$PM_{2.5}$ Forecasting with Chronos: Accuracy–Performance Trade-offs from Laptop to Raspberry Pi

Aneeket Yadav and Vaibhav Bajaj
Department of Computer Science and Engineering
Indian Institute of Technology Delhi
{cs1221116,cs5210126}@iitd.ac.in

Abstract

We study end-to-end $PM_{2.5}$ forecasting for two Indian cities (Gurgaon and Patna) using the Chronos family of time-series language models, and systematically evaluate accuracy and performance across heterogeneous hardware.

In Part 1, we run zero-shot forecasting with multiple Chronos-2, Chronos Bolt and Chronos T5 variants on hourly city-level $PM_{2.5}$ data from July–December 2024, with varying context length and forecast horizon.

In Part 2, we instrument the inference loop on a laptop CPU with a custom CSV logger plus `psutil`, capturing per-batch latency, throughput, CPU utilization, DRAM usage, and process RSS.

In Part 3, we incorporate meteorological covariates (relative humidity, temperature, and wind speed) using Chronos-2 multivariate inputs and compare RMSE distributions with and without covariates.

In Part 4, we port the forecasting code to a Raspberry Pi, pin the process to $1 \ldots 3$ cores, and measure latency, throughput, and CPU temperature over 30 minutes of continuous inference, contrasting behaviour with the laptop.

Finally, in Part 5, we implement a FastAPI-based inference server that accepts client requests with different hyper-parameters (context, horizon, model variant, covariates flag), replays synthetic traces of 500 requests (both bursty and Poisson arrivals), and explores optimizations inspired by recent LLM systems work.

Our results highlight clear accuracy–performance trade-offs: larger Chronos-2 models and longer contexts improve RMSE but saturate CPU resources, while smaller Bolt variants and carefully chosen horizons yield acceptable accuracy with much better latency, especially on embedded hardware.

1 Introduction

Short-term forecasting of fine particulate matter ($PM_{2.5}$) is a critical component of air-quality monitoring infrastructure, as it supports both city-scale pollution control and personal exposure decisions. Recent work on time-series language models, such as Chronos-2, demonstrates strong zero-shot forecasting performance across diverse domains without task-specific fine-tuning. However, most demonstrations focus solely on accuracy, with limited analysis of runtime metrics or edge deployment constraints.

In this project, we consider a concrete deployment-oriented setting: hourly $PM_{2.5}$ prediction for two Indian cities (Gurgaon and Patna) using publicly available data from https://vayu.undp.org.in/. We systematically explore (i) which Chronos model variant, context length, and forecast horizon yields the best average RMSE across the two cities; (ii) how these choices affect latency, CPU and memory utilization on a laptop CPU; (iii) whether meteorological covariates (relative humidity, temperature, wind speed) improve forecasting; (iv) how performance changes when moving to constrained embedded hardware (Raspberry Pi); and (v) how a basic inference server behaves under realistic request patterns and simple LLM-style optimizations.

The assignment is organized into five parts; we mirror this structure in the report.

Contributions.

- A systematic evaluation of multiple Chronos-2 and Bolt variants on city-level $PM_{2.5}$, sweeping context length and forecast horizon, with per-city RMSE analysis.
- A measurement framework that logs latency and system-level metrics (CPU, memory, temperature) during continuous forecasting on both laptop and Raspberry Pi.

- An empirical study of meteorological covariates, comparing RMSE distributions with and without covariates under a common test split and multi-variate Chronos-2 setup.

- A simple but functional inference server that accepts heterogeneous client requests and explores batching, strict model-count limits, and basic quantization as accuracy–latency trade-offs.

2 Data and Problem Setup

2.1 Dataset

We are given two CSV files, `gurgaon.csv` and `patna.csv`, each containing 185 days of hourly data between July and December 2024. The columns are:

- column 1: timestamp (hourly),
- column 2: relative humidity,
- column 3: temperature,
- column 4: wind speed,
- column 5: calibrated $PM_{2.5}$, averaged over all sensors in the city.

For Part 1 and Part 2 we only use timestamp and $PM_{2.5}$, treating the series as univariate. In Part 3 we additionally use the meteorological covariates as exogenous inputs.

We load each CSV into a Pandas dataframe and standardize to a Chronos-compatible format:

- convert timestamps to `datetime` and sort by time,
- create an `id` column with values `"gurgaon"` and `"patna"`,
- rename the $PM_{2.5}$ column to `"target"`.

2.2 Forecasting Task

For each city, we consider a sliding-window setup:

- context length: $C \in \{2, 4, 8, 10, 14\}$ days (i.e., $C \cdot 24$ hours),
- forecast horizon: $H \in \{4, 8, 12, 24, 48\}$ hours.

Given a context window of C days of past $PM_{2.5}$ values, we zero-shot forecast the next H hours using Chronos and compute RMSE against the ground truth.

We use an expanding sliding window:

$$\text{context indices: } [i - C{\cdot}24,\ i), \quad \text{horizon indices: } [i,\ i + H),$$

stepping by a configurable stride (e.g., 1 or 4 hours) and optionally limiting the number of windows per configuration to bound runtime.

3 Part 1: Chronos Forecasting Accuracy

3.1 Chronos Models and Interface

We evaluate several pre-trained models from the Chronos family on Hugging Face, including Chronos-2 and Chronos Bolt variants. We use the official `Chronos2Pipeline`, `ChronosBoltPipeline`, and `ChronosPipeline` interfaces, following the updated Chronos-2 quick-start:

- load models via `from_pretrained(model_name, device_map, dtype)`,
- call `predict_df` with arguments: `id_column="id"`, `timestamp_column="timestamp"`, `target="target"`, `prediction_length=H`, and quantile levels $\{0.1, 0.5, 0.9\}$.

We treat the median forecast (quantile 0.5) as the point prediction.

