
Sustainability of TSFM Deployment and Pretraining: From Carbon Footprint Analysis to Mitigation via Pruning

Cécile Luc

Strategy & Innovation, AI Hub
Schneider Electric, Rueil-Malmaison, France
cecile.luc92@gmail.com

Nathanaël Lemonnier

Strategy & Innovation, AI Hub
Schneider Electric, Grenoble, France
nathanael.lemonnier@se.com

Ievgen Redko

Strategy & Innovation, AI Hub
Schneider Electric, Rueil-Malmaison, France
ievgen.redko@se.com

Abstract

Time Series Foundation Models (TSFMs) offer strong zero-shot forecasting capabilities, but their high pretraining cost raises a practical sustainability question: does this upfront investment pay off over a full deployment lifecycle, compared to supervised standalone models that must be periodically retrained? We model the cumulative carbon cost of different approaches across a realistic industrial scenario: Chronos-2 amortizes its pretraining cost within months for large or frequently queried deployments, but a classical model remains cheaper indefinitely for small or rarely queried ones. For cases where reducing the TSFM’s own cost matters, we show that removing a single encoder block (7.9% of parameters) via a gradient-free pruning criterion yields a consistent 6 to 9% reduction in FLOPs, CO₂ emissions, and latency, at essentially no cost to zero-shot forecasting performance across a 25-task benchmark.

1 Introduction

Time Series Foundation Models (TSFMs) such as Chronos-2 [Ansari et al., 2025], TimesFM [Das et al., 2024], and Moirai [Woo et al., 2024] have shown strong zero-shot generalization capabilities across a range of tasks, including forecasting, anomaly detection, and classification. This generalization capability comes at a price that is rarely quantified: pretraining a TSFM on massive datasets requires substantial computing resources, with an associated energy and carbon cost, paid once, upfront, unlike task-specific deep learning or classical models (e.g., PatchTST [Nie et al., 2023] or LightGBM [Ke et al., 2017]), which must be periodically retrained, per asset, to follow the evolution of the data.

Carbon accounting for machine learning has received growing attention, from early estimates of the energy footprint of NLP model training ([Strubell et al., 2019]), to broader calls for treating computational efficiency as a first-class evaluation criterion alongside accuracy ([Schwartz et al., 2020]). However, to our knowledge, no existing study compares the cumulative carbon cost of TSFMs against specialized models across a full deployment lifecycle: pretraining, fine-tuning or periodic retraining, and inference. If a TSFM’s upfront cost cannot be amortized over time, it can instead be reduced directly. A common approach is pruning, which removes blocks of a pretrained model to reduce computational and memory costs with limited loss in performance.

Contributions.

1. **A lifecycle carbon cost comparison** of TSFMs against two representative alternatives: LightGBM, a classical gradient-boosted tree model, and PatchTST, a task-specific deep learning forecaster to determine whether a TSFM’s pretraining cost is amortized over a realistic deployment horizon.
2. **A structured pruning study of Chronos-2**, showing that removing a single encoder block via a one-shot cosine similarity criterion reduces inference cost by 6 to 9% with no measurable loss in zero-shot forecasting performance.
3. **A multi-dimensional cost characterization** of both the full and pruned models in terms of FLOPs, CO₂ emissions, and latency.

2 Background and Related Work

Time Series Foundation Model. TSFM architectures are highly diverse, and these differences directly affect their computational cost. Our study includes Chronos-2, an encoder-only transformer, built on the T5 design, with a group attention layer that aggregates information across time series within a batch, enabling zero-shot, multivariate, in-context forecasts without any task-specific training. This comes at a cost: self-attention scales quadratically with context length, on top of the group attention overhead. Chronos-2 does, however, produce the full forecast horizon in a single forward pass (for prediction lengths of up to 1024), unlike earlier autoregressive TSFMs such as TimesFM in its first version [Das et al., 2024], which generated forecasts patch by patch in several steps. Regarding the size, Chronos-2 has around 120M parameters. Other recent TSFMs are much smaller: FlowState, for instance, uses under 10M parameters by replacing the transformer with a state-space encoder [Graf et al., 2026]. Naturally, the cost numbers we report for Chronos-2 cannot represent TSFMs universally, but they represent our baseline for this paper.

Carbon Footprint and Sustainability. We use CodeCarbon [Schmidt et al., 2021] to quantify the environmental cost of each approach, available as a Python package that estimates CO₂-equivalent (CO₂eq) emissions from the energy consumed by the underlying hardware, converted using the carbon intensity of the local electricity grid. A common alternative is to count FLOPs, the number of arithmetic operations a model performs. Unlike carbon cost, this figure does not depend on the hardware or software stack used, which makes it a useful way to compare architectures independently of any specific measurement setup.

Model Compression for Time Series. Pruning is a common method to reduce computation and memory footprint of a model after training, both directly relevant for the sustainability focus of our study. Pruning removes parts of a network, weights, neurons, channels, or entire layers, to reduce it. We distinguish two different types of pruning. Unstructured pruning removes individual weights, but the resulting sparse matrices need specialized hardware or software to actually run faster. Structured pruning removes whole components instead, such as attention heads or channels, which speeds things up on regular hardware without any extra tooling (see [Zhu et al., 2024]). It has been studied for LLMs, but far less so for TSFMs. The study from Zhao et al. [2025] is, to our knowledge, the closest prior work applying pruning to a TSFM. While the motivation of this paper is different, the authors prune neurons before fine-tuning, specifically to improve downstream task performance, based on the idea that a given forecasting task only needs a fraction of a TSFM’s parameters. Our goal is not to improve task-specific performance through pruning, but to reduce cost while preserving the model’s original zero-shot forecasting ability, without any task-specific fine-tuning.

3 Carbon cost of TSFM Deployment

3.1 Deployment scenario and cost model

Problem setup. We model a production forecasting system deployed hourly across N_a industrial assets, for instance UPS units in a data center or electrical transformers. We consider a baseline scenario: a 24-hour forecast is requested for each asset every day, and models are periodically retrained or fine-tuned to keep up with data evolution. The full notation used in this section is given in the appendix.

We consider two supervised methods as alternatives to a TSFM. The first one is LightGBM [Ke et al., 2017], a classical gradient-boosted tree model (300 estimators in our configuration ; a fixed budget chosen without hyperparameter tuning, see Appendix E), called as a rolling forecast hour by hour, each call receiving a window of $L = 168$ real sensor values, so a 24-hour forecast costs $H = 24$ times a single call. We suppose it is a realistic use of a classical, lightweight model like LightGBM. The second is PatchTST [Nie et al., 2023], a patch-based transformer for time series forecasting (around 620K parameters in our configuration), which, like Chronos-2, is natively multi-step and produces the full 24-hour forecast in a single call.

Cost model. We now introduce our cost model for all models considered above. LightGBM’s cumulative carbon cost, as a function of time t (months since deployment) and number of assets N_a , is:

$$\text{ML}(t, N_a) = \sum_{j=1}^{N_a} \sum_{i=0}^{k_{\text{tr}}} \alpha W_i^{(j)} + \lfloor t \rfloor N_a n_{\text{inf}} H C_{\text{inf}}^{\text{C}} \quad (1)$$

with $k_{\text{tr}} = \lfloor (t - 1)/T_{\text{tr}} \rfloor$ for $t \geq 1$, and $\text{ML}(0, N_a) = 0$. The double sum adds up the cost of every retraining that has happened so far: one LightGBM model is trained from scratch per asset, then retrained every T_{tr} months, and each of these retrains costs α (the carbon cost of training on a single window) times the size of the retraining window at that point, $W_i^{(j)}$. The remaining term is the inference cost: n_{inf} is the number of 24-hour forecasts requested per month and per asset, and $C_{\text{inf}}^{\text{C}}$ is the cost of a single LightGBM call, so that each forecast costs $H \times C_{\text{inf}}^{\text{C}}$, as described above.

