BeeAI Introduction

Presented by – Kenneth (Alex) Jenkins (B.S. CompE 2026)

Georgia Tech

The Problem

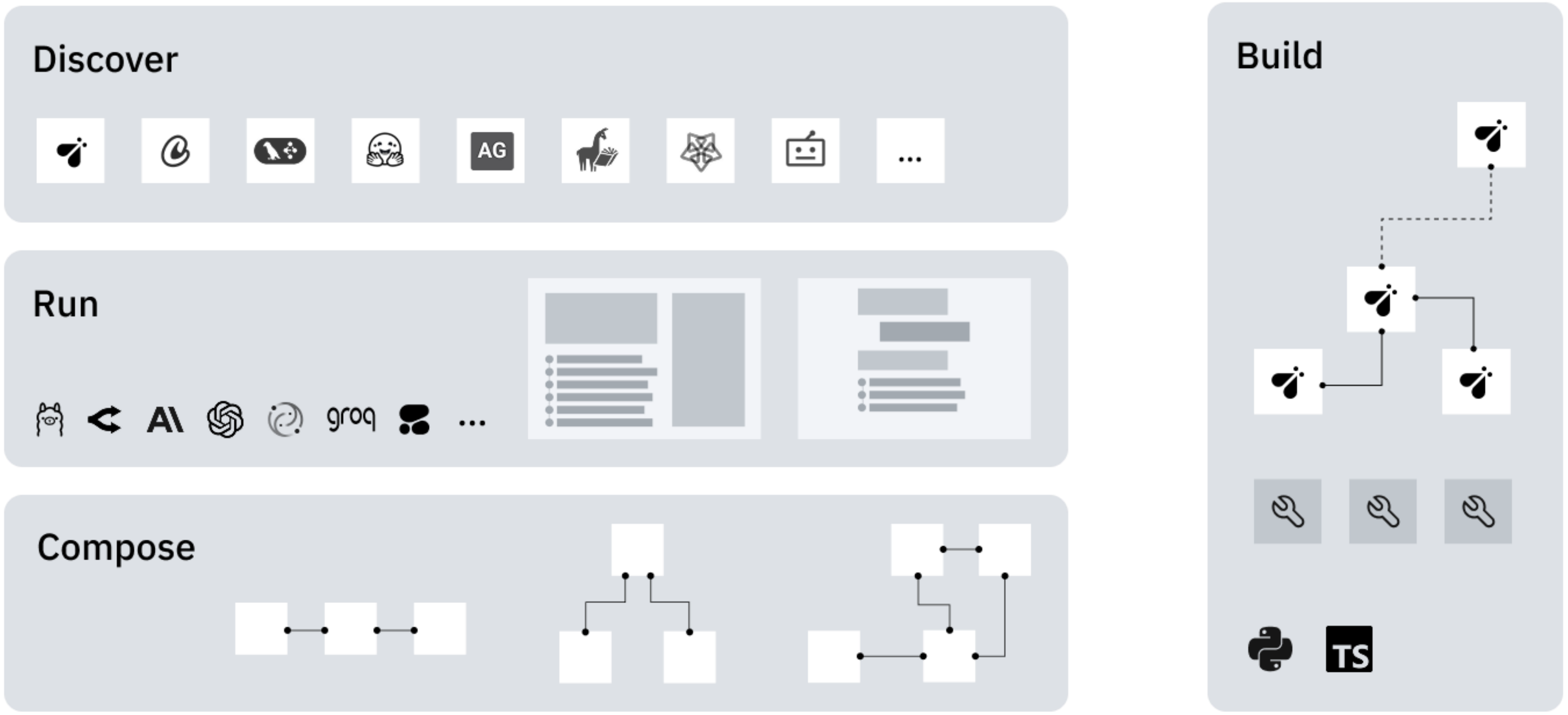
- You’re building an AI workflow… but how do you manage the messy details?
  - Do you:
    - Hand-roll your own parsing & orchestration?
    - Glue together external tools?
    - Juggle prompts across different providers?
  - Hard problems developers face today:
    - Handling different model APIs (OpenAI, Ollama, Anthropic, Granite, etc.)
    - Ensuring consistency and reliability across runs
    - Dealing with data pipelines, chunking, and metadata
    - Making workflows maintainable and scalable

It’s all just so hard!