3.2 RMSE vs. Context Length

For each model and each city, we fix the forecast horizon to 24 hours and sweep context length $C \in \{2, 4, 8, 10, 14\}$ days. For each configuration we evaluate up to a fixed number of windows (e.g., 256) to keep runtime manageable and compute mean RMSE across windows.

Figures 1 and 2 show a plot of average RMSE (y-axis) vs. context length (x-axis) and models for Gurgaon, while Figures 3 and 4 show a plot of average RMSE (y-axis) vs. context length (x-axis) and models for Patna.

From the figures we can see that:

- RMSE decreases when increasing context from 2 to 4–10 days, as the model sees more diurnal and weekly patterns.

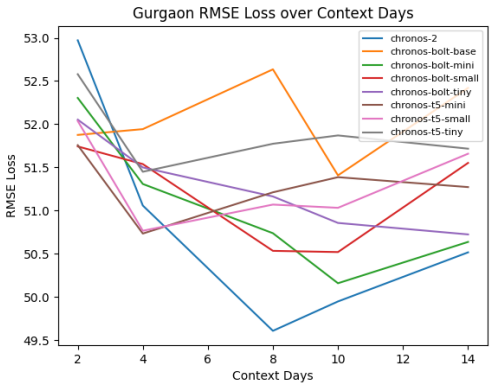

Figure 1: Average RMSE vs. context length (2–14 days) for a 24-hour forecast horizon, for Gurgaon.

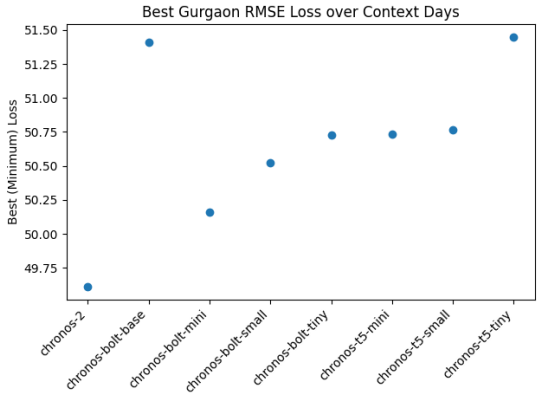

Figure 2: Best Average RMSE vs. Model for all Context lengths (2–14 days) for a 24-hour forecast horizon, for Gurgaon.

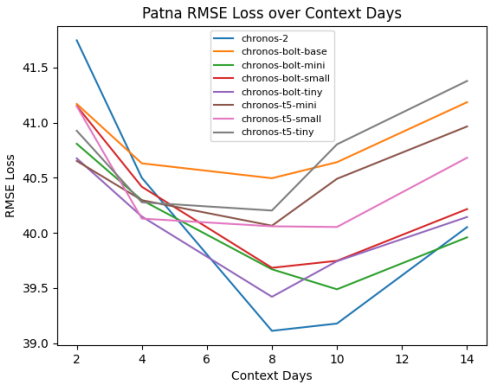

Figure 3: Average RMSE vs. context length (2–14 days) for a 24-hour forecast horizon, for Patna.

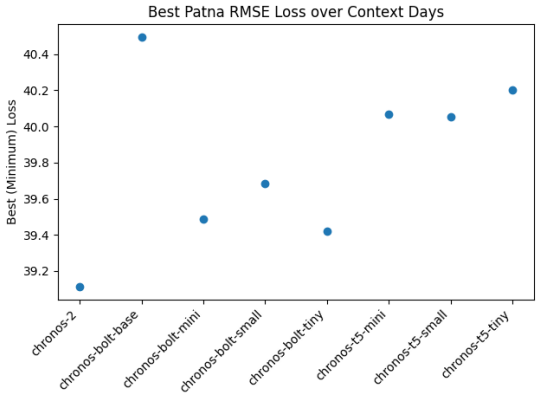

Figure 4: Best Average RMSE vs. Model for all Context lengths (2–14 days) for a 24-hour forecast horizon, for Patna.

- Diminishing returns (and slight degradation) for 10–14 days due to older context being less informative and increased sequence length.
- Differences between cities, with Gurgaon exhibiting sharper pollution spikes and higher baseline RMSE.

3.3 RMSE vs. Horizon

Next, we fix context length to 10 days (240 hours) and sweep forecast horizon $H \in \{4, 8, 12, 24, 48\}$ hours. Figure 5, 6, 7, and 8 show plots of RMSE vs. horizon for both cities.

We observe the expected pattern of RMSE increasing with horizon: short horizons (4–8 hours) are easier, while 24–48 hour forecasts must anticipate sharper dynamics and longer-term trends.

3.4 Best Hyper-Parameters per City

For each model and city, we record:

- the context length with best average RMSE for 24h horizon;
- the horizon with best average RMSE for 10-day context.

We summarize per-city best configurations in Table 1. These hyper-parameters are used in Part 3 when we compare forecasting with and without covariates.

4 Part 2: Laptop CPU Performance Measurement

4.1 Measurement Framework

To analyze performance metrics, we instrument the evaluation loop with:

- a lightweight CSV logger (`CsvMetricsLogger`) that logs: `city`, `model`, `context_hours`, `horizon_hours`, `latency_s`, `batch_size`, `stride`,

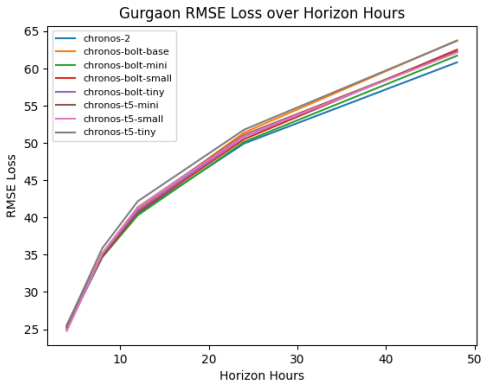

Figure 5: Average RMSE vs. forecast horizon (4–48 hours) for a 10-day context window, for Gurgaon.

