AthenaInstruments

Accelerating the practical application of generative AI in varied workflows

Tim Child

2025-05-02

Table of contents

Problems

Here's a short overview of some of the problems that currently exist in the generative AI space, that AthenaInstruments aims to solve:

- The best generative AI providers change on a weekly basis – It's not optimal to be tied to a single provider given the pace of innovation
- The best models are different for different tasks – Even if you're tied to the best overall provider, they may not have the best models for every task.

- The model doesn't know your workflows – You know how a certain task should be carried out, but the model is going to make up a new method every time it's asked.

Overview

AthenaInstruments is a platform that tackles these problems without re-inventing the wheel.

I want to provide users an easy way to work at the cutting edge of generative AI without needing to be an expert in the field, or to re-build their workflows on a weekly basis.

At the same time, I want to let the user have access to the customizations they are most interested in.

The general idea is to build a platform that works well for a range of tasks right out of the box, seamlessly taking advantage of the best models from any provider on any given day. And then to expose the tools that allow us to provide this service in a way that an interested user can tweak an existing service to their needs, or build a whole new service from scratch if they so desire.

Use cases

Typical user

For the typical user, initially, the platform will behave much like the products offered by the large generative AI providers including:

- `OpenAI`
- `Google`
- `xAI`
- `Anthropic`
- etc.

It will be a chat interface (with multimodal inputs), that produces a text (or speech) output, potentially with images or other media depending on the task. They won't need to do anything different from how they'd interact with any of the other services.

The differences will all be behind the scenes:

- The assistant will seamlessly chose an optimized workflow for the task at hand[1]

[1] I expect several of the offerings from the big players take a similar approach, but we have the advantage of having access to all of them.

    - If all we need to do is respond to a simple “hello”, it could pick a fast a friendly model – maybe this week the best for that is Claude-Haiku – and it’s response can be returned directly, and quickly.
    - If the task is more complex – say a user wants to generate a report on a topic – it seamlessly hands off to a more sophisticated workflow that can produce a fully formatted PDF report.
- Before producing a report, and even before calling any LLMs, we can first draw on a range of other services to gather some preliminary data quickly.
    - E.g., Loading recent news articles, searching the web, checking if the user has previously discussed this topic or given preferences as to how they like their reports formatted, etc.
    - This can all be done in parallel, and typically takes less than 1 second.
- Now, we are ready to get a first draft of the report. We call on the best models from each of OpenAI, Google, and Anthropic simultaneously to get their best drafts.
    - Again this happens in parallel, and we can – for example – decide that if we’ve already got two answers back, that we don’t need to wait for the third.
- Then, we ask another agent that is specifically set up (and trained) to generate report content based on those answers and the most relevant data we’ve gathered so far.
    - At this point, the agent could also decide that the drafts aren’t good enough yet, and first demand a better draft from the LLMs while giving it’s feedback to them, or it could present a friendly prompt to the user to provide some key missing data.
- Finally, a custom service converts the report into a PDF and attaches it to the message that is sent back to the user along with a summary of the steps taken and the key findings.

Power user

For a power user, the platform will provide a friendly interface that exposes all the crucial knobs and buttons that facilitate building a custom workflow.

For example, a project manager that would like morning coffee updates on the progress of a project, but doesn’t want to bog down their team with the task of coming together to produce a one page summary every day. They know that what they’d really like can be boiled down to a process like:

- Gather data for project X and employees Anna, Bob, and Charlie from:
    - The latest commits in GitHub
    - The latest issues opened/closed in Jira
    - The latest messages in the project Slack channel

- Also look at the last weeks worth of morning summaries (and whether any feedback on those was given)
- Generate a new morning summary for today

They can spend some extra time at the beginning of the project to set this custom workflow up. The process could look something like this:

- The PM sees that there are already agents that are good at getting data from each of GitHub, Jira, and Slack, so adds those to the main agent, and then in plain english says: “I want you to call the GitHub, Jira, and Slack agents simultaneously for project X over the last 48 hours for employees Anna, Bob, and Charlie and see what’s new.”
- There’s also a tool for searching previous messages, so they add that option, and continue descrbing what they want: “You should also get the last weeks worth of morning summaries and see if I gave you any feedback on those that you should take into account for this report.”
- There’s already a report generating agent, so they add that too and say: “Once you have all that data, generate a new morning summary for today. I want to know what progress was made yesterday, what’s planned for today, whether anyone is blocked, or likely to be blocked soon, and any other relevant information.”

That’s enough to get started, and if the summary isn’t quite right on the first day, they can just give some feedback and get an updated version right then, and the next day the custom agent will already know to take that feedback into account.

Even 3 weeks into the project, when that initial feedback won’t be directly collected in the data gathering step for that days morning report, since the agent will still see the last weeks worth of morning summaries, it’s going to produce a report that is similar in form and quality to those, so it’s probably going to still be following that earlier advice. And, if what you really want to see changes over time, it will change with you, even without needing to update the workflow.