The Vision

- BeeAI provides a ***unified framework*** for building LLM-powered apps:
  - A typed data layer (Document, Chunk, Metadata) for clarity & safety
  - Unified ingestion & parsing across file types (PDF, DOCX, CSV, Markdown)
  - Built-in model orchestration (Ollama, WatsonX, OpenAI, etc.)
  - Extendable parsers and pipelines for custom domains
  - Consistent developer-first APIs to speed up prototyping and production

Think of BeeAI as the React for AI engineering - modular, reusable, and designed to scale!



Agent Communication Protocol

ACP

Georgia Tech

Example – Simple AI Workflow

```
# Requires: beeai-framework, beeai-cli, & ollama
import asyncio
from beeai_framework.backend.message import UserMessage
from beeai_framework.backend.chat import ChatModel

async def main():
    model = ChatModel.from_name("ollama:granite3.3:2b")

    user_message = UserMessage(content="Hello! Please say 'Hello, world!' and tell me the capital of France.")

    # NOTE: pass the messages list as a keyword argument `messages=...`
    output = await model.create(messages=[user_message])

    print("=== Model response ===")
    print(output.get_text_content())

if __name__ == "__main__":
    asyncio.run(main())
```

BeeAI normalizes → parses → orchestrates → delivers insights

Georgia Tech

Why BeeAI is Powerful

- **Model-agnostic orchestration** → swap providers without rewriting pipelines
- **Typed AI engineering** → reduce errors & make workflows maintainable
- **Composable building blocks** → chain tasks like summarization, classification, reasoning
- **Production-ready foundation** → no more “glue code hell”
- **Community-driven** → open-source, extensible, and evolving

BeeAI provides a structured way to engineer AI workflows - from ingesting data, to orchestrating models, to delivering production ready applications!

FOSS LLM-Powered Root Cause Analysis Tool

Students: Kenneth (Alex) Jenkins
Mentors: Cynthia Unwin (IBM)

Georgia Tech
Open Source Program Office

Project Overview

GraniteRCA is an advanced, open-source diagnostic tool designed to perform root cause analysis (RCA) on Linux systems using AI-powered techniques and intelligent document parsing. Built with system administrators and DevOps engineers in mind, GraniteRCA leverages the capabilities of the BeeAI framework and Docling parser to analyze logs, documents, and structured data for system issues, errors, and performance bottlenecks. It supports various file formats including PDF, DOCX, HTML, and logs, and can operate in different modes optimized for specific use cases—from routine scans to live outage triage. With built-in support for containers like Docker and Podman, GraniteRCA delivers comprehensive diagnostics across both bare-metal and containerized environments.

Goals and Milestones

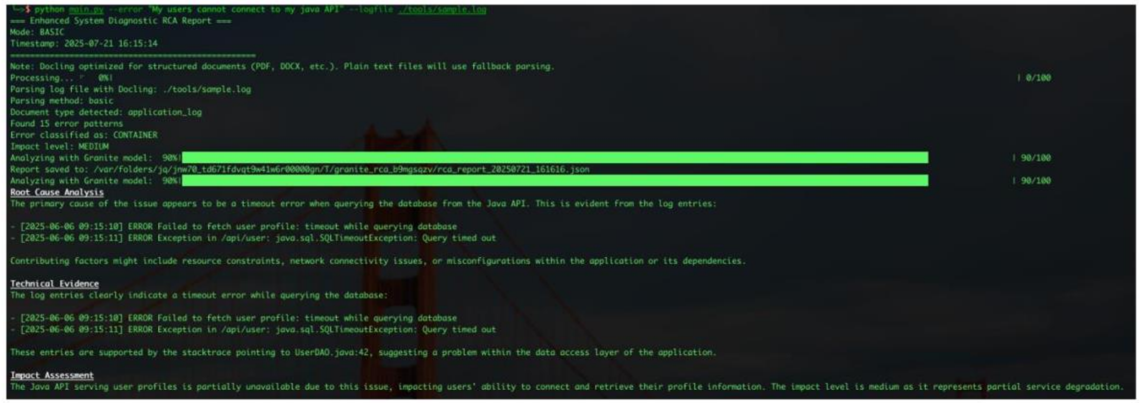
The project's primary goal is to streamline and enhance root cause analysis workflows by offering a smart, modular tool that integrates seamlessly into DevOps pipelines and live response situations. Key milestones achieved include:

- Multi-mode operation (Basic, System Scan, Quick Analysis, Triage Mode)
- Integration with BeeAI for AI-based diagnostics and context inference
- Deep document parsing capabilities using Docling
- Container monitoring and log analysis for Docker and Podman environments
- JSON-formatted report generation with impact scoring and actionable insights
- Multiple LLM backends supported (BeeAI platform, granite-io)

Open Source Outcomes

GraniteRCA embodies the spirit of collaborative development, licensed under Apache V2 and designed for transparency, extensibility, and community-driven innovation. It integrates with several open-source tools, including Docling and BeeAI, and supports contributions through clear guidelines and modular architecture. The tool has already attracted attention on GitHub, with stars, forks, and active discussions in the issues tab. By providing flexible installation methods, comprehensive documentation, and offline mode support, GraniteRCA empowers system operators across a variety of environments, including air-gapped systems.

Screenshot



```
└─$ python main.py --error "My users cannot connect to my java API" --logfile ./tools/sample.log
=== Enhanced System Diagnostic RCA Report ===
Mode: BASIC
Timestamp: 2025-07-21 16:15:14
==================================================
Note: Docling optimized for structured documents (PDF, DOCX, etc.). Plain text files will use fallback parsing.
Processing... ⠋   0%|                                                                    | 0/100
Parsing log file with Docling: ./tools/sample.log
Parsing method: basic
Document type detected: application_log
Found 15 error patterns
Error classified as: CONTAINER
Impact level: MEDIUM
Analyzing with Granite model:  90%|█████████████████████████████████████████████      | 90/100
Report saved to: /var/folders/jq/jnw70_td671fdvqt9w41w6r00000gn/T/granite_rca_b9mgsqzv/rca_report_20250721_161616.json
Analyzing with Granite model:  90%|█████████████████████████████████████████████      | 90/100
Root Cause Analysis
The primary cause of the issue appears to be a timeout error when querying the database from the Java API. This is evident from the log entries:

- [2025-06-06 09:15:10] ERROR Failed to fetch user profile: timeout while querying database
- [2025-06-06 09:15:11] ERROR Exception in /api/user: java.sql.SQLTimeoutException: Query timed out

Contributing factors might include resource constraints, network connectivity issues, or misconfigurations within the application or its dependencies.

Technical Evidence
The log entries clearly indicate a timeout error while querying the database:

- [2025-06-06 09:15:10] ERROR Failed to fetch user profile: timeout while querying database
- [2025-06-06 09:15:11] ERROR Exception in /api/user: java.sql.SQLTimeoutException: Query timed out

These entries are supported by the stacktrace pointing to UserDAO.java:42, suggesting a problem within the data access layer of the application.

Impact Assessment
The Java API serving user profiles is partially unavailable due to this issue, impacting users' ability to connect and retrieve their profile information. The impact level is medium as it represents partial service degradation.
```

Highlights and Accomplishments

GraniteRCA's intelligent design allows it to analyze a wide variety of system and application failures, from database timeouts to container crashes and security violations. Its unique multi-mode architecture enables:

- Sub-30 second quick assessments without needing file access
- Full system log scans with time-based filtering and pattern aggregation
- Live outage triage with emergency response suggestions and rollback procedures
- Structured document correlation with error pattern detection across formats
- Seamless integration between document parsing and AI-based error classification, enabling context-aware diagnostics that adapt based on the input format and urgency level.

Future Work

Future work includes:

- CI/CD testing,
- Text-based UI (TUI) for easy configuration