Melaku Desalegn

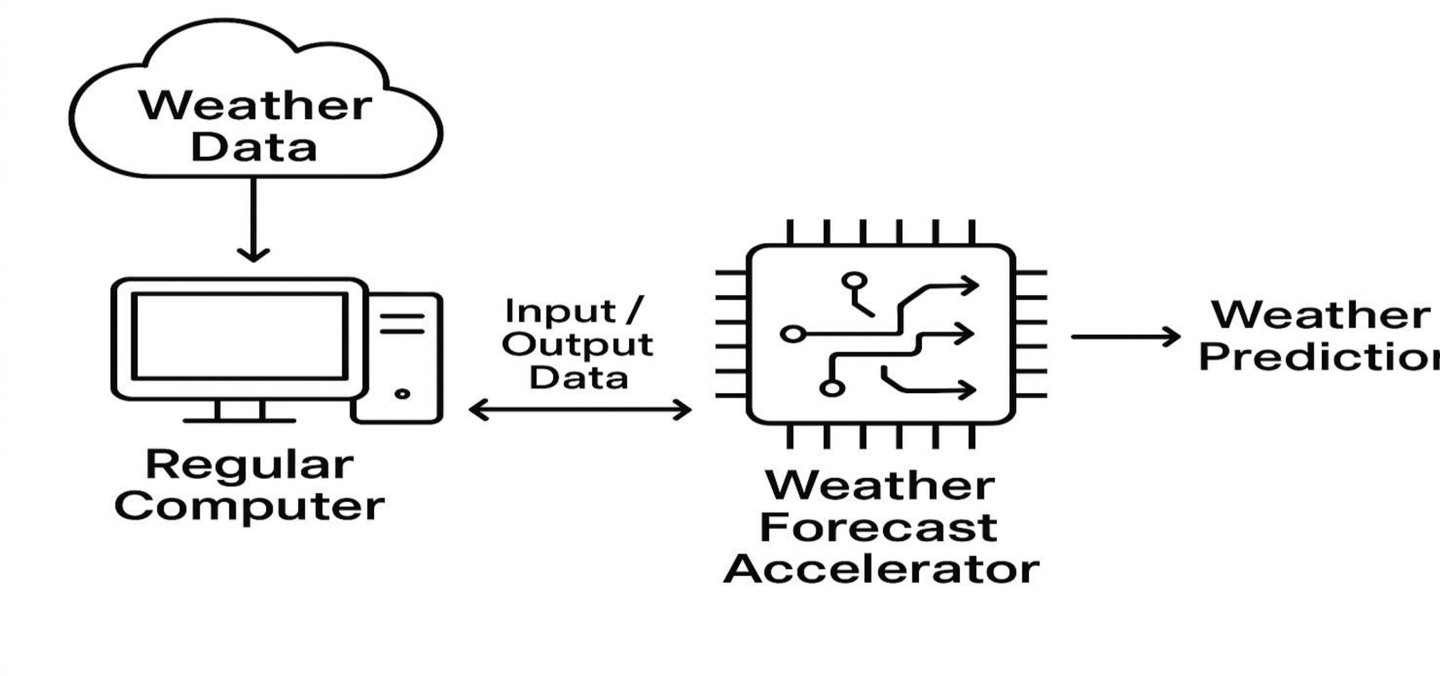
ECE510- Spring 2025

Week2

Design, test, and benchmark a co-processor chaplet that accelerates parts of some AL. ML code /algorithm for Time series analysis for weather forecasts.

1. What are you trying to do?

We want to build a small, specialized computer chip that can make weather forecasts faster and more efficiently. It works alongside a regular computer to speed up the parts of the weather prediction process that take the most time.



2. How is it done today, and what are the limits of current practice?

Right now, weather forecasting with machine learning is usually done on regular CPUs or GPUs. These systems are powerful, but they're not optimized for the repetitive and time-based nature of weather data. This can lead to high power use, longer wait times, and less efficient scaling—especially when running models on large amounts of local weather data (like for cities or farms).

3. What is new in your approach and why do you think it will be successful?

We’re creating a custom chiplet that is purpose-built for the kind of math used in time-based weather prediction models like GRUs or Transformers. Unlike general-purpose chips, this one only focuses on what’s needed—like handling time sequences and doing small, fast calculations. It’s smaller, uses less power, and can sit close to the main computer or even inside edge devices (like weather stations). That focus gives us speed, energy savings, and cost benefits.

4. Who cares? What difference will it make if you succeed?

- **Weather services** could run forecasts more often and at lower cost.
- **Farmers and renewable energy operators** could get more accurate local predictions in real-time.
- **Developing regions** could run advanced models on low-power hardware.
- **Research labs** could run massive simulations faster, with lower cloud compute bills.

