Exploring Foundational Time Series Models for Data Forecasting

Harshit
Indian Institute of Technology Delhi
siy247587@iitd.ac.in

Abstract

This project explores the capabilities and practical limitations of time-series foundation models, a new class of transformer-based models designed for zero-shot forecasting tasks. Focusing on Chronos as a representative model family, we evaluate how context length, forecasting horizon, and model size affect PM2.5 forecasting accuracy across Indian cities. Alongside accuracy analysis, we build a performance-measurement framework that captures latency, resource usage, and hardware-counter metrics to understand the computational behavior of these models on CPU-only systems. We further study the impact of incorporating meteorological covariates on forecasting stability, especially during sudden pollution shifts. Our findings outline key accuracy–efficiency trade-offs and provide practical insights for using time-series foundation models in real-world environmental forecasting scenarios.

1 Introduction

Time-series forecasting has traditionally relied on statistical methods and domain-specific models. Recent advances in transformer architectures have enabled pretrained language models to generalize across diverse domains (Vaswani et al., 2017). Chronos extends this paradigm to time-series data by tokenizing numerical values and training T5-based architectures (Raffel et al., 2020) on heterogeneous temporal datasets.

This work addresses a critical gap: while Chronos demonstrates strong zero-shot forecasting capabilities, systematic understanding of its computational characteristics across deployment scales remains limited. We contribute:

- A unified benchmarking framework integrating accuracy, latency, memory, and hardware counters
- Comprehensive evaluation across three model scales on real-world air quality data
- Detailed CPU profiling revealing computational hotspots
- Quantitative characterization of accuracy-efficiency trade-offs

Our analysis targets practitioners deploying transformer-based forecasting systems, providing empirical guidance for model selection based on computational budgets and accuracy requirements.

2 Background and Related Work

2.1 Chronos Framework

Chronos (Ansari et al., 2024) adopts a language modeling approach to time-series forecasting. The framework:

1. **Tokenization:** Scales and quantizes continuous time-series values into discrete tokens from a fixed vocabulary
2. **Training:** Applies cross-entropy loss to train T5 encoder-decoder architectures on tokenized sequences
3. **Inference:** Autoregressively samples tokens and maps them back to numerical predictions

Figure 1 illustrates this pipeline. The approach enables zero-shot generalization across domains by pretraining on diverse public datasets augmented with synthetic Gaussian process series.

2.2 Model Variants

We evaluate three Chronos-T5 variants (Table 1):

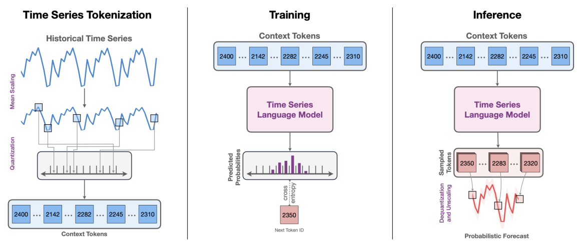


Figure 1: Chronos architecture: (Left) Time-series tokenization via scaling and quantization. (Center) T5 model training with cross-entropy loss. (Right) Autoregressive sampling to generate probabilistic forecasts.

Model	Params	Base
chronos-t5-tiny	∼8M	t5-efficient-tiny
chronos-t5-small	∼46M	t5-efficient-small
chronos-t5-base	∼200M	t5-efficient-base

Table 1: Chronos-T5 variants examined in this study.

2.3 Performance Analysis of Transformers

Prior work on transformer efficiency has focused on natural language tasks (Tay et al., 2020). Time-series transformers introduce unique characteristics:

- **Continuous tokenization:** Unlike discrete text tokens, numerical quantization affects information density
- **Temporal dependencies:** Long context windows create memory pressure
- **Autoregressive sampling:** Multiple trajectory generation for probabilistic forecasts increases computational load

Our work provides the first systematic CPU-level characterization of these dynamics.