Chronos-2 using zero-shot, by contrast, is never retrained:

$$\text{TSFM}_{\text{ZS}}(t, N_a) = C_{\text{pr}} + \lfloor t \rfloor N_a n_{\text{inf}} C_{\text{inf}}^{\text{T}} \quad (2)$$

C_{pr} , the cost of pretraining, is paid once and does not depend on N_a , and $C_{\text{inf}}^{\text{T}}$ is the cost of a single Chronos-2 call, which already covers the full 24-hour forecast. PatchTST follows the same logic as LightGBM but without the H factor, since it is also natively multi-step, and Chronos-2 with LoRA or full fine-tuning follows the same logic as the zero-shot case with an added periodic fine-tuning term. We give the full equations, parameter notation (Table 4), and their measured values (Table 7) in the appendix (Section A).

Measurement protocol. Carbon is measured with CodeCarbon [Schmidt et al., 2021] at the local grid carbon intensity $56 \text{ gCO}_2/\text{kWh}$, to which all values are **proportional**. Chronos-2’s undisclosed **pretraining cost is extrapolated from one measured step**, hence a **lower bound**. For details about those two points, see Appendix B.

3.2 Results

Raw measures. Table 1 reports the measured carbon cost and FLOPs for each approach. We note that Chronos-2, despite being a foundation model, is one order of magnitude more carbon-intensive at inference than both supervised baselines. PatchTST is on par with LightGBM with respect to this metric as the latter provides forecasts for 1 step at a time. Finally, the full pre-training of Chronos-2 for 200,000 steps would roughly make it pre-training 60 times more carbon-intensive than PatchTST.

Baseline comparison. Figure 1 shows that, under the baseline scenario, LightGBM starts at zero cost and grows steadily, with small jumps every 6 months from periodic retraining, while Chronos-2 ZeroShot starts higher, from paying C_{pr} upfront, but grows more slowly, so eventually, the two curves cross. PatchTST’s training cost per window, α_{ptst} , is orders of magnitude higher than LightGBM’s (Table 7 in Appendix). It exceeds the plotted range within a few months, or a few dozen assets, and never becomes competitive with LightGBM or Chronos-2 under this scenario.

Table 1: Carbon cost and FLOPs per approach (LightGBM: CPU; others: GPU). Full measurements and details in Appendix B.

Model	Operation	Carbon (gCO ₂)	FLOPs
LightGBM	Training	7.2×10^{-3}	5.3×10^{10}
	Inference	6.8×10^{-6}	2,973
PatchTST	Training	1.08	5.5×10^{14}
	Inference	6.6×10^{-6}	7.6×10^8
Chronos-2	Pretraining (1 step)	3.3×10^{-4}	1.5×10^{12}
	Inference	6.1×10^{-5}	1.5×10^9

This assumes PatchTST is trained per asset; a PatchTST shared across similar assets would not scale with N_a in training cost (Appendix E).

Where this crossover happens depends on both N_a and t : at $N_a = 1000$, Chronos-2 becomes cheaper after 18 months; at $N_a = 5000$, after less than 3 months; at $N_a = 100$, only after 180 months (15 years); at $N_a = 10$, it does not become cheaper even within 100 years (see Appendix, Figure 6, for the full curve). For a small number of assets or a short deployment, a classical model stays cheaper; the TSFM only pays off once the number of assets or the usage duration is large enough to amortize C_{pr} .

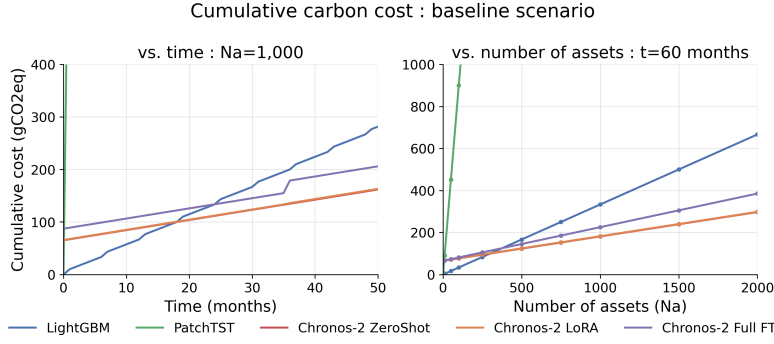


Figure 1: Cumulative carbon cost for all five approaches, baseline scenario, versus time at $N_a = 1000$ (left) and versus number of assets at $t = 60$ months (right).

Sensitivity to deployment context. We test how sensitive this break-even point is to two deployment parameters: (i) how often LightGBM is retrained and Chronos-2 is fine-tuned (T_{tr} , T_{ft} , in months), modeling the need to retrain more or less often as data drifts, and (ii) how often a forecast is requested (n_{inf}). We consider four retraining/fine-tuning scenarios: severe drift ($T_{tr} = 1$, $T_{ft} = 6$), baseline ($T_{tr} = 6$, $T_{ft} = 36$), mild drift ($T_{tr} = 12$, $T_{ft} = 60$), and a utopia scenario where neither model is ever retrained, used as an asymptotic reference. We also vary inference frequency across three scenarios: hourly ($n_{inf} = 720/\text{month}$), daily ($n_{inf} = 30/\text{month}$), and weekly ($n_{inf} \approx 4/\text{month}$).

We look at the drift scenarios under two variants of Chronos-2 (Figure 2): ZeroShot, whose cost equation has no fine-tuning term at all, and full fine-tuning, which occurs every T_{ft} months like LightGBM is retrained every T_{tr} months. This lets us separate two effects: how much LightGBM’s own retraining cost changes with drift, and how much Chronos-2’s break-even point shifts once it also has to pay a periodic fine-tuning cost to keep up.

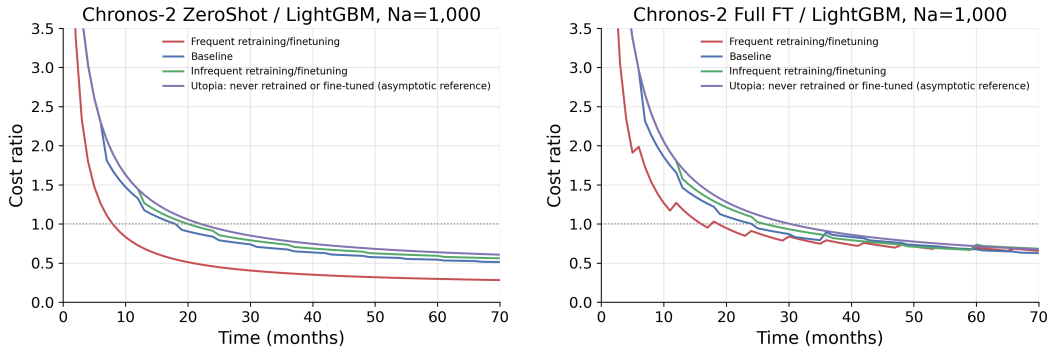


Figure 2: Cost ratio of Chronos-2 ZeroShot (left) and full fine-tuning (right) over LightGBM, at $N_a = 1,000$, under four retraining/fine-tuning scenarios.

