Editing the USER CODE: Explanation

October 4, 2024

Overview

The USER CODE section in the generated X-CUBE-AI project allows for custom implementation of data acquisition, preprocessing, inference, and post-processing. The functions `acquire_and_process_data` and `post_process` are the key points for integrating your application's logic with the generated AI model.

1. One Input and One Output Only

The generated code from X-CUBE-AI defines the number of input and output buffers as `AI_NETWORK_IN_NUM` and `AI_NETWORK_OUT_NUM`, both set to `1`:

- This means that the network can process only a single input and produce a single output at a time.
- The input and output are handled as single-dimensional arrays, simplifying the structure but limiting throughput.
- For this reason, we have to feed data sequentially, one value at a time, which aligns with the design of the predefined float inputs.

2. Predefined Float Indices Array

Instead of using a dynamic input buffer, we define a static array of float values to represent the sentiment scores for our text examples:

```
const float predefined_inputs[] = {0.8f, 0.5f, 0.2f};
```

- Each value in the array corresponds to a sentiment score mapped in the Python preprocessing step.
- These values are fed into the neural network input buffer one by one in the `acquire_and_process_data` function.
- This approach eliminates the need for complex string or integer tokenization on the embedded side.

3. Editing `int acquire_and_process_data(ai_i8* data[])`

This function is responsible for loading the predefined input data into the model's input buffer:

- The `data` parameter is a pointer to the input buffer where data needs to be written.
- The function loops through the `predefined_inputs[]` array and assigns each value to the buffer, one at a time.
- For example:

```
float* input_data = (float*)data[0];
input_data[0] = predefined_inputs[current_index];
```

- After loading the input value, the `current_index` is incremented, allowing the next value to be used in the following inference.

4. Editing `int post_process(ai_i8* data[])`

This function handles the output generated by the model:

- The output buffer contains a single float value, which represents the prediction for the given input.
- Example of extracting and printing the output:

```
float* output_data = (float*)data[0];
printf("Prediction: %f\r\n", output_data[0]);
```

- The function can be extended to map this numerical output back to a text label, such as:

```
if (output_data[0] < 0.3) {
    printf("Prediction: Negative Sentiment\r\n");
} else if (output_data[0] < 0.7) {
    printf("Prediction: Neutral Sentiment\r\n");
} else {
    printf("Prediction: Positive Sentiment\r\n");
}
```

5. Reviewing `app_x-cube-ai.c`

The `app_x-cube-ai.c` file is the primary user entry point for managing inference. Key sections include:

- **Initialization**: Functions to set up the model and memory.
- **Inference Loop**: Where `acquire_and_process_data` is called, followed by inference execution, and then `post_process`.
- **Utility Functions**: Additional helper functions for managing buffer allocations, error handling, and memory cleanup.

Editing this file allows you to customize how data flows through the network and how results are displayed or processed.

Conclusion

The USER CODE section in the X-CUBE-AI project is where application-specific logic is implemented. By editing the `acquire_and_process_data` and `post_process` functions, you can manage inputs and outputs effectively, ensuring the model integrates seamlessly with your application.