

Those with greatest experience in developing LLMs articulate that common friction points with GenAI applications can include logically modeling application flow, debugging and recreating error cases, and curating data for testing/evaluation. The developers of **Burr** say these problems all got easier to think about when they modeled applications as *state machines* composed of `actions` designed for introspection. An application will hold state and make decisions off of that state. You can therefore capture your application as a state-machine, modifying the state as you go. If you do that explicitly, you get all sorts of benefits, per 1 - 5 on the right.

Many other solutions out there, including LangGraph, use graphs to model an application, since they can be easily reasoned about as connected nodes `<some-input>--> <data-transform> --> <some-output>`. Per Fig 1, Burr is a light-weight framework that wraps around python functions, making a helpful distinction around *Layer 1* actions (nodes) that are composable into *Layer 2* applications, allowing for extending or refactoring over time.



```python
from burr.core import action, State, ApplicationBuilder

# `reads` and `writes` specify what data is read/written via the `State`.
@action(reads=[], writes=["blog_content"])
def ingest_blog(state: State, blog_post_url: str) -> State:
    """Download a blog post and parse it"""
    blog_content = ...
    return state.update(blog_content=blog_content)


@action(reads=["blog_content", "history"], writes=["history"])
def ask_question(state: State, user_query: str) -> State:
    """Reply to the user's query using the blog's content."""
    history = state["history"]
    blog_content = state["blog_content"]
    response = ...
    return state.append(history=response)
```

```python
# the `ApplicationBuilder` receives the actions,
# specifies the transitions, and sets the starting action.
(
    ApplicationBuilder()
    .with_actions(ingest_blog, ask_question)
    .with_transitions(("ingest_blog", "ask_question"))
    .with_entrypoint("ingest_blog")
    .build()
)
```

Figure 1. **2-layer approach to build a maintainable system** - The graduation problem: avoid frameworks getting in the way, Nov. 2024

Developer Concerns when Evolving LLM Applications

Burr is meant to start off as an extremely lightweight tool to make building LLM applications easier. You'll find a useful 90-second intro here and an informative thread on Hacker News at HN: Burr — A Framework for building and debugging GenAI apps faster, Apr. 2024. Below are some of the out-of-the-box features. Particularly useful are its ability to capture telemetry and other meta-data pertaining to app interactions either directly with LLM end-points or LLM-enabled agentic tools.

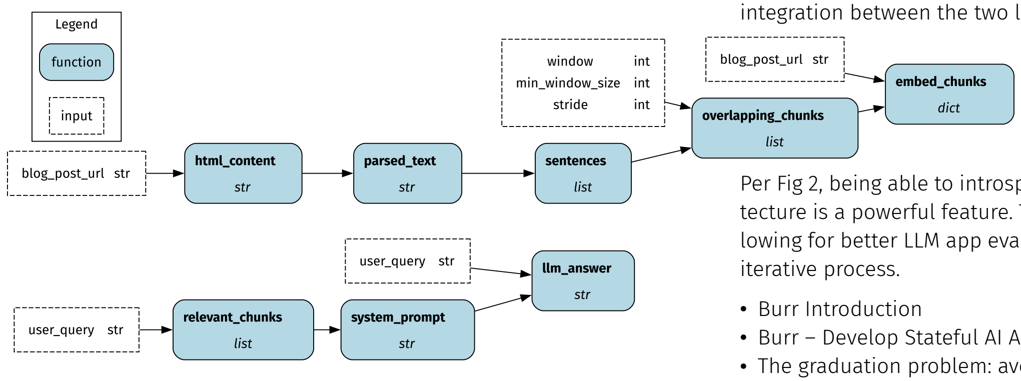
1. **Tracing/telemetry** – LLMs can be chaotic, and you need visibility into what decisions it made and how long it took to make them.
2. **State persistence** – thinking about how to save/load your application is a whole other level of infrastructure you need to worry about.
3. **Visualization/debugging** – when developing you'll want to be able to view what it is doing/did + load up the data at any point
4. **Manage interaction between users/LLM** – pause for input in certain conditions
5. **Data gathering for evaluation + test generation** – storing data run in production to use for later analysis/fine-tuning