5. What are the risks?

- The chiplet might not outperform optimized GPUs if not carefully designed.
- Adapting existing ML models to run on the chiplet could be complex.
- There’s a risk that energy savings don’t offset development cost unless used at scale.
- Fabrication delays or bugs could slow progress.

6. How much will it cost?

- **Prototype development** (FPGA or emulation): ~$50K
- **First silicon test chiplet** (with shuttle runs): ~$100K–$250K
- **Full product development** (design + software stack): ~$500K–$1M
- **Optional:** Additional $100K+ for integration with cloud platforms or edge hardware

7. How long will it take?

- **Design & simulation (python/HLS)**: 3–4 months
- **FPGA prototype testing**: 2–3 months
- **Chiplet tape out & fabrication**: 3–5 months
- **Software stack & runtime (ONNX adapter, etc.)**: parallel over 6–8 months
- **Total time to first deployment-ready chiplet**: ~12 months

8. What are the mid-term and final “exams” to check for success?

Mid-Term Milestones:

- FPGA-based chiplet emulation outperforms CPU baseline for GRU inference
- Weather forecast model converted to ONNX runs on chiplet with >90% accuracy retained
- Power usage <50% of equivalent GPU workload

Final Exams:

- End-to-end weather model (e.g., 30-day sequence) runs 3× faster and 5× more energy-efficient than on CPU
- Integration tested with real datasets (e.g., NOAA, WeatherBench)
- Deployed in at least one real-world environment (research station, drone, or farm)

1. Functional Focus:

- Accelerate **GRU**, **LSTM**, and **Transformer-style attention** operations, especially:
  - Matrix-vector multiplies for sequential data
  - Softmax + layer norm optimizations
  - Quantized activations to reduce memory and bandwidth pressure

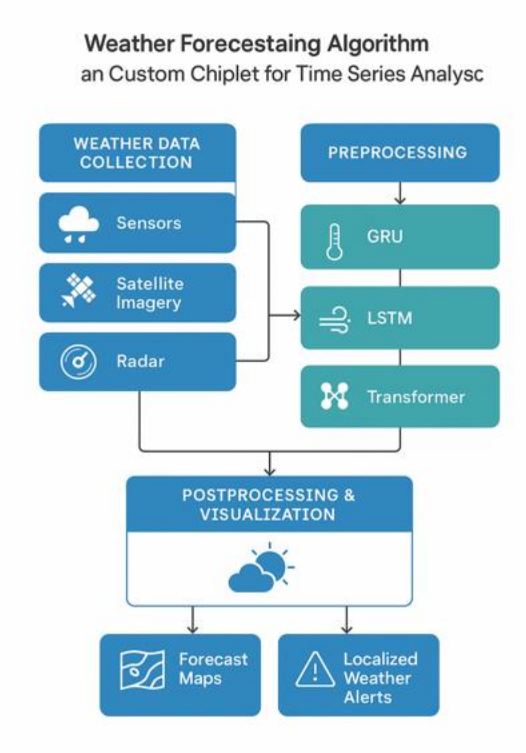
2. Chiplet Architecture:

- Memory-near-compute: Local SRAM for time-series window buffers
- Lightweight scheduler for sequence ops
- On-chip accelerator cores: 4–8 small vector engines with support for time-step parallelism
- IO support: PCIe/CXL or tile-based interconnect

3. Software Stack Outline:

- Runtime for scheduling sequences
- ONNX runtime compatibility layer
- Quantization-aware training pipeline (optional but helpful)

Draw for me an algorithm for weather forecast.



A code for simulating the diagram in Verilog.

create a **basic Verilog simulation** that mimics the data flow shown in the diagram:

- **Weather Data** comes into a **regular CPU** (modeled as a testbench).
- It sends time series data to a **Weather Forecast Accelerator**.
- The accelerator processes it (simple logic) and sends back **weather predictions**.

Simulation Scope

For simplicity:

- We'll simulate **1D time series data** as an input.
- The "forecast accelerator" will be a placeholder that performs a moving average (simulating simple model behavior).

Verilog Modules

1. `forecast_accelerator.v` — Simple Moving Average Module

```
verilog
CopyEdit
```

```verilog
module forecast_accelerator(
    input clk,
    input rst,
    input [7:0] data_in,
    input       valid_in,
    output reg [7:0] data_out,
    output reg       valid_out
);

    reg [7:0] buffer[2:0];  // 3-sample window
    integer i;

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            for (i = 0; i < 3; i = i + 1)
                buffer[i] <= 0;
            data_out <= 0;
            valid_out <= 0;
        end else if (valid_in) begin
            // Shift buffer
            buffer[2] <= buffer[1];
            buffer[1] <= buffer[0];
            buffer[0] <= data_in;

            // Simple moving average output
            data_out <= (buffer[0] + buffer[1] + buffer[2]) / 3;
            valid_out <= 1;
        end else begin
            valid_out <= 0;
        end
    end
endmodule
```

