Rust PaLM 2.0 SDK*

1st Aniruddh Anand
Co-Founder
NovX Education
Frisco, TX

2nd Neil Shirsat
Co-Founder
NovX Education
Houston, TX

Abstract—The Rust PaLM 2.0 SDK simplifies access to the PALM 2.0 API for Rust users. This poster presents an overview of the SDK, its installation process, and usage instructions. It also highlights its unique features, such as flexible API key handling and the ability to inject custom business logic. The poster aims to demonstrate the advantages and use cases of the Rust PaLM 2.0 SDK in AI Backend Development Projects.

Introduction

The Rust PaLM 2.0 SDK was created by Aniruddh Anand and Neil Shirsat to simplify access to Googles PaLM2.0 Model on a Rust Backend. The current implimentations for Googles API's are only available for: Python, Node.js, Java, Kotlin, and Swift. This SDK widely increases access for developers to the AI model.

Installation

Command Line

In your cargo project run the command:

```
cargo add palm2_sdk
```

Cargo.toml

In your cargo.toml add the following line:

```
palm2_sdk = "0.1.0"
```

Then run the following command:

```
cargo build
cargo add palm2_sdk
```

Usage

First inialize the client:

```
let client = Client::new(
|_resource| String::from("$API_KEY"),
None,
None,
None,
None,
);
```

This library handels [api_keys] in a different way. API Keys are provided in the form a callback function or a closure. This allows keys to be provided on a per-resource level.

This library allows organizations to build and add custom business logic to the application in the form of call back functions that are specified to the client. Otherwise, a None should be specified. Specifically the library allows the client to inject a function that customizes the Http Method based on the resource fetched, a function that specificies a global header map at a resource level, a function that generates the URL to query from based on the query and path parameters, api token, and method. Additionally, an optional [reqwest::Client] can be provided to the application to customize the client used to fetch the resource.

In order to query data use the function [query] of [Client] to generate a [Query], first generate the Input Data for the Query.

All request, response, and core entities are located in use [palm2_sdk::entities].

```
let input: GenerateTextRequest =
    GenerateTextRequest {
        prompt: TextPrompt {
                text: String::from("Write
                    a story from the
                   perspective of Mickey
                   Mouse")
        },
        temperature: None,
        candidate_count: None,
        top_p: None,
        top_k: None,
        max_output_tokens: None,
        safety_settings: None,
        stop_sequences: None
};
```

Then create the [Query]

```
let query = client
.query(Resource::GenerateText)
.add_path_parameter(String::from("model")
    , String::from("text-bison-001"))
```

```
.body(input)
.build();
```

In order to fetch the resource and obtain the response use the [execute] and [execute_raw] functions (which are [async]).

```
let response: Response<
    GenerateTextResponse> = execute(query)
    .await?;

println!("Generated Ouput: {}", response.
    value.candidates[0].output);
let response: Response<String> =
    execute_raw(query).await?;

println!("Raw Output: {}", response.value
    );
```

[execute] deseializes the response data into a struct allowing its values to be used by the program.

[execute_raw] keeps the response data in a string representation. Use this function over the [execute] function when the library is being it used in a reverse proxy since proxies do not need to deserialize and reserialize the data. Essentially, [execute_raw] reduces the overheat and improves performance of the library in a proxy enviornment.

Acknowledgment

We thank Sahithi Morla for help in researching AI Models during our selection process, and both Camden McLeod and Franchesca Untalan for helping support us through the process, managing NovX Education, and implimenting subsequent resources which will impliment the API.

References

[1] R. Anil et al., "PaLM 2 Technical Report," arXiv:2305.10403, 2023.