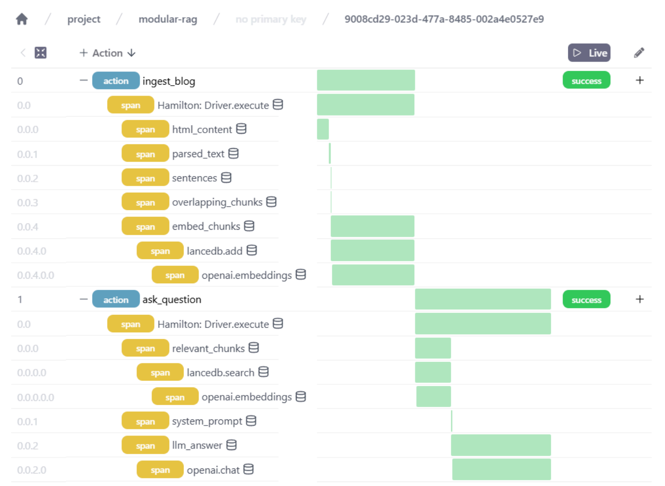
Burr / Hamilton Pairing for Improved Granularity of Flow

Burr facilitates implementing the flow chart mental model for agents. Writing an application with Burr is a matter of defining action and state. It makes developing reliable RAG and LLM agents with predictable and debuggable behaviors much simpler.

A sister project from DAGWorks called **Hamilton** brings modularity and structure to any Python application moving data: ETL pipelines, ML workflows, LLM applications, RAG systems, BI dashboards, and the Hamilton UI allows you to automatically visualize, catalog, and monitor execution.

Figure 3: Structure Data Transformations as DAGs, using Hamilton





project / modular-rag / no primary key / 9008cd29-023d-477a-8485-002a4e0527e9
+ Action
Live

0	action	ingest_blog	success
0.0	span	Hamilton: Driver.execute	
0.0.0	span	html_content	
0.0.1	span	parsed_text	
0.0.2	span	sentences	
0.0.3	span	overlapping_chunks	
0.0.4	span	embed_chunks	
0.0.4.0	span	lancedb.add	
0.0.4.0.0	span	openai.embeddings	
1	action	ask_question	success
0.0	span	Hamilton: Driver.execute	
0.0.0	span	relevant_chunks	
0.0.0.0	span	lancedb.search	
0.0.0.0.0	span	openai.embeddings	
0.0.1	span	system_prompt	
0.0.2	span	llm_answer	
0.0.2.0	span	openai.chat	

Figure 2. **Burr UI* - Introspect Application Logic and LLM Interface while Capturing Telemetry for Evaluation

Decorating Existing Code for Free Visualistion / Telemetry

From Fig 1, an app to ingest content from a URL and enable a user to ask questions of it, starts with two 'actions', `ingest_blog` and `ask_question`. This works, but an app creator may seek greater granularity and so Fig 3 shows a refactor of actions using Hamilton to structure data transformations as directed acyclic graphs (DAGs). Hamilton uses the function and parameter names to infer the dependencies between functions and the graph structure. See V3: Keeping your code modular. Burr was originally built as a harness to handle state between executions of Hamilton DAGs and so integration between the two libraries is tight.

Per Fig 2, being able to introspect the flows between the app's node architecture is a powerful feature. These interactions gets persisted on disk, allowing for better LLM app evaluation, thereby improving the development iterative process.

- Burr Introduction
- Burr – Develop Stateful AI Applications, Mar. 2024
- The graduation problem: avoid frameworks getting in the way, Nov. 2024
- Ep. #19, Auditability Matters with Stefan Krawczyk of DAGWorks