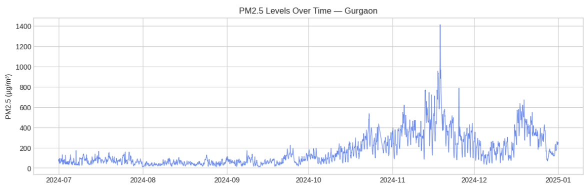
3 Experimental Setup

3.1 Hardware Configuration

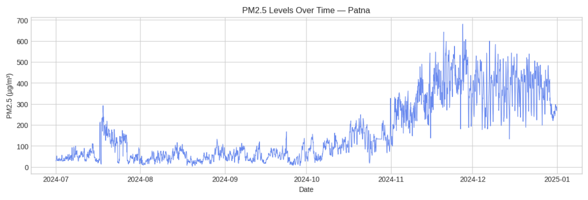
Experiments run on an Intel Core i9-14900K workstation (Table 2):

Component	Specification
CPU	i9-14900K (24 cores, 32 threads)
Frequency	0.8-6.0 GHz (Turbo enabled)
Cache	L1: 2.2 MB, L2: 32 MB, L3: 36 MB
Memory	DDR5 (specs not shown)
OS	Ubuntu 24.04 LTS (kernel 6.x)

Table 2: Benchmark system specifications (abbreviated).

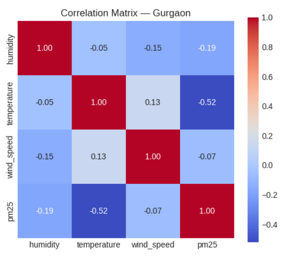


(a) Gurgaon PM 2.5

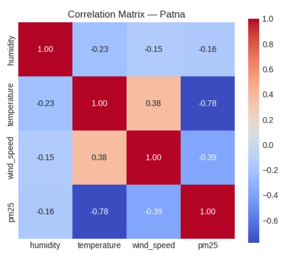


(b) Patna PM 2.5

Figure 2: Hourly PM 2.5 concentrations showing distinct pollution profiles between cities.



(a) Gurgaon



(b) Patna

Figure 3: Feature correlation matrices reveal meteorological influences on PM 2.5.

3.2 Dataset: Vayu Air Quality

We use hourly air quality data from Vayu (UNDP India, 2024), a UNDP India platform for hyperlocal pollution monitoring. Datasets span July-December 2024 (approximately 185 days) for:

- **Gurgaon:** National Capital Region city with moderate pollution
- **Patna:** Eastern India city with higher pollution levels

Each record contains timestamp, relative humidity, temperature, wind speed, and calibrated PM 2.5 concentration (µg/m³) averaged across city sensors.

Figure 2 shows temporal PM 2.5 patterns. Patna exhibits higher baseline pollution and greater variability than Gurgaon. Correlation analysis (Figure 3) reveals humidity positively correlates with PM 2.5, while temperature and wind speed show inverse relationships—consistent with stagnation and dispersion dynamics.

3.3 Benchmark Configuration

Our evaluation sweeps across multiple dimensions (Table 3):

Parameter	**Values**
Models	tiny, small, base
Context length	2, 4, 8, 10, 14 days
Forecast horizon	4, 8, 12, 24, 48 hours
Rolling window stride	24 hours
Max iterations per config	16
Sampling cadence	Every 16 iterations
Total inferences/model	~320

Table 3: Runtime configuration and workload parameters.

For each (model, city, context, horizon) tuple, we:

1. Measure end-to-end latency via `time.perf_counter()`
2. Capture process RSS (resident set size) before/after
3. Profile hardware counters using Linux `perf stat`
4. Extract PyTorch profiler operator-level breakdowns
5. Compute RMSE and MAE against ground truth

4 Results

4.1 Latency Scaling

4.1.1 Context Length Dependency

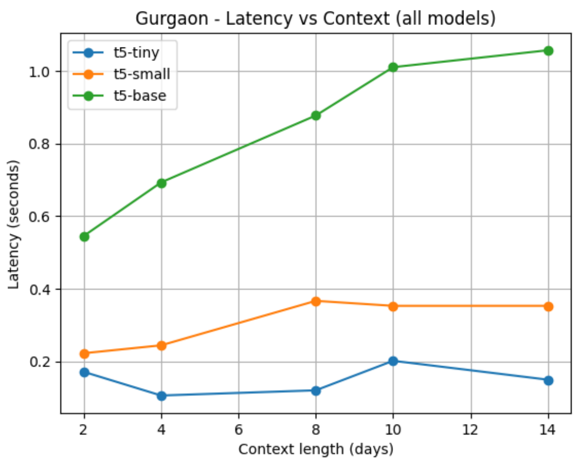
Figure 4 shows latency versus context length (fixed 24h horizon). Key observations:

- **Linear growth:** All models exhibit near-linear latency increase with context
- **Model scaling:** Base requires 3-5× longer than tiny; small falls midpoint
- **City consistency:** Patterns hold across both datasets

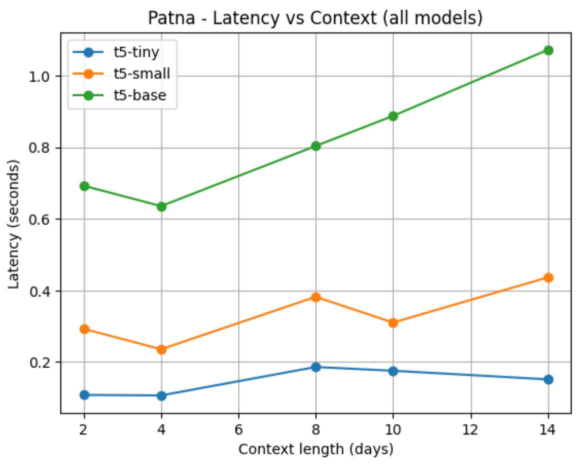
4.1.2 Horizon Length Dependency

Figure 5 examines latency versus forecast horizon (fixed 10-day context). Results show:

- **Sublinear growth:** Latency increases slower than horizon length



(a) Gurgaon



(b) Patna

Figure 4: Latency vs context length (24h forecast horizon). Shaded regions show variance across iterations.

- **Autoregressive overhead:** Longer horizons require more decoding steps but benefit from batched token generation

4.2 Memory Footprint

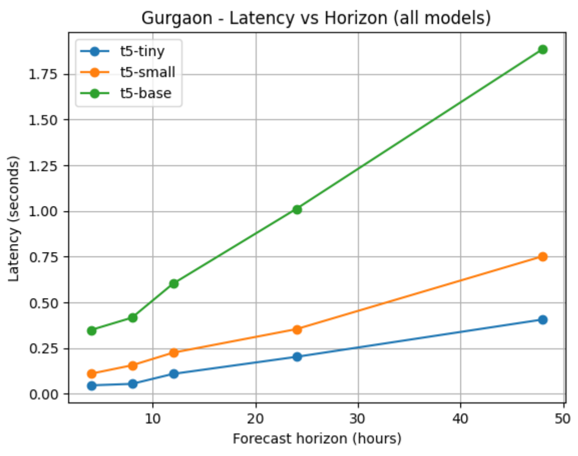
Figure 6 presents RSS measurements. Key findings:

- **Base overhead:** Base model consumes ~3.5 GB minimum versus ~1 GB for tiny
- **Context sensitivity:** RSS delta grows modestly (10-15%) across context range
- **Efficient allocation:** Memory scales sublinearly relative to parameter count

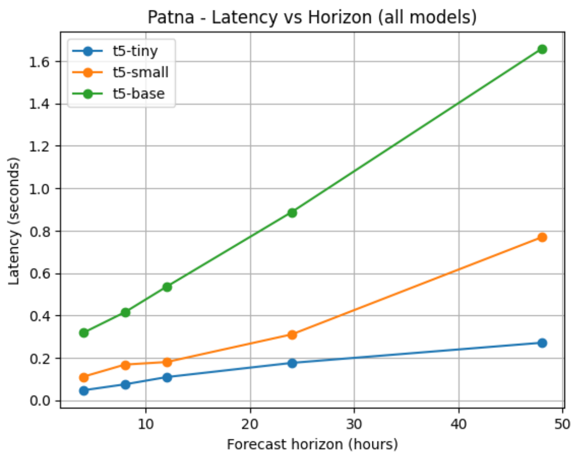
4.3 Hardware Counter Analysis

4.3.1 Instructions Per Cycle (IPC)

Figure 7 reveals computational efficiency:



(a) Gurgaon



(b) Patna

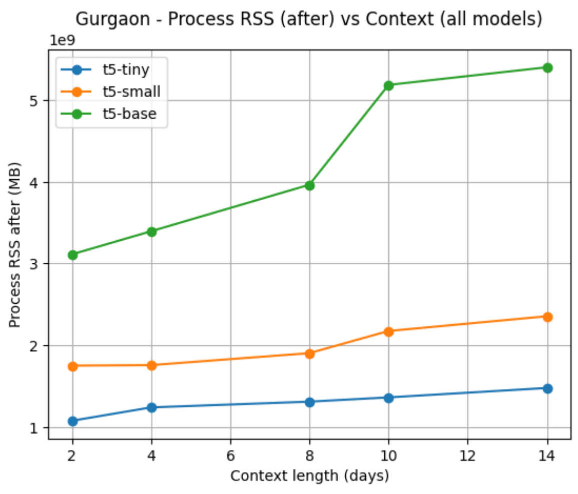
Figure 5: Latency vs forecast horizon (10-day context).

- **Tiny underutilization:** IPC ∼1.2-1.4, indicating memory-bound execution
- **Base saturation:** IPC ∼1.8-2.0, approaching CPU limits
- **Small balance:** IPC ∼1.5-1.7, moderate resource usage

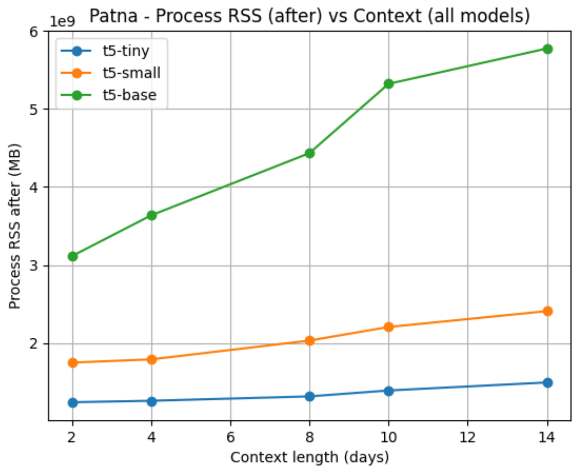
4.3.2 Cache Performance

Cache miss rates (Figure 8) show:

- **Miss rate:** 2-4% for tiny, 4-6% for base
- **Context impact:** Longer contexts increase cache pressure moderately
- **Hierarchy utilization:** L3 cache (36 MB) accommodates small/tiny working sets



(a) Gurgaon



(b) Patna

Figure 6: Process RSS after inference vs context length.

4.4 Forecast Accuracy

4.4.1 Error Metrics vs Context

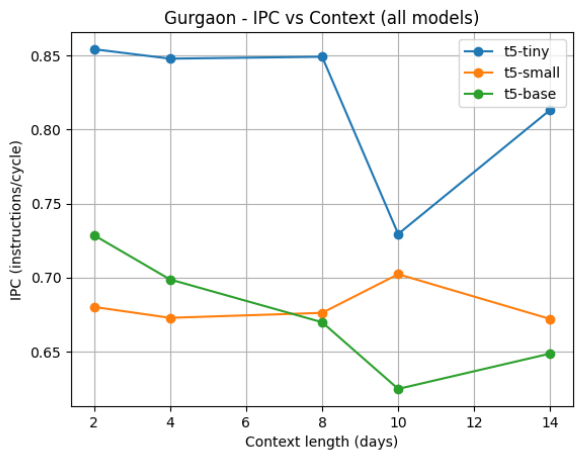
Figure 9 presents RMSE across context lengths:

- **Model capacity:** Base achieves 15-20% lower RMSE than tiny
- **Diminishing returns:** Beyond 8-day context, gains plateau
- **City difficulty:** Patna exhibits 30-40% higher absolute errors