At $N_a = 1000$, ZeroShot’s break-even usage duration (where the lines crosses 1) shifts from 8 months under severe drift, to 22 under utopia (18 in the baseline, 20 under mild drift), purely from LightGBM’s retraining cost changing. Full fine-tuning follows the same trend, shifted later by roughly 6-8 months in every scenario, the added cost of keeping Chronos-2 itself up to date (Figure 2).

Inference frequency has a much larger effect (Figure 3): Chronos-2 ZeroShot is cheaper within the first month under hourly requests, at 18 months under baseline, and only after 49 months under weekly requests. **How often a forecast is requested matters more to carbon cost than how often the model is kept up to date.**

Carbon cost based on the frequency of inference : $N_a=1,000$

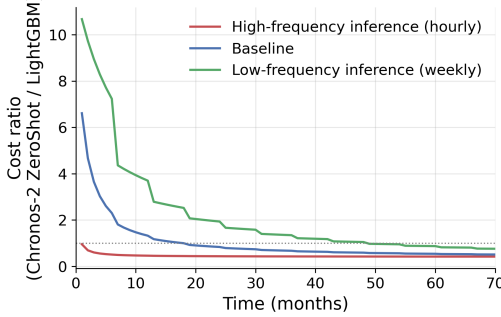


Figure 3: Cost ratio of Chronos-2 ZeroShot over LightGBM, at $N_a = 1,000$, under three inference frequency scenarios.

Overall, a small or rarely queried set of assets stays cheaper with a classical model over any realistic usage duration, while a large or frequently queried one amortizes Chronos-2’s pretraining cost within a few months. Cases in between depend on the specific usage duration, number of assets, and query frequency, and should be read against Figures 1–3. When the TSFM does not amortize its pretraining cost, reducing its inference cost directly becomes the question, which we turn to below.

4 Model Compression with Pruning

Having established Chronos-2’s carbon footprint relative to classical models, we now investigate whether that footprint can be reduced through model compression, and at what cost to accuracy. We focus on structured pruning: removing entire encoder blocks from the pretrained model, without any fine-tuning, so that any efficiency gain comes purely from a smaller architecture applied zero-shot. Concretely, we ask: (i) can we identify redundant encoder blocks using a lightweight, gradient-free signal; (ii) how much of the model can be removed before accuracy degrades meaningfully; and (iii) does the resulting model meaningfully reduce inference-time CO₂, latency, and compute cost.

4.1 Problem setup

Pruning method. Chronos-2’s encoder is composed of 12 transformer blocks, each alternating time attention and group attention sub-layers followed by a feed-forward network. We use structured block-level pruning based on the cosine similarity between the input and output of each block, following outcomes from Gromov et al. [2024]. A score close to 1 means the block’s output is nearly identical to its input (i.e., it applies a near-identity transformation) and is a strong removal candidate. The original method targets causal decoder models and only looks at the last token. As Chronos-2 is an encoder-only, bidirectional architecture, we instead average the cosine similarity across all context positions for a more robust per-block score (the input to each block is a patch embedding covering multiple raw time steps, hence multiple positions). Removing a block removes its time attention, group attention, and feed-forward network (FFN) sub-layers all at once, so no single mechanism can be isolated at this granularity. This scoring gives a redundancy ranking of the 12 blocks. Given a compression budget of k blocks, we remove the k highest-ranked (most redundant) blocks as a group; the ranking is computed once, on a dedicated calibration set, and reused for every k we test.

Dataset. We evaluate using 27 zero-shot forecasting tasks spanning diverse domains and frequencies (30 min to yearly) from *Chronos Benchmark II* [Ansari et al., 2024]. None of the tasks were seen during Chronos-2 pretraining. Task-level statistics are reported in Table 10 (Appendix). Datasets are downloaded from HuggingFace (autogluon/chronos_datasets). Two tasks (ETT 15-min, ETT hourly) are excluded due to a loading incompatibility (deprecated *dataset script*) with the current datasets library. Our evaluation covers the remaining 25 tasks.

Calibration and configuration selection. Six tasks are used for calibration (*ercot*, *exchange_rate*, *monash_hospital*, *dominick*, *monash_m3_monthly*, *m4_yearly*), covering a range of domains, frequencies, and context lengths. For each, up to 30 series are sampled by history length, with early stopping

once the median cosine score stabilizes ($<1\%$ change over 5 series). Scores are aggregated by median within each task, then by equal-weight average across tasks, so large tasks (*dominick*, 100k+ series) do not dominate small ones (*ercot*, 8 series). Using the resulting ranking, we test compression budgets $k \in \{0, \dots, 6\}$ on these same 6 tasks and keep the k that maximizes the aggregated skill score. Final evaluation adds the 19 held-out tasks never used for calibration.

Evaluation protocol. We follow the original Chronos Benchmark II protocol [Ansari et al., 2024]: a single, predefined test window per series (task-specific horizon H , 4–56 steps), with all preceding history used as context. All evaluation runs in strict univariate mode (`cross_learning=False`), isolating the effect of pruning from in-context learning across series.

Metrics. We report MASE for point-forecast accuracy and WQL for probabilistic quality, using standard GluonTS conventions; MASE seasonality is inferred automatically from each series’ frequency (`get_seasonality`), following Ansari et al. [2024] (Appendix C). Scores are aggregated across tasks using a geometric mean of the ratio between the model’s metric and a Seasonal Naive baseline, with equal weight per task, following the aggregation convention of Ansari et al. [2024]. We report this aggregate as a skill score, $100 \times (1 - \text{gmean}(r))$ (higher is better), the convention used by Ansari et al. [2025]. We also report a win rate: the share of tasks on which the pruned model beats the full model.

CO₂, latency, and FLOPs measurements details in Appendix B.

4.2 Results

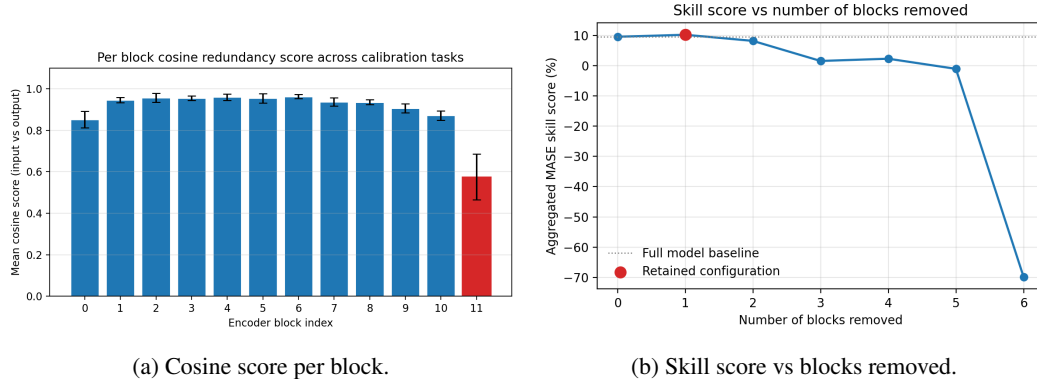


Figure 4: (a) Mean cosine similarity score per Chronos-2 encoder block, averaged across the 6 calibration tasks (error bars: std across tasks). (b) Aggregated MASE skill score vs number of blocks removed on the same tasks; the dotted line marks the full model baseline, and the retained configuration (1 block) is highlighted.