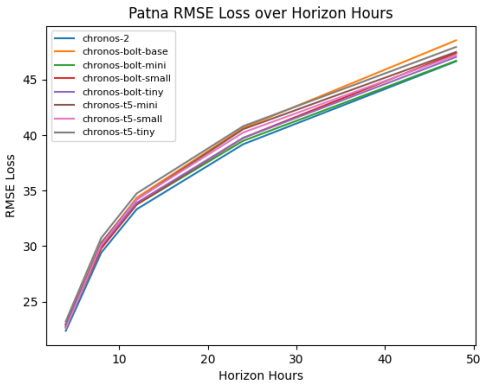

Figure 7: Average RMSE vs. forecast horizon (4–48 hours) for a 10-day context window, for Patna.

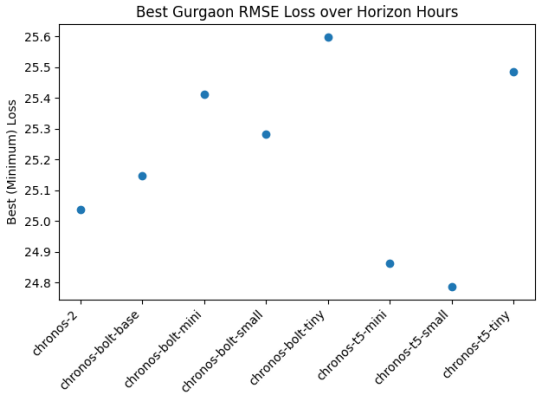

Figure 6: Best Average RMSE vs. Model for all forecast horizon (4–48 hours) for a 10-day context, for Gurgaon.

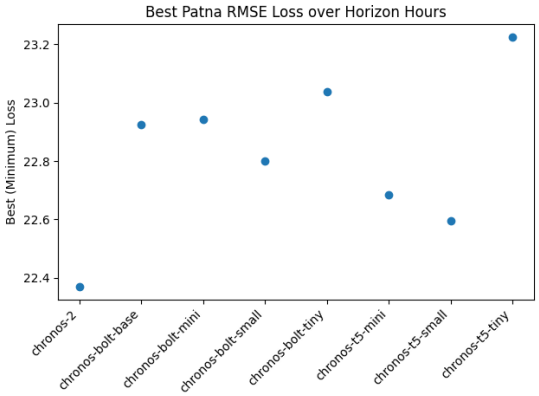

Figure 8: Best Average RMSE vs. Model for all forecast horizon (4–48 hours) for a 10-day context, for Patna.

- a separate `system_stats.csv` file written via Python's `csv.DictWriter`, containing `wall_ts`, `cpu_total_pct`, `cpu_user_pct`, `cpu_sys_pct`, `mem_used_mb`, `mem_avail_mb`, `proc_cpu_pct`, `proc_rss_mb`.

We use the `psutil` library to sample CPU and memory stats at the end of each batched call to `predict_df`. Batch size and sliding-window stride are configurable; we use moderate batch sizes (e.g., 1–32 windows per batch) to simulate continuous inference without overwhelming RAM.

4.2 Continuous Benchmark Runs

Instead of exhausting the entire dataset per configuration, we run each (model, city, context, horizon) combination for a fixed wall-clock budget (e.g., 2 minutes). Within that budget, the script slides the context window, assembles batches, calls Chronos, and logs per-batch latencies and system metrics.

Figure 9, 10, and 11 shows a plot of:

- end-to-end batch latency over time,
- CPU utilization over time,
- process RSS (memory usage) over time,

for a single Chronos-bolt-tiny (the fastest) model on the laptop as the context length increases.

4.3 Accuracy vs. Performance Trade-offs

By joining the RMSE results from Part 1 with the latency and system metrics from Part 2, we can inspect accuracy–performance trade-offs:

- Chronos-2 performed the best both in terms of accuracy for Patna, and in context window sizes for Gurgaon, with comparable best results for Horizons in Gurgaon by the best Chronos-T5-Small (25.037 vs 24.786).
- Larger Chronos models generally offer similar RMSE but higher per-batch latency and CPU utilization.

City	Best context (24h)	Best horizon (10d)
Gurgaon	8 days (C-2)	4 hours (C-T5-S)
Patna	8 days (C-2)	4 hours (C-2)

Table 1: Summary of best context length and horizon per city, based on average RMSE.

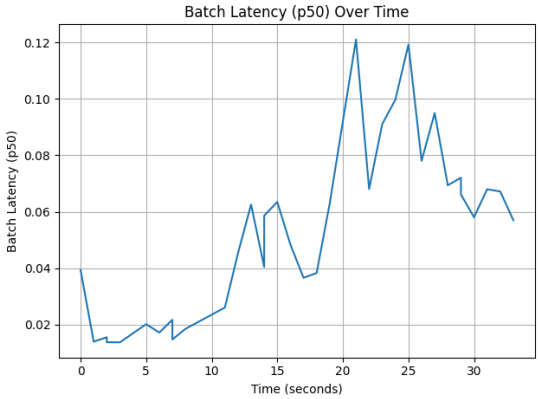

Figure 9: Example laptop p50 during a benchmark run as the context length increases

- Increasing context length increases sequence length and thus inference cost; beyond 10 days, the marginal accuracy gains (or losses) are small relative to latency.

- Shorter horizons (4–8 hours) are both more accurate and faster than 24–48 hours, as prediction length directly affects computation.

We compared the models runtimes and usages in the Figures 12, 13, 14, and 15.

5 Part 3: Effect of Meteorological Covariates

5.1 Setup and Splitting

To study the effect of covariates, we used the update Chronos-2 model, as it was also the best performing model out of all of the others.

For the "best" Chronos-2 hyper-parameters from Part 1 (model variant, context length, horizon), we run two pipelines:

1. **Univariate:** Chronos-2 using only $PM_{2.5}$ as target, same as Part 1.

2. **With covariates:**

    - Chronos-2 with multi-variate inputs, adding humidity, temperature, and wind speed as extra time-series variables (assuming perfect future covariates from the CSV).