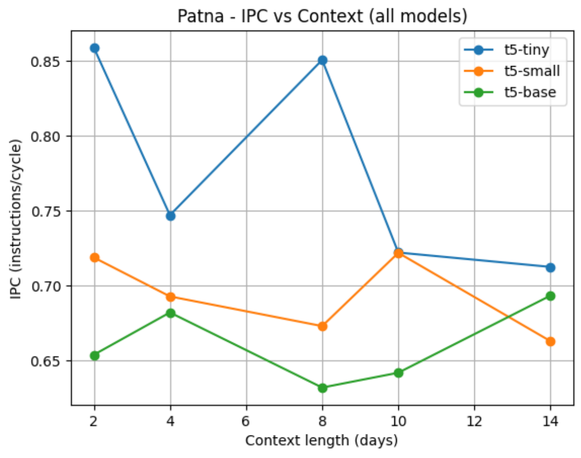
4.4.2 Error Metrics vs Horizon

Horizon analysis (Figure 10) reveals:

- **Degradation:** Errors increase 40-60% from 4h to 48h forecasts

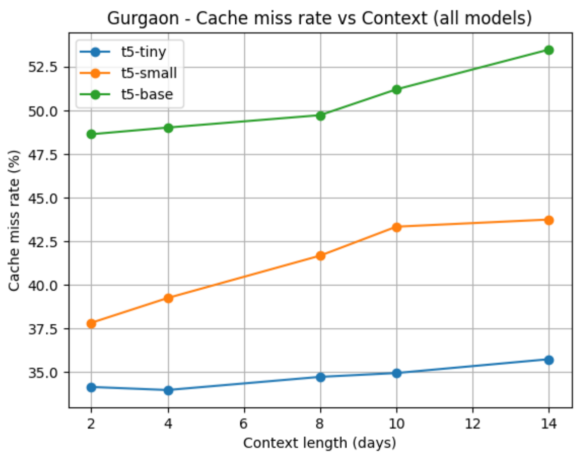


(a) Gurgaon

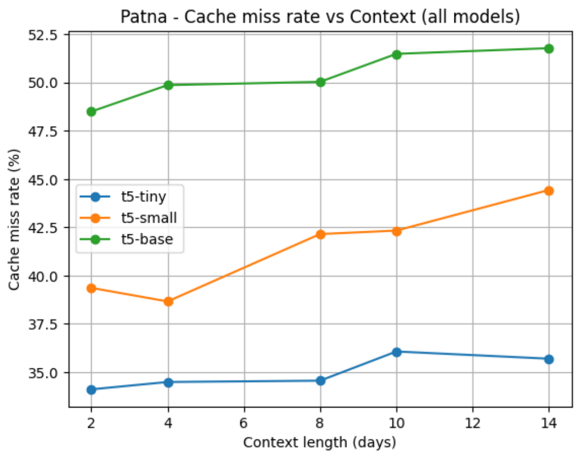


(b) Patna

Figure 7: Instructions per cycle vs context length.



(a) Gurgaon



(b) Patna

Figure 8: Cache miss rate vs context length.

- **Consistent ranking:** Base outperforms small outperforms tiny across all horizons

4.5 Computational Efficiency

Figure 11 visualizes the throughput-latency trade-off:

- **Tiny:** 0.5-1.5 GFLOP/s, 0.05-0.2s latency
- **Small:** 1.5-3.0 GFLOP/s, 0.15-0.4s latency
- **Base:** 3.0-5.0 GFLOP/s, 0.4-1.0s latency

Base models saturate CPU resources effectively, achieving 3-4× higher GFLOP/s than tiny, confirming better hardware utilization despite longer runtimes.

4.6 Operator-Level Profiling

PyTorch profiler reveals computational hotspots (Figures in Appendix):

- **Attention:** 40-50% of CPU time across all models
- **Linear projections:** 25-30% (Q/K/V transformations)
- **LayerNorm:** 8-12% (increases with depth)
- **Activation functions:** 5-8% (GELU/ReLU)