Block redundancy and configuration selection. Figure 4a reports the per-block cosine score obtained during calibration. Block 11 (the last encoder block) has by far the lowest score (0.576) and the highest inter-task standard deviation (0.111). It scores lower than the median of the other 11 blocks on all 6 calibration tasks, a difference confirmed by a one-sided Wilcoxon signed-rank test ($p = 0.016$), indicating that this block is consistently the least redundant one across task domains. The remaining blocks fall in a narrower, higher range (0.85 to 0.96), with lower inter-task variance. This is consistent with the few-layer dominance reported for large time series models by Qiu et al. [2026]: only a small subset of layers is functionally important, while the majority are redundant.

Figure 4b shows the aggregated MASE skill score on the 6 calibration tasks as blocks are progressively removed, following the cosine similarity ordering. Removing a single block (the most redundant one, ranked first by the cosine score) slightly improves the skill score over the full model (10.24% vs. 9.57%). Performance then degrades gradually up to 3 blocks removed, before dropping sharply at 6 blocks removed (-69.9%), suggesting that beyond a certain point the blocks being removed are no longer redundant but functionally important. Based on this exploration, we retain the single-block configuration ($k = 1$) for the remainder of our study, as it is the only tested configuration that does not degrade the aggregated skill score relative to the full model.

4.3 Full-scale evaluation

We confirm this result on the complete 25-task evaluation set, including 19 tasks never seen during calibration or configuration selection. Table 2 summarizes the comparison between the full model and the retained pruned configuration (1 block removed, 7.9% fewer parameters).

The pruned model shows a small drop in point-forecast accuracy (MASE skill score, -1.12 points, about 5% relative) but a small improvement in probabilistic forecast quality (WQL skill score, $+0.29$ points). We do not have a verified explanation for this split between MASE and WQL, and both differences are small. A possible factor, which we did not test, is that MASE only evaluates the median whereas WQL averages the loss over nine quantile levels. Overall, removing the single block selected by our criterion (block 6) changes aggregated performance by roughly one skill-score point at most on both metrics, on tasks that were not used during the pruning procedure. Per-task results are reported in Table 11 (Appendix).

Table 2: Aggregated performance comparison between the full and pruned Chronos-2 models on the 25-task evaluation set. Skill scores are computed relative to a Seasonal Naive baseline (higher is better). Win rate is the share of tasks on which the pruned model beats the full model.

Metric	Full	Pruned	Δ
MASE skill score (%)	22.99	21.87	-1.12
WQL skill score (%)	41.24	41.53	$+0.29$
MASE win rate (%)		–	40.0
WQL win rate (%)		–	52.0

4.4 Compute cost reduction

Table 3 reports parameter count, FLOPs (computed analytically for a single forward pass), measured CO₂ emissions, and measured latency for the full and pruned models. More details in Section B.

Table 3: Compute cost comparison between the full and pruned Chronos-2 models.

Metric	Full	Pruned	Reduction
Parameters	119.48M	110.04M	7.90%
FLOPs	59.52 G	54.67 G	8.15%
CO ₂ emissions (g CO ₂ eq)	1.221 ± 0.037	1.114 ± 0.002	8.72%
Latency (ms)	19.43 ± 1.02	18.27 ± 1.00	5.97%

All four independent measures of computational cost, parameter count, analytically-computed FLOPs, measured CO₂ emissions, and measured latency, show a consistent reduction in the 6 to 9% range, closely matching the 7.9% parameter reduction achieved by removing a single encoder block. This agreement across independent metrics reinforces the reliability of our measurements. Latency shows the smallest relative reduction, which is expected since it includes fixed overheads (data loading, tokenization, CUDA synchronization) that do not scale down with the removed block.

Taken together, these results show that a small, targeted pruning of Chronos-2 (the single block selected by our criterion) reduces inference-time compute cost and CO₂ emissions by roughly 6 to 9%, at essentially no cost to aggregated forecasting performance on a 25-task zero-shot benchmark.

5 Conclusion

This paper examined the sustainability of TSFM deployment from two angles. First, we showed that Chronos-2’s pretraining cost is only amortized under sufficiently large or frequently queried deployments: at $N_a = 1,000$ assets, it breaks even with LightGBM after about 18 months under daily inference, but this threshold shifts by up to $49\times$ with inference frequency alone, and a classical model remains cheaper indefinitely for a small number of assets. Second, removing a single encoder block (7.9% of parameters) selected by a gradient-free pruning criterion yields a consistent 6 to 9% reduction in FLOPs, CO₂ emissions, and latency, at essentially no cost to zero-shot forecasting performance. Together, these results offer a practical framework for sustainable TSFM deployment, and a low-effort compression path when pretraining cost does not amortize. See limitations of our study in Appendix E.

References

- Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, Jasper Zschiegner, Danielle C. Maddix, Hao Wang, Michael W. Mahoney, Kari Torkkola, Andrew Gordon Wilson, Michael Bohlke-Schneider, and Yuyang Wang. Chronos: Learning the language of time series. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL <https://openreview.net/forum?id=gerNCVqqtR>.
- Abdul Fatir Ansari, Oleksandr Shchur, Jaris Küken, Andreas Auer, Boran Han, Pedro Mercado, Syama Sundar Rangapuram, Huibin Shen, Lorenzo Stella, Xiyuan Zhang, Mononito Goswami, Shubham Kapoor, Danielle C. Maddix, Pablo Gueron, Tony Hu, Junming Yin, Nick Erickson, Prateek Mutalik Desai, Hao Wang, Huzefa Rangwala, George Karypis, Yuyang Wang, and Michael Bohlke-Schneider. Chronos-2: From univariate to universal forecasting, 2025. URL <https://arxiv.org/abs/2510.15821>.
- Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 10148–10167. PMLR, 2024. URL <https://proceedings.mlr.press/v235/das24c.html>.
- Lars Graf, Thomas Ortner, Stanisław Woźniak, and Angeliki Pantazi. Flowstate: Sampling-rate-equivariant time-series forecasting. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR, 2026. URL <https://arxiv.org/abs/2508.05287>.
- Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Gloriosi, and Daniel A. Roberts. The unreasonable ineffectiveness of the deeper layers, 2024. URL <https://arxiv.org/abs/2403.17887>.
- Marius Hobbhahn and Jaime Sevilla. What’s the backward-forward flop ratio for neural networks? Epoch AI, 2021. URL <https://epoch.ai/blog/backward-forward-FLOP-ratio>.
- Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL <https://arxiv.org/abs/2001.08361>.
- Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, volume 30, pages 3149–3157, 2017. URL <https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree>.
- Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *International Conference on Learning Representations*, 2023. URL <https://arxiv.org/abs/2211.14730>.
- Xin Qiu, Junlong Tong, Yirong Sun, Yunpu Ma, and Xiaoyu Shen. Revealing scaling paradox in large-scale time series models: Implications for more efficient and accurate forecasting. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL <https://openreview.net/forum?id=YmFkYtD6w0>.
- Victor Schmidt, Kamal Goyal, Aditya Joshi, Boris Feld, L. Conell, N. Laskaris, D. Blank, J. Wilson, S. Friedler, and Sasha Luccioni. Codecarbon: Estimate and track carbon emissions from machine learning computing, 2021. URL <https://github.com/mlco2/codecarbon>.
- Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, 2020. doi: 10.1145/3381831. URL <https://arxiv.org/abs/1907.10597>.
- Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1355. URL <https://arxiv.org/abs/1906.02243>.

- Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Unified training of universal time series forecasting transformers. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, page 53140–53164. PMLR, 2024. URL <https://proceedings.mlr.press/v235/woo24a.html>.
- Lifan Zhao, Yanyan Shen, Zhaoyang Liu, Xue Wang, and Jiaji Deng. Less is more: Unlocking specialization of time series foundation models via structured pruning. In *Advances in Neural Information Processing Systems*, 2025. URL <https://arxiv.org/abs/2505.23195>.
- Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. A survey on model compression for large language models. *Transactions of the Association for Computational Linguistics*, 12:1556–1577, 2024. URL <https://arxiv.org/abs/2308.07633>.

A Appendix 1: Cost Model Details

A.1 Notation

Table 4: Notation used in the cost equations.

Symbol	Unit	Description
t	months	Time elapsed since deployment.
N_a	–	Number of assets.
n_{inf}	inf./month	Number of 24h forecasts requested per month, per asset.
W_i	windows	Size of the training set used at the i -th retraining.
a, b	windows	Bounds of $W_i \sim \mathcal{U}[a, b]$.
α	gCO ₂ /window	Carbon cost of training LightGBM on a single window.
T_{tr}	months	Time between two LightGBM/PatchTST retrainings.
H	–	Number of successive calls per 24h forecast for LightGBM ($H = 24$).
$C_{\text{inf}}^{\text{C}}$	gCO ₂	Carbon cost of a single LightGBM inference call.
α_{ptst}	gCO ₂ /window	Carbon cost of training PatchTST on a single window.
$C_{\text{inf}}^{\text{ptst}}$	gCO ₂	Carbon cost of a single PatchTST inference call.
C_{pr}^{T}	gCO ₂	Carbon cost of pretraining Chronos-2, paid once.
$C_{\text{inf}}^{\text{T}}$	gCO ₂	Carbon cost of a single Chronos-2 inference call.
$C_{\text{ft}}^{\text{LoRA}}$	gCO ₂	Carbon cost of one Chronos-2 LoRA fine-tuning run.
T_{ft}	months	Time between two Chronos-2 fine-tunings (LoRA or full).
β	gCO ₂ /window	Carbon cost of full fine-tuning Chronos-2 on a single window.
p	–	Fraction of each asset’s data included in the full fine-tuning sample.
$W_{\text{Full}}(N_a)$	windows	Resulting size of the full fine-tuning sample: $p \times W_{\text{base}} \times N_a$.

A.2 Full cost equations

PatchTST follows the same logic as LightGBM but without the H factor, since it is natively multi-step:

$$\text{PatchTST}(t, N_a) = \sum_{j=1}^{N_a} \sum_{i=0}^{k_{\text{tr}}} \alpha_{\text{ptst}} W_i^{(j)} + \lfloor t \rfloor N_a n_{\text{inf}} C_{\text{inf}}^{\text{ptst}} \quad (3)$$

Chronos-2 with LoRA fine-tuning adds a periodic fine-tuning term to the zero-shot cost:

$$\text{TSFM}_{\text{LoRA}}(t, N_a) = C_{\text{pr}} + (k_{\text{ft}} + 1) C_{\text{ft}}^{\text{LoRA}} + \lfloor t \rfloor N_a n_{\text{inf}} C_{\text{inf}}^{\text{T}} \quad (4)$$

with $k_{\text{ft}} = \lfloor t/T_{\text{ft}} \rfloor$. We model a single LoRA adapter trained on a representative sample of a few assets and shared across the assets, so its cost does not depend on N_a . A first fine-tuning is performed at $t = 0$, then every T_{ft} months. We suppose it is a realistic way to fine-tune with LoRA.

Chronos-2 with full fine-tuning follows the same structure, but the fine-tuning cost now depends on N_a :

$$\text{TSFM}_{\text{Full}}(t, N_a) = C_{\text{pr}} + \sum_{j=0}^{k_{\text{ft}}} \beta W_{\text{Full}}(N_a) + \lfloor t \rfloor N_a n_{\text{inf}} C_{\text{inf}}^{\text{T}} \quad (5)$$

with $W_{\text{Full}}(N_a) = p \times W_{\text{base}} \times N_a$, and $W_{\text{base}} = (a + b)/2$: full fine-tuning is run on a stratified sample of a fixed fraction p of each asset’s data, so the fine-tuning sample size grows linearly with the number of assets. We suppose it is a realistic way to full fine-tune.

B Appendix 2: Measurement Protocol Details (cost model)

Hardware and carbon intensity. All measurements use an NVIDIA A100 80GB PCIe GPU (Azure Standard_NC24ads_A100_v4 instance, francecentral region), paired with an AMD EPYC 7V13 CPU. CodeCarbon (v3.2.9) reads GPU power directly through `nvidia-ml-py`, while CPU power is estimated from a TDP-based model, since the RAPL hardware counters it otherwise reads directly were not accessible on our virtual machine. This estimation method is applied

identically across all measurements. The carbon intensity CodeCarbon applied throughout, computed directly from the emissions and energy figures logged for each run, is ≈ 56 gCO₂/kWh for Île-de-France, France, consistent with historical French grid intensity figures from Our World in Data (e.g., 56.54 gCO₂/kWh in July 2020).

CodeCarbon applies the electricity carbon intensity of the detected country, taken from Our World in Data, or of the cloud provider region when available. **Every carbon value is an energy times the local grid carbon intensity applied by CodeCarbon, so it is proportional to it (multiply by $I/56$ for a grid of I gCO₂/kWh). If pretraining and deployment run on the same grid, all costs scale together and the break-even durations are unchanged.**

Estimating C_{pr} . Chronos-2 is never pretrained from scratch in this work. We estimate C_{pr} , the fixed cost a TSFM deployment must amortize and the parameter driving every break-even result in Section 4.2, by measuring a single pretraining step and extrapolating from it: one forward and backward pass on a heterogeneous batch combining univariate series, multivariate groups, and series with covariates, with a context length of 2048, following the first stage of Chronos-2’s own pretraining procedure, with tf32 precision, a fused AdamW optimizer, and `torch.compile`. Both these settings and the 200,000-step total used for extrapolation are taken from the publicly available configuration of Chronos T5-base, the closest model for which this information is documented, since Chronos-2’s own pretraining setup is not disclosed. This single-step cost is reported in Table 6; multiplied by 200,000 steps, it gives $C_{\text{pr}} = 65.17$ gCO₂, which should be read as a lower bound: it assumes a single GPU, whereas real pretraining runs typically use many GPUs in parallel.

Sensitivity to C_{pr} . C_{pr} is an extrapolated lower bound: as said, it is estimated on a single GPU, with a step count borrowed from Chronos T5-base. We test how the break-even usage duration T^* between Chronos-2 ZeroShot and LightGBM moves when C_{pr} is multiplied by a factor k from $1\times$ to $100\times$, at $N_a = 1000$ under the baseline scenario (Figure 5). T^* is proportional to C_{pr} : $T^* \approx 18k$ months. C_{pr} is a fixed cost repaid by a constant monthly saving of Chronos-2 over LightGBM, so scaling it scales the time needed to repay it. A $2\times$ higher cost doubles the break-even to 36 months, $5\times$ gives about 90 months, and $10\times$ gives 180 months (15 years). Beyond $60\times$ (≈ 90 years), Chronos-2 does not break even within our 100-year search horizon. A higher pretraining cost lengthens the break-even rather than shortening it. Since our estimate is a lower bound, the 18-month figure at $N_a = 1000$ should be read as optimistic for Chronos-2, which strengthens the conclusion that a TSFM is not amortized in every deployment. This analysis varies C_{pr} only, all other parameters staying at their baseline values.