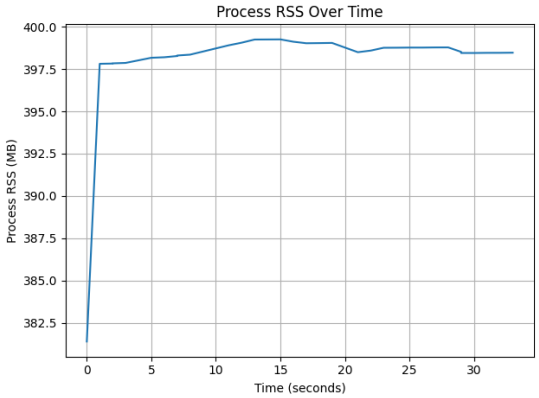

Figure 10: Example laptop process RSS during a benchmark run as the context length increases

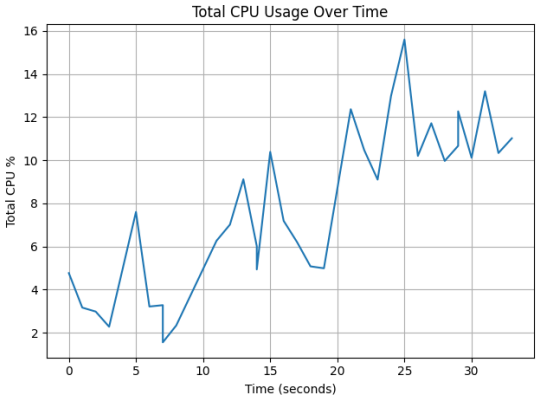

Figure 11: Example laptop total CPU usage during a benchmark run as the context length increases

5.2 RMSE Distribution and Sudden Changes

We plot the full CDF of per-window RMSE for both variants for randomly selected days, as in Figure 16, to capture distributional effects beyond the mean.

To investigate the effects of covariates, we look at all the $2^3 = 8$ cobinations of them, and calculate the RMSE results on randomly selected windows of fixed lengths, as in Figures17,18, and 19. We additionally plot $PM_{2.5}$ over time for selected test days, overlaying ground truth and forecasts, and annotating RMSE (Figure 21, 20, 23, and 22).

5.3 Performance with Covariates

We re-use the logging infrastructure from Part 2 to compare performance metrics with and without covariates for the same model variant and hardware. Chronos-2 multi-variate inputs slightly increase sequence dimensionality but not sequence length, so

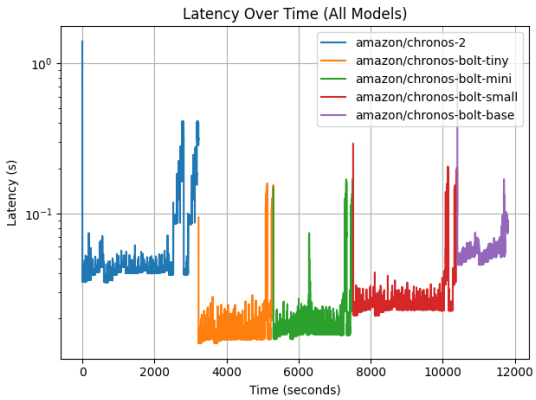

Figure 12: Latency for different model variants over the whole dataset with fixed context lengths and horizons on laptop CPU.

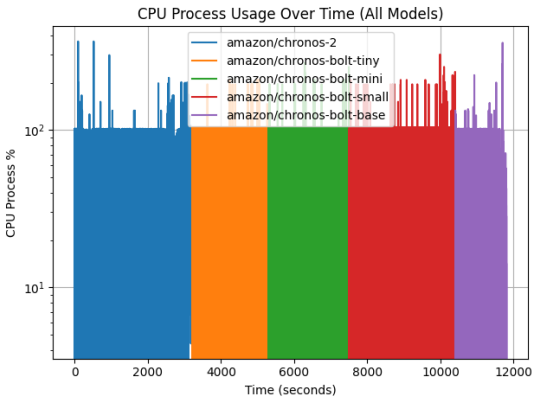

Figure 14: CPU usage for different model variants over the whole dataset with fixed context lengths and horizons on laptop CPU.

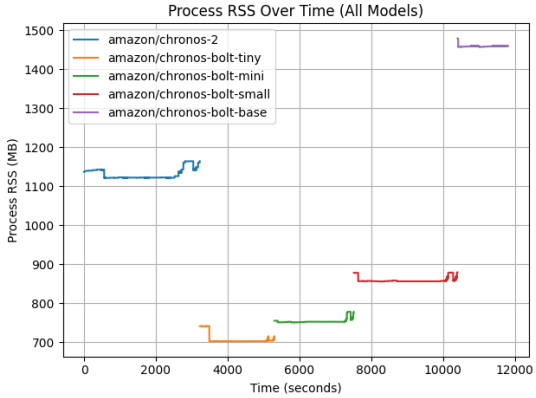

Figure 13: RAM usage for different model variants over the whole dataset with fixed context lengths and horizons on laptop CPU.

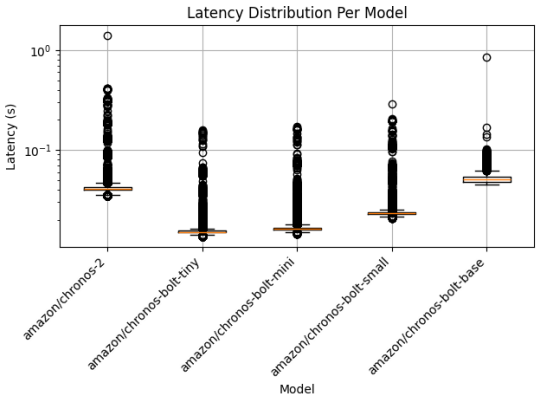

Figure 15: Latency distribution for different model variants over the whole dataset with fixed context lengths and horizons on laptop CPU.

latency overhead is modest. The RMSE for Gurgaon without using any covariates is summarised in the Table 2.