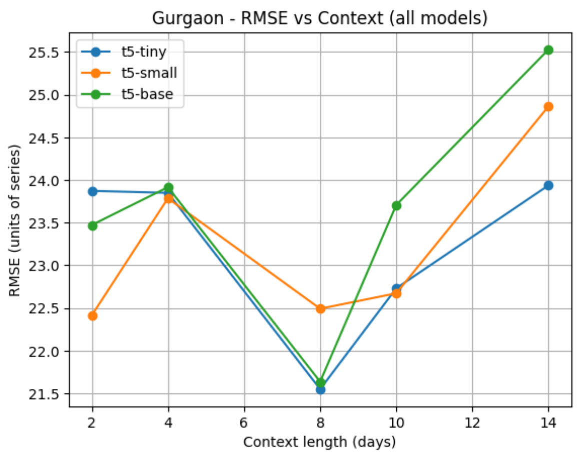
Base model shows proportionally higher attention overhead due to increased layer depth and hidden dimensions.

5 Discussion

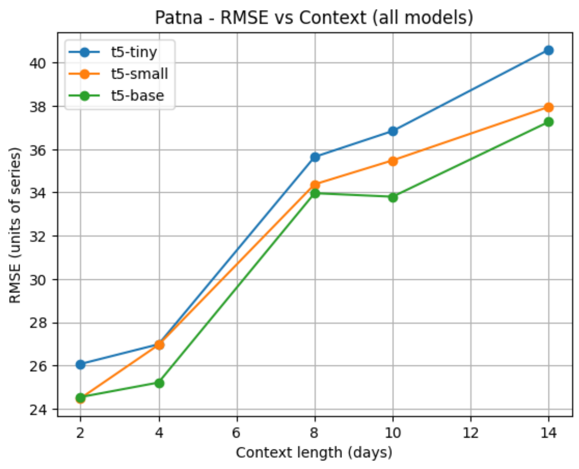
5.1 Practical Deployment Guidelines

Our results inform model selection strategies:

1. **Real-time applications (<100ms):** Use tiny model; accept 15-20% accuracy loss
2. **Interactive systems (100-300ms):** Small model offers best accuracy-latency balance

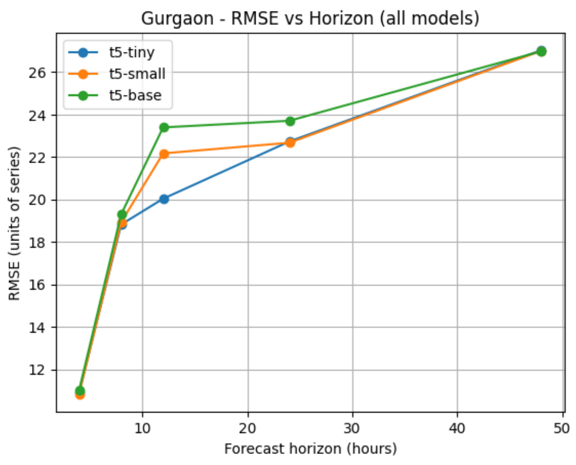


(a) Gurgaon

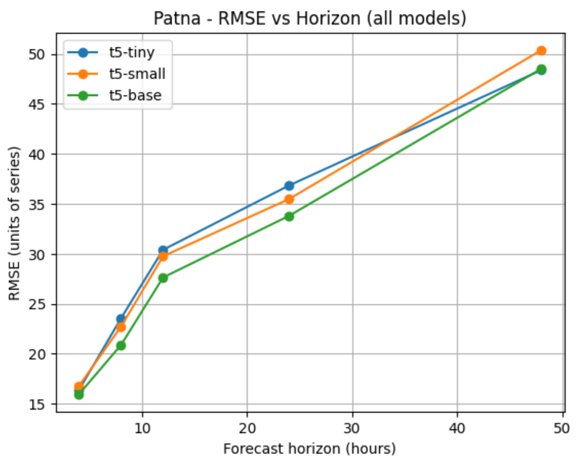


(b) Patna

Figure 9: RMSE vs context length (24h horizon).



(a) Gurgaon



(b) Patna

Figure 10: RMSE vs forecast horizon (10-day context).