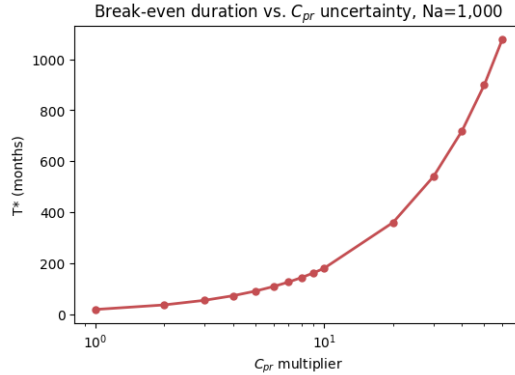


Figure 5: Break-even usage duration T^* between Chronos-2 ZeroShot and LightGBM versus a multiplier on C_{pr} , at $N_a = 1000$ (baseline scenario). Beyond $60\times$, no crossover is found within our 100-year search horizon.

Estimating training FLOPs from a forward pass. For PatchTST and Chronos-2, we measure FLOPs with `fvcore`, which traces a model’s forward computation graph but not its backward pass, generated dynamically by autograd. We estimate a full training step as $3\times$ the forward pass alone, following the scaling law analysis of Kaplan et al. [2020], confirmed empirically across a range of

architectures by Hobbhahn and Sevilla [2021]. LightGBM’s FLOPs are computed analytically from its histogram-based algorithm instead, since `fvcore` does not apply to tree-based models.

Dataset and number of runs for Part I. All measurements of Part I are carried out on ETTh1, an hourly electricity transformer temperature dataset with 7 variables, split 70/15/15 into train, validation, and test sets, yielding 12,026 training windows for LightGBM and PatchTST. **Every carbon cost figure in the tables below is averaged over $N = 10$ independent runs for training, fine-tuning, and pretraining-step measurements, and over $N = 20$ independent runs for inference measurements, each inference run itself averaging 1,000 individual calls.**

Fixed training budget. Training LightGBM and PatchTST for carbon measurement requires choosing a training length, since neither has a natural stopping point; we fix this arbitrarily to 300 estimators and 20 epochs respectively, applied identically in every run.

Model hyperparameters. Table 5 reports the hyperparameters used for LightGBM, PatchTST, and Chronos-2 fine-tuning/pretraining measurements for the carbon cost model.

Table 5: Hyperparameters used for the models in Section 3.1.

Model	Parameter	Value
LightGBM	Estimators	300
	Num. leaves	31
	Max bin	255
	Learning rate	0.05
	Feature/bagging fraction	0.8
	Bagging frequency	5
PatchTST	Patch length / stride	16 / 8
	d_{model}	128
	Num. heads	4
	Num. layers	3
	Dropout	0.2
	Epochs / batch size	20 / 32
Chronos-2 (pretraining step)	Learning rate	10^{-3}
	Context length	2048
	Batch size	32
	Learning rate	10^{-3}
Chronos-2 (LoRA fine-tuning)	Steps per run (extrapolated to)	100 (200,000)
	Steps	2,000
	Batch size	32
Chronos-2 (full fine-tuning)	Learning rate	10^{-4}
	Epochs	3
	Batch size	32
Chronos-2 (full fine-tuning)	Learning rate	10^{-5}

Fine-tuning parameters. $C_{\text{ft}}^{\text{LoRA}}$ is measured directly at 2,000 steps, matching the value used in Section 3.1. β is reported per training window rather than as a fixed cost, since full fine-tuning’s total cost scales with N_a through $W_{\text{Full}}(N_a)$.

Two tables report the results of this protocol, and answer different questions. Table 6 gives the cost of fixed, non-scenario operations, one full training run and one inference call, on both CPU and GPU where measured, except for PatchTST training (GPU only, its realistic training device) and Chronos-2 training (GPU only, since it is never trained from scratch). For Chronos-2, the training row is the single pretraining step described above, not a full run. Table 7 instead gives the per-unit parameters plugged into the cost equations of Section 3.1, from a separate measurement campaign than Table 6: for LightGBM, on CPU (LightGBM is assumed to run on CPU in production); for PatchTST and Chronos-2, on GPU, their realistic deployment device.

Table 6: Raw measures: carbon cost, FLOPs, and latency for one full training run and one inference call, per approach (mean $\pm$ std). Chronos-2’s training row is a single pretraining step, not a full run.

Model	Operation	Carbon (gCO ₂)	FLOPs	Latency (ms)
LightGBM	Training, CPU	0.00722 ± 0.00070	5.32×10^{10}	$2,903.6 \pm 163.7$
	Training, GPU	0.01769 ± 0.00078		$5,698.1 \pm 222.5$
	Inference, CPU	$(6.81 \pm 0.39) \times 10^{-6}$	2,972.5	2.17 ± 0.57
	Inference, GPU	$(6.99 \pm 1.42) \times 10^{-6}$		2.33 ± 1.04
PatchTST	Training, GPU	1.0800 ± 0.0087	5.48×10^{14}	$185,478.9 \pm 784.9$
	Inference, CPU	$(3.26 \pm 0.15) \times 10^{-5}$	7.61×10^8	8.70 ± 13.87
	Inference, GPU	$(6.60 \pm 0.42) \times 10^{-6}$		2.70 ± 0.12
Chronos-2	Pretraining, GPU (1 step)	$(3.26 \pm 0.02) \times 10^{-4}$	1.45×10^{12}	53.80 ± 0.41
	Inference, CPU	$(3.70 \pm 4.13) \times 10^{-4}$	1.51×10^9	73.37 ± 53.92
	Inference, GPU	$(6.08 \pm 0.05) \times 10^{-5}$		23.64 ± 0.96

Table 7: Cost model parameters (Section 3.1).

Parameter	Value
α (gCO ₂ /window)	5.51×10^{-7}
$C_{\text{inf}}^{\text{C}}$ (gCO ₂)	6.44×10^{-6}
α_{ptst} (gCO ₂ /window)	9.01×10^{-5}
$C_{\text{inf}}^{\text{ptst}}$ (gCO ₂)	7.43×10^{-6}
C_{pr}^{T} (gCO ₂)	65.17
$C_{\text{inf}}^{\text{T}}$ (gCO ₂)	6.44×10^{-5}
$C_{\text{ft}}^{\text{LoRA}}$ (gCO ₂)	0.62
β (gCO ₂ /window)	2.21×10^{-5}

Cost measurements for the pruning experiments. Full and pruned models are measured under identical conditions, on the hardware and with the CodeCarbon configuration described above. CO₂ emissions are accumulated over zero-shot inference on a fixed subset of the calibration tasks (*ercot*, *exchange_rate*, *monash_hospital*, *monash_m3_monthly*, *m4_yearly*, and a fixed random sample of 200 series of *dominick*; 25,411 windows in total), identical for both models, rather than on the full 25-task set, to keep repeated measurements tractable. We report the mean and standard deviation over 5 repetitions of this whole pass. Latency is measured on 10 series drawn at random (fixed seed) from the same subset, with 3 untimed warmup passes and 20 timed runs per series; we report the median and standard deviation over the 200 timed calls. FLOPs are counted with `fvcore` for a single forward pass of the model on a random input of 8192 points (the model’s context length) with batch size 1. They depend on the input shape and not on its values, so one deterministic count per model suffices. Both models run on the same grid, so the relative CO₂ reduction does not depend on the carbon intensity applied.

