Serverless Cold-Start Prediction: A Time Series AI Pipeline

Joy Kim (2022195142)

1 Introduction

Serverless computing platforms (AWS Lambda, Azure Functions, Google Cloud Functions) execute code on demand without requiring users to manage infrastructure. However, these platforms suffer from **cold starts** which is a common problem of these serverless functions. When a function has not been invoked recently, the platform must initialize a new container, causing latency spikes of 100ms to several seconds.

This project addresses cold start mitigation through **proactive traffic prediction**. By predicting upcoming request volumes, platforms can pre-warm function instances before traffic spikes occur, reducing user-facing latency.

Prior Work: The cold start problem has been studied extensively, for example, by Shahrad et al. [1], who analyzed Azure Functions traces. They demonstrated that while standard fixed keep-alive policies are inefficient, dynamic policies based on historical histograms can significantly reduce latency. While more recent research has explored ML-based prediction, these approaches often rely on complex, custom-built models that are difficult to deploy at scale. In contrast, this project investigates whether off the shelf pre-trained time series foundation models (like Chronos) can achieve competitive accuracy with minimal engineering effort and zero task-specific tuning.

2 Task Definition

- **Task description:** Predict the number of function invocations for the next minute given historical invocation data. This enables proactive warming of serverless function instances.
- **Motivation:** Cold starts cause significant latency degradation in serverless applications. Accurate traffic prediction enables platforms to pre-warm instances, reducing P99 latency by avoiding cold starts during traffic spikes.
- **Input / Output:**
  - Input: Historical time series of per-minute invocation counts (up to 512 minutes of history)
  - Output: Predicted invocation count for the next minute
- **Success criteria:**
  - Lower Mean Absolute Error (MAE) than baseline
  - Fewer under-predictions (cold starts) than baseline

3 Methods

3.1 Naive Baseline: Reactive Lag-1

Method Description

The baseline uses a simple **reactive** approach: predict the next minute's traffic as equal to the current minute's traffic (lag-1 prediction). Mathematically:

$$\hat{y}_{t+1} = y_t$$

This is equivalent to assuming traffic remains constant minute-to-minute.

Why Naive

This approach is naive because it:

- Cannot anticipate changes (it only reacts after they happen)
- Is always one step behind during traffic ramps
- Ignores all historical patterns (daily cycles, burst patterns, etc.)

Likely Failure Modes

- **Traffic spikes**: Scheduled jobs or viral events cause sudden increases that lag-1 misses entirely
- **Accelerating ramps**: Gradual increases that accelerate are persistently under-predicted
- **Periodic patterns**: Daily/hourly patterns are ignored; the baseline treats midnight and peak hours identically

3.2 AI Pipeline: Chronos Pre-trained Time Series Model

Model Selection

I use **Amazon Chronos** [2], a family of pre-trained time series foundation models available on HuggingFace. Specifically, I use `amazon/chronos-t5-small`, which provides a good balance between accuracy and inference speed.

Why Chronos:

- Pre-trained on diverse time series data (zero-shot forecasting)
- Handles various patterns: trends, seasonality, level shifts
- Lightweight variant available for real-time inference
- Easy integration via HuggingFace Transformers

Pipeline Stages

1. **Preprocessing**: Extract historical traffic as a 1D tensor. Use sliding window of up to 512 minutes (Chronos context length).
2. **Model Loading**: Load pre-trained Chronos-T5-Small from HuggingFace.
3. **Inference**: Generate probabilistic forecast with 20 samples. Chronos outputs a distribution over possible future values.
4. **Post-processing**: Extract median prediction as point estimate. Clip negative values to zero (traffic cannot be negative).

Design Choices

- **Context length = 512**: Captures approximately 8.5 hours of history, sufficient for daily patterns
- **Median aggregation**: More robust than mean for skewed distributions
- **No fine-tuning**: Zero-shot inference demonstrates the power of pre-trained models

4 Experiments

4.1 Dataset

- **Source:** Azure Functions Dataset 2019 [1], containing real production traces from Microsoft Azure's serverless platform.
- **Total examples:** The dataset contains invocation counts for thousands of functions over 14 days. I focus on a single high-traffic function with interesting temporal patterns.
- **Time series length:** 1,440 data points (one per minute for 24 hours)
- **Train/Test split:** 80/20 temporal split
  - Train: 1,152 samples (minutes 0–1151, first 19.2 hours)
  - Test: 289 samples (minutes 1152–1440, last 4.8 hours)
- **Preprocessing steps:**
  - Selected function with high traffic variance (interesting prediction challenge)
  - Converted invocation counts to float32 tensors
  - No normalization (Chronos handles raw values)

4.2 Metrics

I use two primary metrics aligned with my goal of reducing cold starts:

- **MAE (Mean Absolute Error)**: Measures average prediction accuracy

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **Cold Start Count**: Number of under-predictions where $\hat{y}_i < y_i$. Each under-prediction could cause a cold start in production.

