example for RKLLM-Server-Flask, helping users understand the construction logic of the example code for potential secondary development.

The deployment example of RKLLM-Server-Flask primarily relies on the Flask library to achieve the basic implementation of the server. Additionally, for RKLLM model inference, the ctypes library in Python is chosen to directly call the RKLLM Runtime library.

In the overall implementation of rkllm_server/flask_server.py, in order to call librkllmrt.so via ctypes, it's necessary to define relevant structures in Python based on the header file rkllm.h corresponding to librkllmrt.so beforehand. After the Flask server receives the struct data sent by users, it calls relevant functions to perform inference with the RKLLM model. The specific code implementation of flask_server.py mainly consists of the following steps:

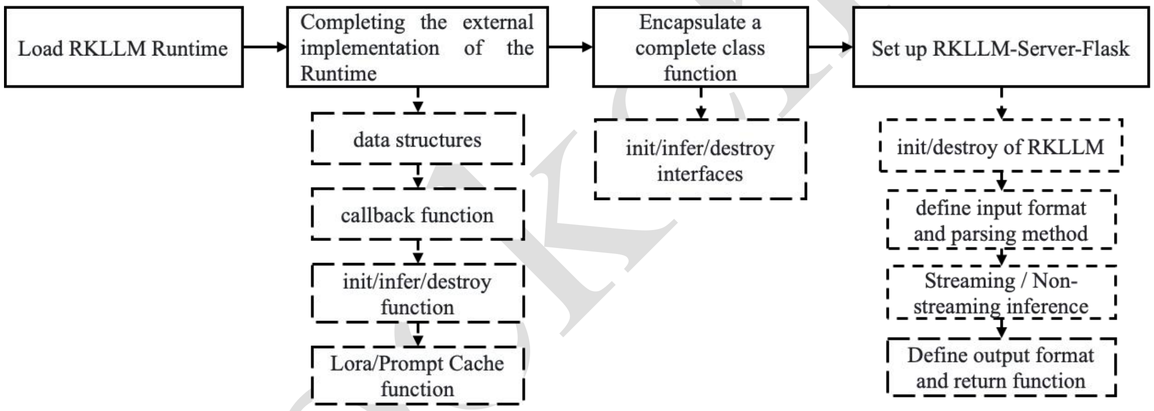
Load RKLLM Runtime
Completing the external implementation of the Runtime
Encapsulate a complete class function
Set up RKLLM-Server-Flask
data structures
callback function
init/infer/destroy function
Lora/Prompt Cache function
init/infer/destroy interfaces
init/destroy of RKLLM
define input format and parsing method
Streaming / Non-streaming inference
Define output format and return function

Figure 3-3 Overview of the RKLLM-Server-Flask Deployment Implementation Process

1) Load RKLLM Runtime: Set the path for the dynamic library and load the dynamic library librkllmrt.so using ctypes to achieve the analysis of the RKLLM Runtime.

2) Completing the external implementation of the Runtime: In the Python code, use ctypes to complete the external definitions of relevant implementations from the RKLLM Runtime header files, including definitions of data structures, callback functions, initialization functions, inference functions, destory functions, and loading functions for Lora/Prompt Cache.

3) Encapsulate a complete class function: Based on the various Runtime data types and function interfaces implemented in step 2), encapsulate complete class functions that integrate RKLLM's initialization, inference, destory, and other operations for easier subsequent calls.