C Appendix 3: Cost Model Results

C.1 Figures

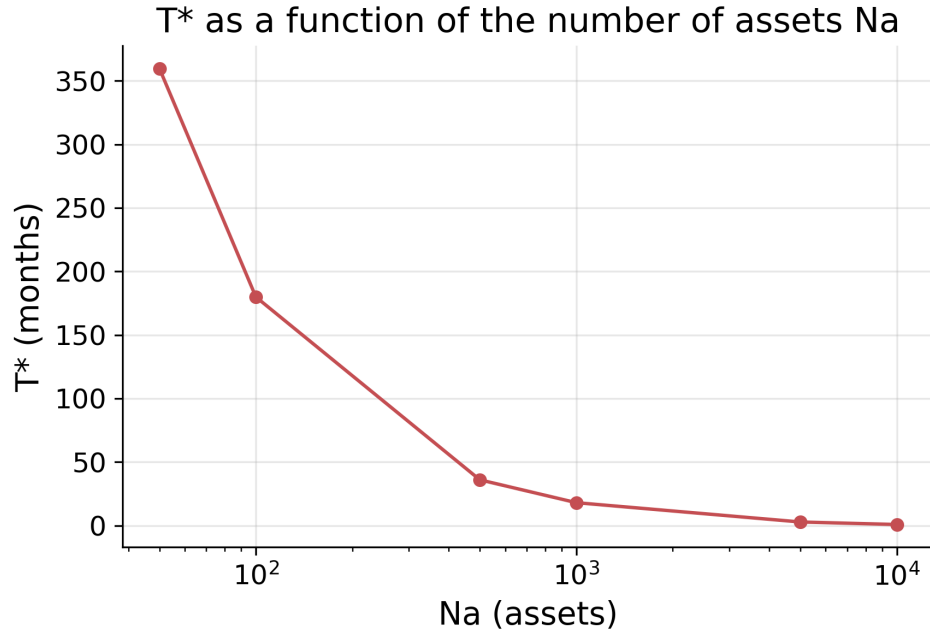


Figure 6: Break-even usage duration T^* (months) as a function of the number of assets N_a , at which Chronos-2 ZeroShot becomes cheaper than LightGBM under the baseline scenario.

D Appendix 4: Model Pruning

Clarification and additional ablation of the pruning criterion To clarify, the block removed in our main experiments (§4.1) is block 6, the block with the *highest* cosine score (0.963) among the 12 encoder blocks, identified by our criterion as the most redundant. Block 11, which has the *lowest* cosine score (0.576), is the block the criterion identifies as the *most* important to keep, and is preserved rather than removed in all configurations we evaluate. Table 9 reports the score for every block.

To verify that our criterion correctly identifies block 11 as critical rather than redundant, we removed it on its own and evaluated the resulting model on the 6 calibration tasks, checking whether its low cosine score does correspond to a substantial performance drop. As an additional check, we removed one arbitrarily chosen non-extreme block not selected by our criterion (block 5), which our criterion also happens to identify as redundant and ranks close to, though slightly lower than, block 6 (cosine score 0.954 versus 0.963), and evaluated the resulting model on the same 6 calibration tasks.

Table 8 reports the aggregated skill scores obtained. Removing block 11 causes a severe degradation (MASE skill score dropping from 9.57% to −47.30%, WQL from 26.89% to −63.09%), confirming that its low cosine score correctly identifies it as critical to keep. By contrast, removing block 5 or block 6, both identified as redundant by the criterion, has a comparably modest effect on performance, consistent with their similarly high cosine scores.

Table 8: Aggregated MASE and WQL skill scores on the 6 calibration tasks, removing a single block at a time.

Removed block	Cosine score	MASE skill (%)	WQL skill (%)
None (full model)	–	9.57	26.89
Block 6 (retained)	0.963	10.24	28.65
Block 5	0.954	9.16	28.40
Block 11	0.576	−47.30	−63.09

Table 9: Mean cosine score per encoder block, averaged across the 6 calibration tasks (median within task, equal weight across tasks). Lower scores indicate less redundant blocks.

Block	Cosine score
6	0.963
4	0.960
2	0.957
3	0.955
5	0.954
1	0.946
7	0.936
8	0.936
9	0.906
10	0.871
0	0.852
11	0.576

Table 10 reports the full task-level statistics for the 27 tasks of Chronos Benchmark II used in our evaluation (25 tasks after excluding the two ETT variants, see Section 4.1). Table 11 reports the per-task MASE and WQL scores for the full and pruned models, complementing the aggregated results of Section 3.2.

Table 10: Task-level statistics for the 27 tasks of Chronos Benchmark II, reproduced from Ansari et al. [2024] (Table 3). *ETT (15 Min.)* and *ETT (Hourly)* are excluded from our evaluation due to a data loading incompatibility; our study covers the remaining 25 tasks.

Dataset	Domain	Freq.	# Series	Series length			H
				min	avg	max	
Australian Electricity	energy	30min	5	230736	231052	232272	48
CIF 2016	banking	1M	72	28	98	120	12
Car Parts	retail	1M	2674	51	51	51	12
Covid Deaths	healthcare	1D	266	212	212	212	30
Dominick	retail	1D	100014	201	296	399	8
ERCOT Load	energy	1H	8	154854	154854	154854	24
ETT (15 Min.)	energy	15min	14	69680	69680	69680	24
ETT (Hourly)	energy	1H	14	17420	17420	17420	24
Exchange Rate	finance	1B	8	7588	7588	7588	30
FRED-MD	economics	1M	107	728	728	728	12
Hospital	healthcare	1M	767	84	84	84	12
M1 (Monthly)	various	1M	617	48	90	150	18
M1 (Quarterly)	various	3M	203	18	48	114	8
M1 (Yearly)	various	1Y	181	15	24	58	6
M3 (Monthly)	various	1M	1428	66	117	144	18
M3 (Quarterly)	various	3M	756	24	48	72	8
M3 (Yearly)	various	1Y	645	20	28	47	6
M4 (Quarterly)	various	3M	24000	24	100	874	8
M4 (Yearly)	various	1Y	23000	19	37	841	6
M5	retail	1D	30490	124	1562	1969	28
NN5 (Daily)	finance	1D	111	791	791	791	56
NN5 (Weekly)	finance	1W	111	113	113	113	8
Tourism (Monthly)	various	1M	366	91	298	333	24
Tourism (Quarterly)	various	1Q	427	30	99	130	8
Tourism (Yearly)	various	1Y	518	11	24	47	4
Traffic	transport	1H	862	17544	17544	17544	24
Weather	nature	1D	3010	1332	14296	65981	30

Table 11: Per-task comparison between the full and pruned Chronos-2 models on the 25 tasks of Chronos Benchmark II. Lower MASE and WQL indicate better performance. Bold values mark the better-performing configuration on each task, based on full-precision values even where rounded figures appear equal.