6 Part 4: Raspberry Pi Deployment

6.1 Hardware and Environment

We deploy the forecasting script on a Raspberry Pi (exact model and specs to be filled in), using CPU-only Chronos variants that fit in RAM. We pin the process to $k \in \{1, 2, 3, \ldots\}$ cores using taskset/affinity controls, and run continuous inference for 30 minutes.

	No Covariates	With Covariates
Gurgaon	23.62940583	22.43521243
Patna	23.06736581	25.19677393

Table 2: RMSE Scores with and without Covariates by Chronos2

We log:

- per-batch latency and throughput (windows per second),
- CPU utilization per core,
- process RSS,
- CPU temperature every minute (via `/sys/class/thermal` or Pi-specific tools).

6.2 Results and Discussion

Figure 25, 26, 24, 27, and 28 show a visualization of Raspberry Pi metrics for a single configuration.

As expected, the Pi exhibits higher latencies and more pronounced thermal throttling behavior under sustained load. Pinning to fewer cores reduces instantaneous throughput but can lower peak temperature and smooth thermal throttling over time.

Specification	Laptop (Ultra 7 155H)	Raspberry Pi 3
CPU Architecture	x86_64	ARM Cortex-A53
Cores / Threads	16 Cores / 22 Threads	4 Cores / 4 Threads
Base Frequency	3.80 GHz	1.20 GHz
L1 Cache	1.6 MB	∼128 KB
L2 Cache	18 MB	1 MB
L3 Cache	24 MB	None
System Memory	16 GB DDR5	1 GB LPDDR2
Memory Speed	7467 MT/s	∼900 MT/s
Memory Type	DDR5	LPDDR2
Hardware Reserved	595 MB	Shared with GPU
Thermal Throttling	Active (high headroom)	∼80°C
Power Class	45–65 W (class)	5–7 W

Table 3: Hardware Comparison: Laptop vs Raspberry Pi 3

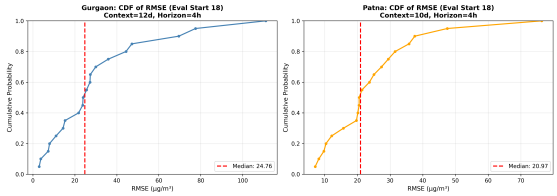

Figure 16: CDF of per-window RMSE, comparing univariate Chronos-2 ($PM_{2.5}$ only) between Patna and Gurgaon.

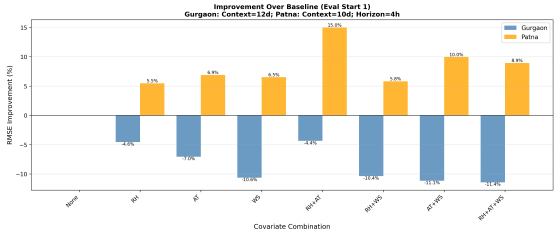

Figure 17: Improvement over Baseline for both the cities in all covariate combinations for a fixed selected window.

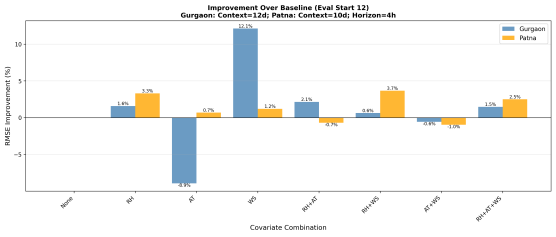

Figure 18: Improvement over Baseline for both the cities in all covariate combinations for a fixed selected window.

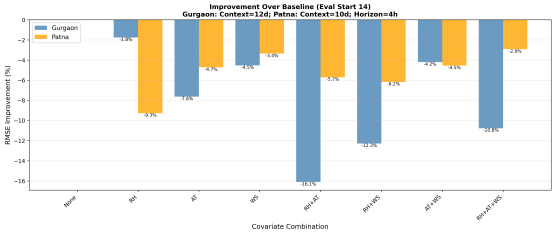

Figure 19: Improvement over Baseline for both the cities in all covariate combinations for a fixed selected window.

7 Part 5: Inference Server and LLM Optimizations

7.1 Server–Client Architecture

We implement a FastAPI-based server that exposes an endpoint:

- `POST /forecast`

The request schema includes:

- `request_id`: unique identifier,
- `city`: `"gurgaon"` or `"patna"`,
- `model_name`: Chronos-2, Bolt variant, and T5 variant
- `context_hours`: number of past hours,
- `horizon_hours`: forecast horizon,
- `use_covariates`: boolean flag.

The server maintains:

- a registry of loaded Chronos pipelines,
- a strict limit on the number of simultaneously loaded models (to avoid RAM exhaustion),
- per-model request queues and optional batching (e.g., up to 8 requests, waiting up to 100 ms).

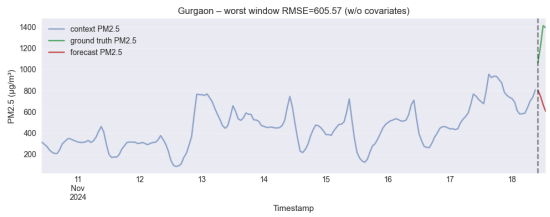

Figure 20: Example test day: ground truth vs. forecasted $PM_{2.5}$ without covariates

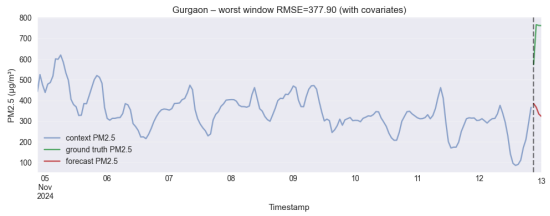

Figure 21: Example test day: ground truth vs. forecasted $PM_{2.5}$ with covariates

