OpenAI

Rig DB 2.1 Rust SDK

write a comprehensive wikipedia article on rig github (rust llm ai) including specific examples in Rust sdk. span all topics from basic to advanced

Rig GitHub: An Overview

Introduction Rig is an advanced, open-source Rust library designed to streamline the development of applications powered by Large Language Models (LLMs). Its primary goal is to provide developers with an intuitive and efficient framework for integrating AI capabilities such as text classification, semantic search, and conversational agents.

This article provides a detailed breakdown of Rig's features, its Rust SDK, and step-by-step examples ranging from basic text analysis to advanced semantic search and AI agent creation.

Overview of Rig

Rig leverages Rust’s performance and safety features, making it a reliable choice for building high-performance AI systems. It abstracts away the complexities of working with LLMs like GPT models by providing robust APIs, modular components, and out-of-the-box integrations with libraries like LanceDB for vector search.

Features

- **Unified API**: Simplifies access to multiple LLMs.
- **Custom Tools**: Extensible interfaces for creating domain-specific tools.
- **Integration with LanceDB**: Allows efficient semantic search using vector embeddings.
- **Concurrency Support**: Utilizes Rust's async capabilities to enhance performance.
- **Comprehensive SDK**: Covers tasks like text classification, sentiment analysis, and interactive AI agents.

Getting Started

Setting Up a Project

To start a project with Rig:

1. Install Rust using `rustup`.
2. Create a new Rust project:

```bash
cargo new my_project
cd my_project
```

3. Add Rig dependencies in `Cargo.toml`:

```toml
[dependencies]
rig-core = "0.4.0"
```

```
tokio = { version = "1.40.0", features = ["full"] }
serde = { version = "1.0", features = ["derive"] }
```

4. Export environment variables for API keys:

```
bash

export OPENAI_API_KEY=your_api_key
```

Examples Using Rig

1. Text Classification

Using Rig to build a sentiment analysis tool:

```
rust


use rig::providers::openai;
use serde::{Deserialize, Serialize};

#[derive(Debug, Deserialize, Serialize)]
enum Sentiment {
    Positive,
    Neutral,
    Negative,
}

#[derive(Debug, Deserialize, Serialize)]
struct SentimentResult {
    sentiment: Sentiment,
    confidence: f32,
}

#[tokio::main]
async fn main() {
    let client = openai::Client::from_env();
    let classifier = client.extractor::<SentimentResult>
("gpt-4")
```

```
        .preamble("Analyze the sentiment of the given tex
t.")
        .build();

    let text = "I absolutely loved the new product!";
    let result = classifier.extract(text).await.unwrap();
    println!("{:?}", result);
}
```

2. Semantic Search with LanceDB

Efficiently search documents using embeddings:

```rust
use rig::embeddings::{DocumentEmbeddings, EmbeddingsBuilde
r};
use lancedb::connect;

#[tokio::main]
async fn main() {
    let db = connect("data/my_database").await.unwrap();
    let embeddings = EmbeddingsBuilder::new("openai-text-e
mbedding").simple_documents(
        vec![
            ("doc1", "Rust is amazing."),
            ("doc2", "Python is great for data science."),
        ],
    ).build().await.unwrap();

    let result = db.search(embeddings, "What is great abou
t Rust?").await.unwrap();
    println!("{:?}", result);
}
```

3. Building an AI Agent

Create an interactive AI assistant:

```rust
use rig::tool::{Tool, ToolDefinition};
use serde::{Deserialize, Serialize};

#[derive(Deserialize)]
struct FlightSearchArgs {
    source: String,
    destination: String,
}

pub struct FlightSearchTool;

impl Tool for FlightSearchTool {
    const NAME: &'static str = "search_flights";

    type Args = FlightSearchArgs;
    type Output = String;
    type Error = String;

    async fn call(&self, args: Self::Args) -> Result<Self::Output, Self::Error> {
        Ok(format!("Searching flights from {} to {}", args.source, args.destination))
    }
}

#[tokio::main]
async fn main() {
    let agent = rig::agent::Agent::new("AI Agent")
        .tool(FlightSearchTool)
        .build();

    let response = agent.handle_request("Find flights from NYC to LA").await.unwrap();
    println!("{}", response);
}
```

Best Practices

1. **Clear Prompts**: Structure clear and concise prompts for LLMs.
2. **Error Handling**: Handle API rate limits and failures gracefully.

3. **Security**: Protect API keys and sensitive information.
4. **Monitoring**: Implement logging and analytics to observe AI behavior.

Conclusion

Rig simplifies building advanced AI systems with Rust. From text classification to semantic search and custom agents, Rig offers tools for a variety of tasks. Explore the Rig GitHub repository for more examples, contributions, and discussions.