Task	MASE		WQL	
	Full	Pruned	Full	Pruned
dominick	0.809	0.796	0.324	0.321
ercot	0.832	0.822	0.026	0.026
exchange_rate	1.852	1.807	0.012	0.011
m4_quarterly	1.201	1.186	0.076	0.075
m4_yearly	3.378	3.427	0.117	0.118
m5	0.905	0.907	0.553	0.553
monash_australian_electricity	0.629	0.744	0.031	0.036
monash_car_parts	0.834	0.834	0.961	0.958
monash_cif_2016	0.933	0.898	0.011	0.010
monash_covid_deaths	35.406	32.476	0.039	0.029
monash_fred_md	0.469	0.592	0.020	0.024
monash_hospital	0.773	0.766	0.054	0.052
monash_m1_monthly	1.027	1.037	0.147	0.146
monash_m1_quarterly	1.661	1.700	0.083	0.085
monash_m1_yearly	3.830	3.825	0.138	0.133
monash_m3_monthly	0.813	0.815	0.089	0.089
monash_m3_quarterly	1.162	1.213	0.070	0.073
monash_m3_yearly	2.996	3.084	0.147	0.150
monash_nn5_weekly	0.882	0.891	0.082	0.082
monash_tourism_monthly	1.423	1.436	0.073	0.077
monash_tourism_quarterly	1.566	1.586	0.065	0.063
monash_tourism_yearly	3.666	3.780	0.152	0.154
monash_traffic	0.818	0.819	0.242	0.237
monash_weather	0.774	0.777	0.125	0.126
nn5	0.577	0.559	0.149	0.144

E Appendix 5: Limitations

E.1 Carbon cost modeling

Scope of the measurements. All carbon measurements are specific to a single hardware setup (NVIDIA A100 80GB, AMD EPYC 7V13), a single dataset (ETTh1), and a single grid carbon intensity (≈ 56 gCO₂/kWh, Île-de-France). Absolute values would differ on other hardware, datasets, or regions, and our results should be read as illustrative of a scenario rather than universal constants.

CPU power estimation. CPU power is estimated from a TDP-based model rather than measured directly, since the RAPL hardware counters CodeCarbon otherwise reads were not accessible on our virtual machine. This affects every measurement using this fallback, but disproportionately LightGBM, which we assume runs on CPU in production.

Estimating C_{pr} . C_{pr} , the parameter driving every break-even result in this paper, is extrapolated from a single measured pretraining step using the publicly documented step count of Chronos T5-base, since Chronos-2’s own pretraining schedule and infrastructure are not disclosed. It should be read as a lower bound: it assumes a single GPU rather than the distributed setup a real pretraining run would likely use.

Comparison at unequal footing. Each model is costed under the usage pattern it was built for (Section 3.1): LightGBM and PatchTST are trained per asset, while a single LoRA adapter is shared across assets for Chronos-2. Sharing a single PatchTST across similar assets would remove its dependence on N_a in training cost and would likely make it much cheaper than reported here. The comparison is thus not at equal informational access, and PatchTST’s cost should be read as that of per-asset training. A matched comparison requires a separate measurement campaign.

Hyperparameter tuning cost. The training budgets of LightGBM (300 estimators) and PatchTST (20 epochs) are fixed, untuned choices, and no hyperparameter search was run, so no tuning cost enters the cumulative cost of these baselines. In practice, a search over K configurations would add roughly K training runs per tuned model. Chronos-2 zero-shot requires none, so omitting it favors the baselines: including it would move the break-even earlier. Also, since LightGBM’s cost grows roughly linearly with the number of estimators, a smaller model would shift the break-even later, and a larger one earlier. We did not quantify this effect.

Consistency across measurement campaigns. The parameters used in the cost equations (Table 7) come from a separate, earlier measurement campaign than the raw measures reported in Table 6.

Measurement variance. One measurement, Chronos-2’s CPU inference cost, shows unusually high variance across runs (Table 6) that we did not fully investigate. We report it as measured rather than excluding it, but it should be treated with more caution than the rest of the table.

E.2 Pruning

Benchmark coverage. Two tasks of Chronos Benchmark II (*ETT 15-min* and *ETT hourly*) are excluded from our evaluation due to a technical incompatibility between their loading mechanism and the current version of the HuggingFace datasets library; our results cover 25 of the 27 original tasks.

Pruning granularity. Pruning operates at the level of entire encoder blocks. Removing a block removes its time attention, group attention, and feed-forward sub-layers simultaneously, so we cannot isolate which specific mechanism drives the observed redundancy or performance impact.

Block selection at fixed compression. At each compression level k , we only evaluate the specific set of k blocks chosen by the cosine ordering. We do not compare this selection against alternative sets of k blocks chosen by other criteria, **so we cannot claim that the cosine score identifies the best possible blocks to remove at a given compression budget, only that the resulting configuration performs well in absolute terms.**

Univariate evaluation only. All experiments are run in strict univariate mode (`cross_learning=False`). We do not evaluate whether pruning affects Chronos-2’s in-context learning capabilities across series, which depend on batch composition and are only engaged in cross-learning mode.

Fixed compute-cost measurement subset. CO₂ and latency are measured on a fixed subset of tasks (25,411 windows total) rather than the full 25-task evaluation set, to keep repeated measurements tractable. While inference cost primarily depends on input size rather than task content, this subset is not identical to the one used for the performance evaluation in Table 2.

Single hardware and single run. All compute cost measurements are collected on a single machine (see Appendix B) and, for the main performance comparison (Table 2), from a single deterministic zero-shot evaluation run per configuration.

Scope of the pruning criterion. The cosine similarity score is a lightweight, gradient-free heuristic. We do not compare it against alternative pruning criteria (e.g., gradient-based importance scores), so we cannot claim it is the most effective method available, only that it is computationally cheap and yields a configuration with no measurable aggregated performance loss on this benchmark.

NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope?

Answer: [\[Yes\]](#)

Justification: Yes

Guidelines:

- The answer [\[N/A\]](#) means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [\[No\]](#) or [\[N/A\]](#) answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [\[Yes\]](#)

Justification: Yes, see Appendix E

Guidelines:

- The answer [\[N/A\]](#) means that the paper has no limitation while the answer [\[No\]](#) means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate “Limitations” section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren’t acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [\[N/A\]](#).

Justification: The paper doesn't include any theoretical result.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental result reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The Appendix B explains all the details to reproduce the carbon cost modelization, and Section 4.1 details the setup to reproduce the pruning experiences.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
 - (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
 - (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: While the code is not available, we describe all the steps to reproduce it. Data is accessible.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://neurips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://neurips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental setting/details

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: Yes (see answer 4)

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment statistical significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: In the first part : partially. Raw carbon cost, FLOPs, and latency measurements are reported with standard deviations over multiple independent runs (Table 6, Appendix B). The cost model figures (Figures 1, 2, 3) show simulated cumulative cost curves without uncertainty bands, for visual clarity. In the second part, our goal is to assess whether pruning can reduce Chronos-2's compute cost without substantially degrading forecasting performance, rather than to establish statistical superiority of one configuration over another. We report point estimates (aggregated skill scores, win rate) for this comparison across the 25 evaluation tasks, without a formal significance test on this specific result. We do report standard deviations for CO2 emissions and latency, and a Wilcoxon signed-rank test for the block-11 redundancy analysis.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments compute resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: Yes, see Appendix B

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines>?

Answer: [Yes]

Justification: Yes

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: This paper directly addresses the environmental cost of machine learning deployment. It provides a practical framework for reducing CO₂ emissions associated with TSFM deployment, and demonstrates that structured pruning can reduce inference-time emissions by 6 to 9% at no cost to forecasting performance.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: the paper poses no such risks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Yes

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. New assets

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: the paper does not release new assets.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. Crowdsourcing and research with human subjects

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: the paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. Institutional review board (IRB) approvals or equivalent for research with human subjects

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: the paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: only for writing, editing, or formatting purposes

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.