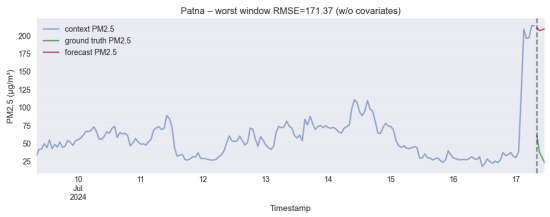

Figure 22: Example test day: ground truth vs. forecasted $PM_{2.5}$ without covariates

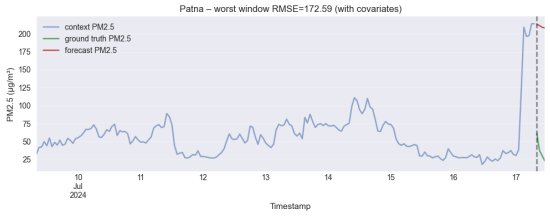

Figure 23: Example test day: ground truth vs. forecasted $PM_{2.5}$ with covariates

The client reads a synthetic trace of 500 requests and supports two modes:

1. **Burst mode:** send all 500 requests as quickly as possible.
2. **Poisson mode:** sample inter-arrival times from a Poisson process with rate λ requests per second and `await` accordingly.

The client logs send and receive timestamps to compute end-to-end latency; the server logs per-request:

- queue time (time spent waiting in the per-model queue),
- model load time,
- inference time (Chronos call),
- total server time.

7.2 Baseline and Optimized Variants

We implement and compare:

1. **Baseline server:** naive version that loads a model for each request, without batching or reuse.
2. **Cached server:** keeps a limited number of models resident and uses an LRU policy to evict when crossing the limit; requests for a loaded model skip loading cost.
3. **Batched server:** extends the cached server with per-model queues and batching windows (up to B requests or Δt wait).

These choices mirror common LLM-serving techniques at smaller scale:

- limiting the number of active models resembles serverless LLM schedulers that bound GPU memory usage(Gujarati et al., 2024);
- batching corresponds to token- or request-level batching in systems like Orca or Sarathi-Serve(Chen et al., 2024);

7.3 Latency Results Under Load

We replay the 500-request trace under both modes for each server variant and produce:

- latency-over-time plots (Figure 29, 30),
- RPS-over-time plots (Figure 31, 32),
- Plots of queue vs. load vs. inference time (Figure 33, 34),
- latency histograms over models (Figure 35, 36).

We see:

- Burst client to have very high queue times and long tails in latency.
- Poisson client to reduces both average and tail latency due to not overwhelming the server.

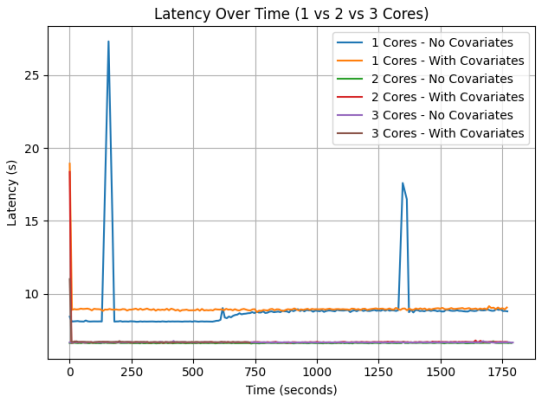

Figure 24: Raspberry Pi Latency vs Time for Chronos-2 using varying number of cores

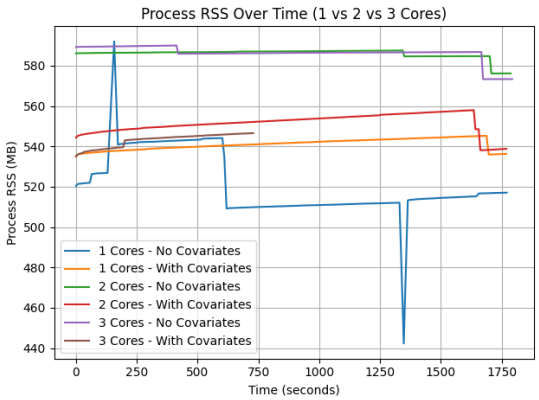

Figure 26: Raspberry Pi Process RSS vs Time for Chronos-2 using varying number of cores

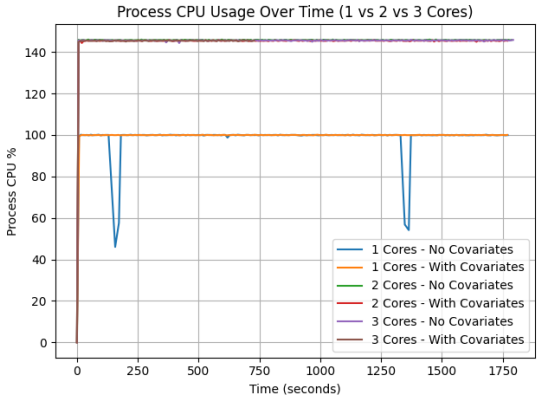

Figure 25: Raspberry Pi CPU% vs Time for Chronos-2 using varying number of cores

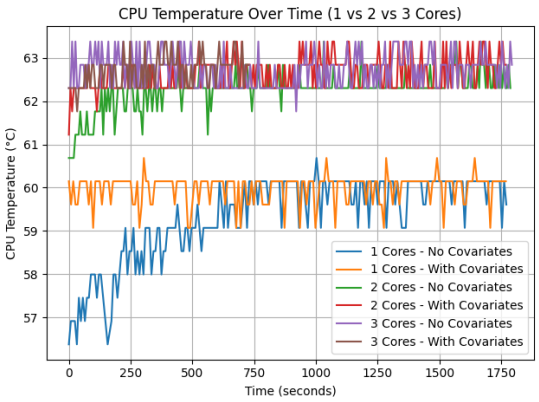

Figure 27: Raspberry Pi Temperature vs Time for Chronos-2 using varying number of cores

- Batching to reduce per-request inference cost at the expense of slight queueing increases the overall throughput.