---

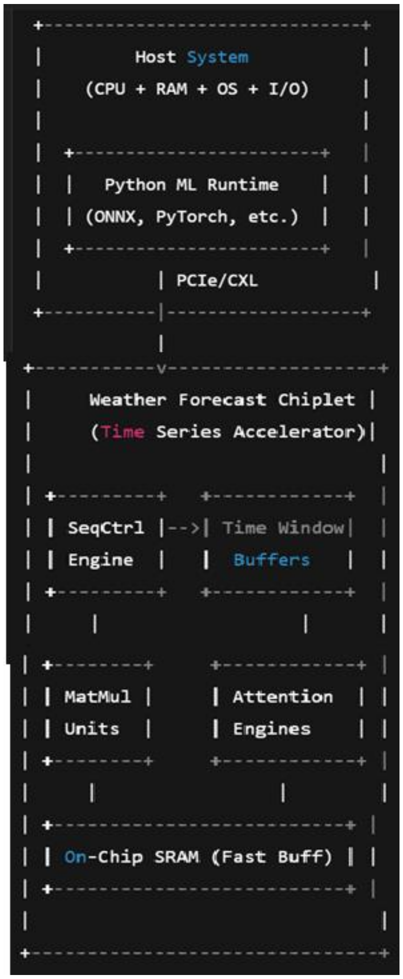
2. `testbench.v` — Simulates "Regular Computer."

```
verilog
CopyEdit
`timescale 1ns/1ps
module testbench;

    reg clk = 0;
    reg rst = 1;
    reg [7:0] data_in;
    reg       valid_in;
    wire [7:0] data_out;
    wire       valid_out;

    // Instantiate Accelerator
    forecast_accelerator accel (
        .clk(clk),
        .rst(rst),
        .data_in(data_in),
        .valid_in(valid_in),
        .data_out(data_out),
        .valid_out(valid_out)
```

```
    );

    // Clock generation
    always #5 clk = ~clk;

    integer i;
    reg [7:0] test_data[9:0];

    initial begin
        // Simulated weather data (e.g., temperature or pressure values)
        test_data[0] = 8'd20;
        test_data[1] = 8'd22;
        test_data[2] = 8'd23;
        test_data[3] = 8'd25;
        test_data[4] = 8'd24;
        test_data[5] = 8'd26;
        test_data[6] = 8'd27;
        test_data[7] = 8'd29;
        test_data[8] = 8'd28;
        test_data[9] = 8'd30;

        #10 rst = 0;  // Release reset

        for (i = 0; i < 10; i = i + 1) begin
            @(posedge clk);
            data_in = test_data[i];
            valid_in = 1;
        end

        @(posedge clk);
        valid_in = 0;

        #100 $finish;
    end

    // Monitor output
    always @(posedge clk) begin
        if (valid_out)
            $display("At time %t: Forecast = %d", $time, data_out);
    end

endmodule
```

Output

Hardware Block Diagram – Weather Forecast Co-Processor Chiplet.



weather_forecast_week_2.py ×

C: > Users > melde > PycharmProjects > PythonProject1 > PythonProject > weather_forecast_week_2.py > WeatherChiplet > reset

```python
class WeatherChiplet:
    def __init__(self):
        self.reset()

    def reset(self):
        self.state = "RESET"
        self.input_seq = []
        self.output_forecast = None

    def load_sequence(self, input_seq):
        self.input_seq = input_seq
        self.state = "READY"

    def compute_forecast(self):
        if self.state != "READY":
            raise Exception("Chiplet not ready. Load sequence first.")

        # Simulate some logic: average of last 7 inputs
        self.output_forecast = sum(self.input_seq[-7:]) / 7
        self.state = "DONE"
```

```
        return self.output_forecast


# ==== Python Testbench ====
if __name__ == "__main__":
    # Simulated clock/reset (not needed in Python but mirrored from Verilog)
    clk = True
    rst = True

    # Initialize chiplet
    chiplet = WeatherChiplet()

    # Simulate reset
    if rst:
        chiplet.reset()

```

```python
    # Provide example 30-day input sequence
    input_sequence = [15 + (i % 10) for i in range(30)]  # Example: 15 to 24 degrees

    # Load sequence and compute forecast
    chiplet.load_sequence(input_sequence)
    forecast = chiplet.compute_forecast()

    # Output result
    print("Input Sequence (last 7 days):", input_sequence[-7:])
    print("Forecast Output:", forecast)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\melde> & C:/Users/melde/AppData/Local/Programs/Python/Python313/python.exe c:/Users/melde/PycharmP
cts/PythonProject1/PythonProject/weather_forecast_week_2.py
Input Sequence (last 7 days): [18, 19, 20, 21, 22, 23, 24]
Forecast Output: 21.0
PS C:\Users\melde>
```