Tech stack

Now I’ll break down the various parts of the system:

- Give an overview of the technologies used
- Discuss the choices made
- Address the scalability of each part

Web application

I use `reflex`, a new python framework that facilitates rapid development of performant Next.js websites with a python `FastAPI` backend and websocket communication between them. This largely achieved in pure python, although tweaks can be made easily with added javascript.

This provides a convenient environment to utilize the versatility of python (particularly for new AI libraries) without sacrificing a performant front-end, and without requiring a separate development team. This allows for a faster development cycle with more cohesion at a lower cost.

Backend

The core of the backend is written in `python`, with an architecture that allows for any bottlenecks to be re-written in a more performant language (`go` or `rust`) as needed.

For the main workflows, the large language models are the main bottleneck anyway, so python being a slow language is not a significant issue.

The larger issue with `python` is codebase maintainability, however, that is addressed by the use of linting and static type checking.

Language models

The main language model part of the backend takes advantage of the `LangGraph` library that facilitates graph-based workflows based on Googles `Pregel` system. This allows for relatively unbounded complexity in the workflows that can be created, while efficiently parallelizing the computational steps.

A lot of the computation work here is currently offloaded to the LangGraph Platform API. They are currently offering a very good value for money service while they expand their user base. In anticipation of price hikes or scaling issues, an adapter layer has been implemented to facilitate an easy switch to a self hosted service if needed.

User Authentication

User authentication is offloaded to `Clerk` and is integrated directly into the frontend and backend allowing for a seamless user experience and a secure system.

There is no real limit on scaling here.

Persistence

Long term data is stored in a `PostgreSQL` database. Short term data uses a much faster `Redis` database that shared with the `reflex` framework for managing client sessions.

Initial scaling here can be handled by using more powerful servers. This alone should easily handle up to millions of users. (Additional scaling beyond that will require more work, but is a standard problem with common solutions.)

Third party services

Currently, in addition to the integrations that form the core of the system, there are other third party services that have been integrated to provide additional functionality. At the moment, this is limited to:

- `Tavily` for LLM friendly web searching
- `GitHub` for optimized repository data retrieval

The system is designed to be easily extensible to include a range of additional services. The current limited services are intended to provide a proof of concept for the system's ability to integrate with external services.

Services such as `Tavily` are easy to integrate with, but they are also potentially places where internal implementations could be added to further reduce costs and/or provide more optimized services.

The `GitHub` integration is a good example of a bespoke service offered in AthenaInstruments for which an equivalent is not widely available, and for which a custom implementation is necessary for optimal performance. Currently, a custom data ingestion pipeline is used to create local vectorized representations of the data (including private repositories if the user grants access). Custom tools can be provided to the LLM agents so that they can interact with this data efficiently (while ensuring no data leakage between users).

This also provides an opportunity to implement organizations or workspaces that allow data to be efficiently shared between users within the same organization.

Future integrations are planned with services such as:

- Storage:
  - `Google Drive`
  - `Dropbox`
  - `OneDrive`
- Communication:
  - `Slack` for team communication
  - `Discord` for community building
- Project Management:
  - `Jira` for project management

    - `Confluence` for documentation
    - `Notion` for note-taking
    - `Trello` for task management
- CRM and Support:
    - `Salesforce` for CRM
    - `Zendesk` for customer support
- Marketing:
    - `Mailchimp` for email marketing
    - `Hubspot` for marketing automation
- Analytics:
    - `Google Analytics` for web analytics
    - `Hotjar` for user behavior analytics

> **Note**
>
> I believe that building these integrations in-house will be crucial to optimizing the performance of the system when integrated with various LLMs. Additionally, I believe that offering a few well-integrated services will be more valuable to users than the approach that many of the larger companies are taking, where they leave it up to external developers to build integrations with their services.

Deployment

Deployment is currently handled via automated GitHub Actions workflows that build and deploy first to a staging server, and only to a production server after an automated test suite has passed.

This ensures a high level of reliability of the user-facing service, while still allowing for continuous deployment of new features, security updates, and bug fixes. It's currently possible to patch and deploy a fix in under 15 minutes with no downtime and cutting no corners on the standard security and testing protocols.

The deployed service is currently hosted on a single DigitalOcean virtual private server with docker compose managing the various services that run in tandem. This is cost-effective at this early stage, and the various services are all designed with future scaling via a kubernetes cluster deployment in mind. Although, even a simple load balancer would be sufficient for a significant scale up.

Monitoring

Live monitoring is an area that is currently lacking, however, there are several interfaces already implemented that allow for manual monitoring and management of the system. Future work to implement a more automated approach is planned, but not yet a priority.

Security

Security is a top priority, and the system has been designed with security in mind from the ground up. All network communication is encrypted end-to-end, and all persistent user data is stored encrypted at rest. Additionally, because Clerk is used for user authentication, there is no possibility to leak sensitive user data as it is never persisted in our system.

All interactions with large language model providers are anonymized and encrypted.

Still, a significant amount of work will likely be required in preparation for a security audit before obtaining certification of security compliance.