7.4 Relation to LLM Optimization Literature

Although Chronos models are much smaller than production LLMs and run on CPU, several optimizations echo recent LLM systems work:

- **Batching and queueing:** similar to dynamic batching in Orca and Sarathi-Serve(Chen et al., 2024).
- **Model residency limits:** analogous to serverless LLM schedulers and model-caching strategies in ServerlessLLM and BlitzScale(Gujarati et al., 2024).

Due to the assignment constraints (CPU-only, fixed pre-trained Chronos models, Python implementation), we do not implement low-level attention kernel optimizations (e.g. FlashAttention, PagedAttention) and Quantization/Pruning (as the models were already provided by and used by the chronos library) but note that similar ideas could further reduce cost for long contexts.

8 Discussion

8.1 Accuracy vs. Runtime Choices

Our experiments highlight that good forecasting accuracy does not uniquely determine deployment choices. Key trade-offs include:

- For a fixed model, longer contexts usually improve RMSE up to a point, but increase per-batch latency roughly linearly with sequence length.
- Larger Chronos-2 models improve/worsen accuracy marginally compared to smaller Bolt variants but have significantly higher latency and memory usage.
- Meteorological covariates provide modest and unconsistent RMSE improvements and some-

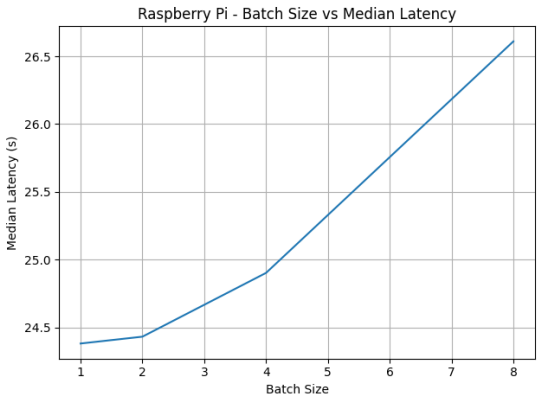

Figure 28: Raspberry Pi Batch Size vs the Median Latency for Chronos-2 using all cores

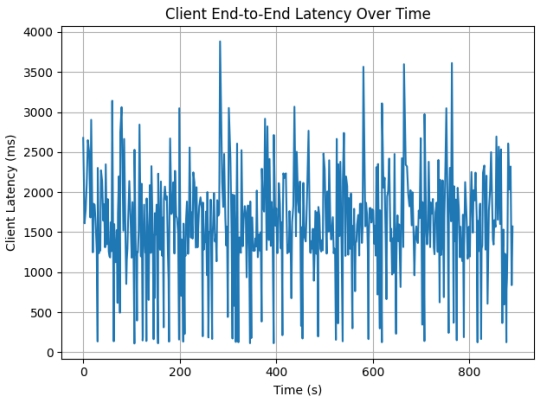

Figure 30: End-to-end latency over start time for Poisson requests by the client under a 500-request trace.

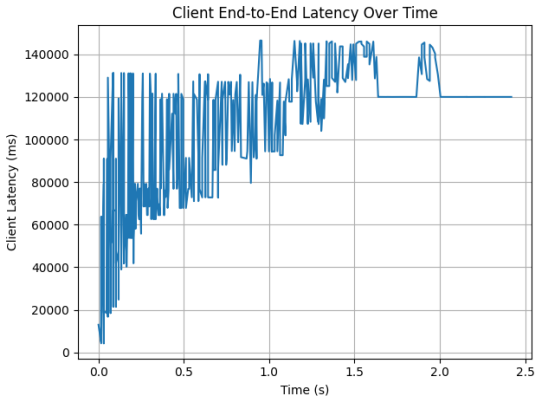

Figure 29: End-to-end latency over start time for Burst requests by the client under a 500-request trace.

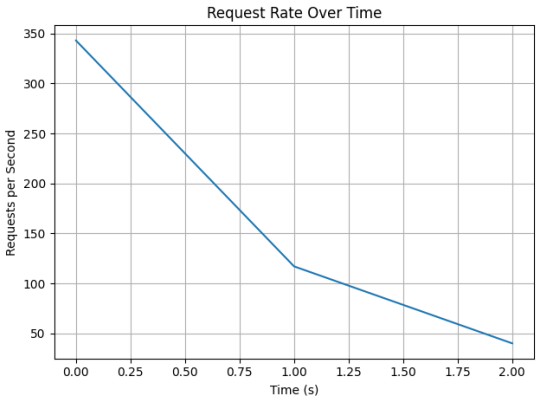

Figure 31: Requests per second (RPS) over time for burst client.

times worsen it on difficult test days with sharp changes, with minimal overhead in Chronos-2 multi-variate mode.

8.2 Laptop vs. Embedded Hardware

The same Chronos configuration behaves very differently on laptop vs. Raspberry Pi:

- On the laptop, CPU saturation and DRAM pressure are the main concerns; thermals remain manageable.
- On the Raspberry Pi, CPU temperature quickly becomes a limiting factor under continuous high load.
- Careful choice of context/horizon and possibly smaller Bolt variants are crucial for practical Pi deployments (e.g., personal exposure apps).

8.3 Server Behavior Under Load

Under synthetic traces, the inference server exhibits behavior similar to small-scale LLM serving:

- Tail latency is dominated by queue time in high-load scenarios; reducing load time (via caching) and inference time (via batching) indirectly reduces queueing.
- Batching improves throughput but must be balanced against added waiting time to form batches.
- Even simple policies—bounded number of loaded models, LRU eviction, small batching windows—yield large gains over a naive baseline.

