Streaming WASI-NN For LLM

@hydai
2024/02/05

Agenda

- Metadata for the configuration and return values
- Pulling function `compute_single` & `get_output_single` for streaming tokens
- `fini_single` for destroying the context
- Custom errors

Metadata for `setInput`

- `setInput` function:
  - Index = 0, the raw input for the model, e.g. “What’s the capital of Japan?”
  - Index = 1, the metadata JSON object
- Metadata JSON object:
  - Key-Value set
  - Including all configuration that needs for LLM:
    - `embedding`: Enable the embedding mode, which will return a embedding float vector.
    - `ctx-size`: Set the context size
    - `n-predict`: Set the number of tokens to predict
    - `n-gpu-layers`: Set the number of layers to store in VRAM
    - `reverse-prompt`: Set it to the token at which you want to halt the generation
    - `batch-size`: Set the batch size number for prompt processing
    - `temp`: Set the temperature for the generation
    - And more…

Metadata for `getOutput`

- `getOutput` function:
  - Index = 0, the raw output produced by the model, e.g. “The capital of Japan is Tokyo.”
  - Index = 1, the metadata JSON object
- Metadata JSON object:
  - Key-Value set
  - Including all other informations produced by the LLM
    - `llama_commit`: The backend version, use commit hash
    - `llama_build_number`: The backend version, use build number
    - `input_tokens`: The # of input tokens, which is encoded by the given input from idx(0).
    - `output_tokens`: The # of output tokens, which is encoded by the produced output from idx(0).
    - And more…

Step1. Create Graph

- Compose the configuration JSON Object `options`

```
wasi_nn::GraphBuilder::new(
    wasi_nn::GraphEncoding::Ggml,
    wasi_nn::ExecutionTarget::AUTO)
   .config(
       serde_json::to_string(&options)
       .expect("Failed to serialize options"))
   .build_from_cache(model_name)
   .expect("Failed to build graph");
```

Step2. Init context

- `let mut context = graph`
- `    .init_execution_context()`
- `    .expect("Failed to init context");`
- The context will use the configuration from Step 1.
- However, we allow users to overwrite the configuration via setInput function.
- `fn set_metadata_to_context(`
- `    context: &mut GraphExecutionContext,`
- `    data: Vec<u8>,`
- `) -> Result<(), wasi_nn::Error> {`
- `    context.set_input(1, wasi_nn::TensorType::U8, &[1], &data)`
- `}`

Step2 - Con’t

- The reason is that some configuration such as temperature, context size, or batch size may change if the application is a LLM API server.
- These options will depend on the requests from the front end.

Step3. The main loop

```
loop {
    get_prompt_from_users();
    // Set prompt to the input tensor.
    context.set_input(0, wasi_nn::TensorType::U8, &[1], &data)
    // Execute the inference (streaming mode).
    loop {
        match context.compute_single() {
            Ok(_) => (),
            Err(wasi_nn::Error::BackendError(wasi_nn::BackendError::EndOfSequence)) => {break;}
            Err(wasi_nn::Error::BackendError(wasi_nn::BackendError::ContextFull)) => {break;}
            Err(wasi_nn::Error::BackendError(wasi_nn::BackendError::PromptTooLong)) => {break;}
            Err(err) => {break;}
        }
        // Retrieve the single output token and print it.
        let token = context.get_output_single(index, &mut output_buffer).unwrap();
        print!("{}", token);
    }
    // Delete the context in compute_single mode.
    context.fini_single().unwrap();
}
```

Extended APIs and Types

- https://github.com/bytecodealliance/wasi-nn/compare/main...second-state:wasmedge-wasi-nn:ggml