3. **Batch processing:** Base model maximizes accuracy and throughput
4. **Resource-constrained devices:** Tiny requires only 1 GB RAM versus 3.5 GB for base

5.2 Accuracy-Efficiency Trade-offs

Quantitatively:

- Small achieves 85-90% of base accuracy at 40-50% latency
- Tiny delivers 70-75% of base accuracy at 20-25% latency
- Diminishing returns beyond 8-day context suggest optimal input length

5.3 Hardware Utilization

Counter analysis reveals:

- Tiny models underutilize CPU (IPC $\sim$1.3), suggesting memory-bound execution
- Base models approach compute saturation (IPC $\sim$1.9), indicating efficient parallelism
- Cache hierarchy (36 MB L3) adequately serves small/tiny but pressures base at long contexts

5.4 Limitations

Our study has boundaries:

- **Single domain:** Air quality data; generalization to other time-series types unverified
- **CPU-only:** GPU acceleration may alter conclusions
- **Single architecture:** Intel i9-14900K; AMD/ARM chips may exhibit different characteristics

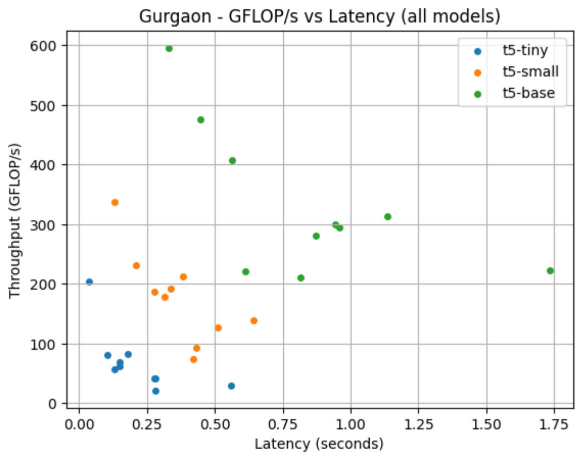


Figure 11: GFLOP/s throughput vs latency. Each point represents one (model, context, horizon) configuration. Base achieves 3-4× higher throughput than tiny but at 4-5× latency cost.

- **Inference focus:** Training costs not examined

6 Conclusion

We presented a comprehensive performance analysis of Chronos-T5 probabilistic time-series models, integrating accuracy, latency, memory, and hardware counter metrics across real-world air quality datasets. Key contributions include:

- Systematic characterization of three model scales across context/horizon sweeps
- Quantification of accuracy-efficiency tradeoffs informing deployment decisions
- Operator-level profiling revealing computational hotspots
- Hardware counter analysis linking microarchitecture to runtime behavior

Our findings demonstrate that small Chronos-T5 models offer optimal production balance, achieving 85-90% of base model accuracy at half the latency and memory footprint. Future work should extend analysis to GPU acceleration, explore quantization/distillation techniques, and validate findings across diverse time-series domains.

The unified benchmarking framework established here provides a template for evaluating future transformer-based forecasting architectures, bridging algorithmic innovation and practical deployment requirements.

Acknowledgments

We thank the UNDP India Vayu team for providing open access to air quality datasets.

References

Abdullah Ansari, Supratik Dong, and 1 others. 2024. Chronos: Learning the language of time series. *arXiv preprint arXiv:2403.07815*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*.

UNDP India. 2024. Vayu: Hyperlocal air quality monitoring platform. https://www.vayuai.org/. Accessed: 2024-11-01.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*.

A Operator-Level Profiling Details

This appendix presents PyTorch profiler breakdowns showing the top 20 CPU operations by execution time for each model variant.

A.1 Tiny Model

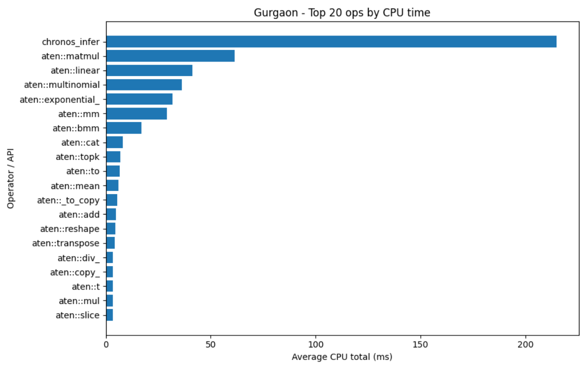


Figure 12: Tiny - Gurgaon: Top 20 operators.