Limitations

- **Zero-shot only.** We only use pre-trained Chronos models in zero-shot mode; fine-tuning on the $PM_{2.5}$ data could further improve accuracy but was out of scope.
- **Short trace lengths.** Our server experiments replay only 500 synthetic requests, which is

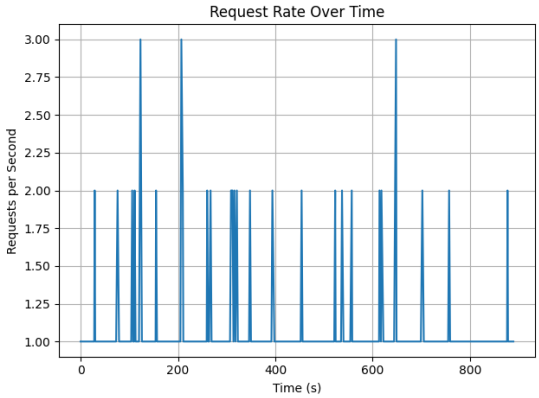

Figure 32: Requests per second (RPS) over time for Poisson client.

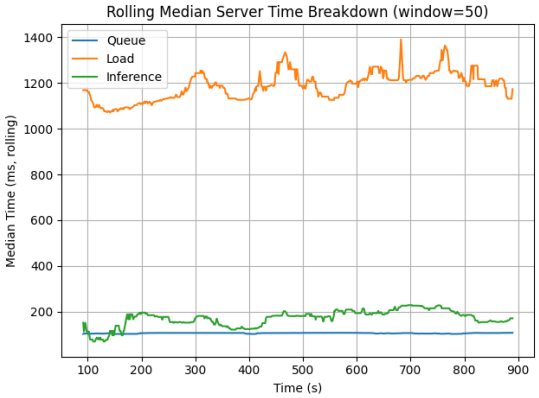

Figure 34: Breakdown of server time into queue, load, and inference components for poisson client.

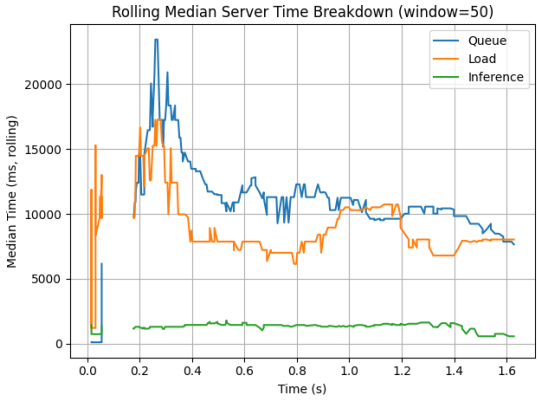

Figure 33: Breakdown of server time into queue, load, and inference components for burst client.

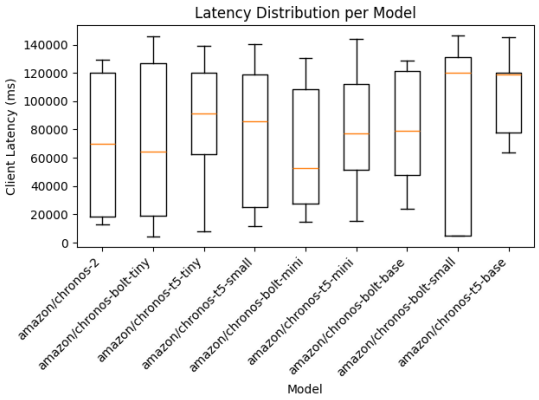

Figure 35: Distribution of end-to-end latency across 500 requests for burst client.

much smaller than real-world LLM traffic traces.

- **Ideal covariate forecasts.** For covariate experiments, we assume perfect future meteorological covariates from the CSV, which is optimistic compared to real forecasting scenarios.
- **CPU-only experiments.** GPU-based Chronos inference and more advanced kernel-level optimizations are not explored due to hardware constraints.

9 Conclusion

We presented an end-to-end study of $PM_{2.5}$ forecasting using the Chronos family of time-series LLMs, from accuracy evaluation across cities and hyper-parameters to detailed performance analysis on laptop and Raspberry Pi, and finally an inference server that exercises basic LLM-style optimizations under synthetic load.

Our main takeaways are:

- Chronos-2 models provide reasonable zero-shot $PM_{2.5}$ forecasts with clear sensitivity to context length and horizon; a 8–10 day context and 4–8 hour horizon often balance accuracy and cost.
- Meteorological covariates do not affect RMSE much, but sometimes for difficult days with sharp pollution changes it helps a little, with limited runtime overhead.
- Performance metrics (latency, CPU, memory, temperature) are crucial when deploying on constrained hardware such as Raspberry Pi; smaller models and careful hyper-parameter choices matter as much as raw accuracy.
- Simple serving-side optimizations—model caching, batching—can substantially reduce tail latency, echoing patterns seen in modern

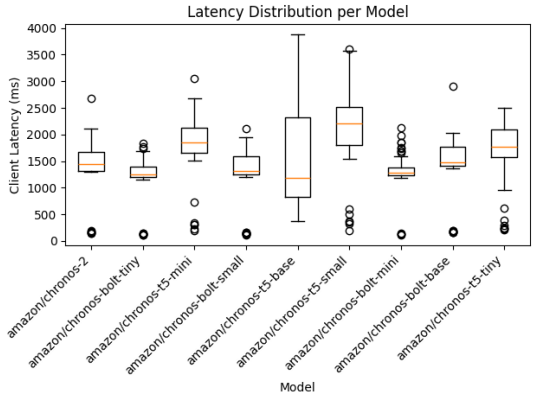

Figure 36: Distribution of end-to-end latency across 500 requests for poisson client.

LLM inference systems, even at the small scale of Chronos-2 on CPU.

References

Tianyi Chen and 1 others. 2024. Sarathi-serve: Efficient llm serving with chunked prefill and fine-grained scheduling. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. For fine-grained LLM scheduling inspiration.

Arjun Gujarati and 1 others. 2024. Serverlessllm: Low-latency serverless inference for large language models. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. For discussion of model loading and cold-start optimizations.