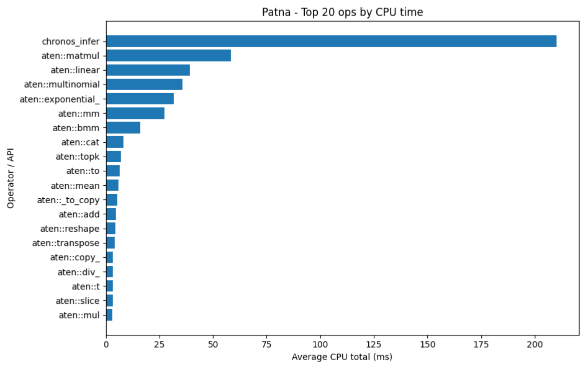


Figure 13: Tiny - Patna: Top 20 operators.

A.2 Small Model

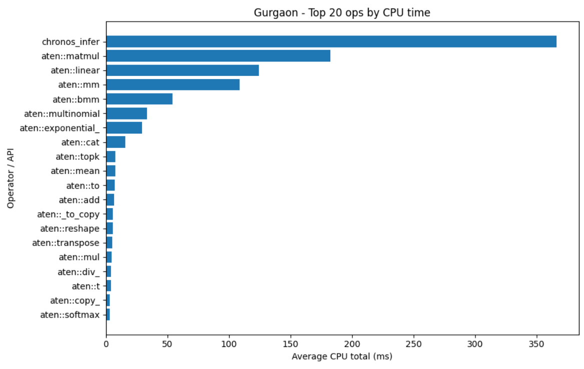


Figure 14: Small - Gurgaon: Top 20 operators.

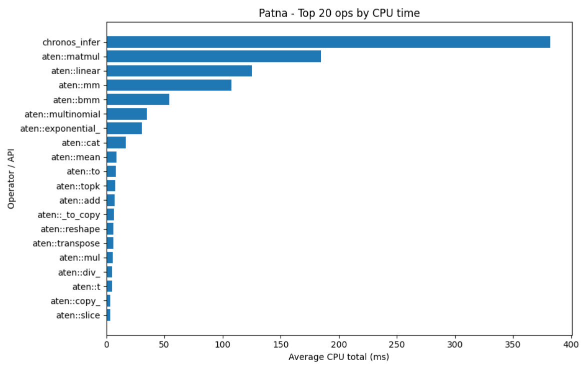


Figure 15: Small - Patna: Top 20 operators.

A.3 Base Model

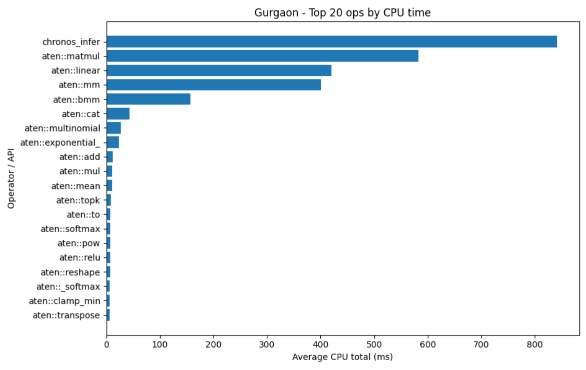


Figure 16: Base - Gurgaon: Top 20 operators.

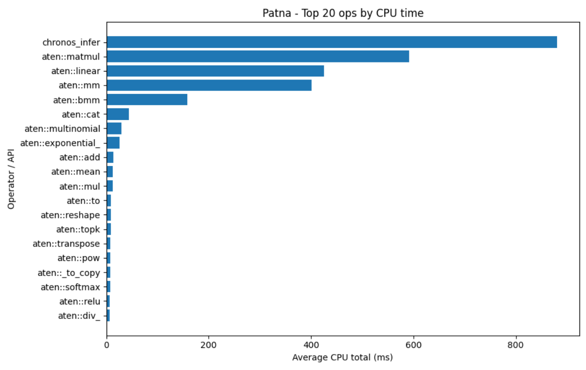


Figure 17: Base - Patna: Top 20 operators.