Why these metrics? MAE captures overall prediction quality while cold start count directly measures the failure mode I am trying to prevent. Over-predictions waste resources but do not really affect user latency.

4.3 Results

Table 1: Comparison of Baseline and AI Pipeline on test set (289 samples)

Method	MAE	Cold Starts
Baseline (Reactive Lag-1)	4728.18	146 (50.5%)
AI Pipeline (Chronos)	4403.89	151 (52.2%)

Key Finding: The AI pipeline achieves a **6.9% reduction in MAE** compared to the baseline (4728 → 4404). The AI model has slightly more cold starts (151 vs 146), indicating a trade-off: the model optimizes for overall accuracy rather than specifically avoiding under-predictions.

Figure 1: Predictions vs actual traffic over the 289-minute test period. The AI model tracks the actual traffic more closely than the baseline during recovery periods.

Qualitative Examples

Example 1: AI Outperforms Baseline (Index 247 - Traffic Recovery)

After a sudden traffic dip (68K → 90K → 124K invocations), the baseline predicted 90K (27.4% error) while the AI predicted 117K (5.5% error). The AI anticipated the recovery pattern while the baseline simply repeated the low value.

Example 2: Baseline Outperforms AI (Index 246 - Rapid Decline)

During rapid traffic decline (137K → 68K), the baseline's lag-1 approach (24.4% error) outperformed the AI (43.7% error). The AI over-predicted because it expected traffic to remain high based on recent history.

Example 3: Peak Traffic (Index 161)

At peak load (150K invocations), both methods performed similarly well: baseline 1.7% error, AI 2.6% error. During stable high-traffic periods, the simple lag-1 heuristic is nearly as effective as the AI model.

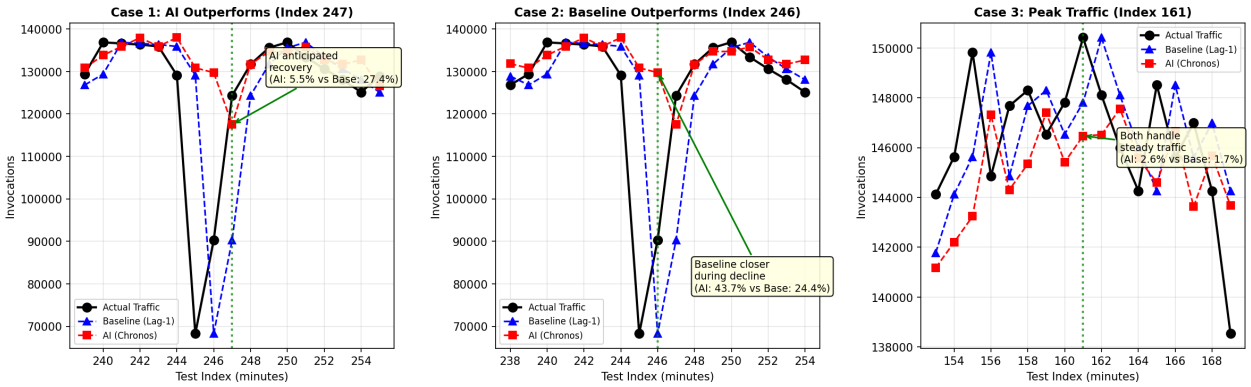

Figure 2: Case study visualizations for the three qualitative examples. Left: AI outperforms baseline during traffic recovery. Center: Baseline outperforms AI during rapid decline. Right: Both methods handle peak traffic similarly.

5 Reflection and Limitations

What worked better than expected: The pre-trained Chronos model achieved a 6.9% MAE improvement over the baseline with zero fine-tuning. The zero shot capability of foundation models is surprising since the model had never seen serverless traffic data during training, but it was able to capture temporal patterns effectively. The setup was also straightforward thanks to the HuggingFace integration. The model ran efficiently on Apple Silicon without requiring a dedicated GPU. AI wins 54% of individual predictions, demonstrating a consistent improvement over baseline.

Failure/Difficulty: The most significant limitation I encountered was the model's performance on cold starts. Despite improving the overall Mean Absolute Error, the AI model resulted in slightly more cold starts (151) compared to the baseline (146). Since minimizing these events was a key success criteria, this outcome was unexpected, and considered a failure in the context of goal of the project. It appears that while the model tracks general traffic trends effectively it tends to smooth over rapid changes that the baseline captures more immediately.

Metric suitability: I found that MAE may not be the perfect metric for this specific problem. It treats over predictions and under predictions as equally bad errors. In a real serverless context, an under prediction is much more costly because it causes latency. While counting cold starts is helpful, it has a limitation as whether the prediction falls short by a single invocation or by a hundred, the metric counts it as just one failure. This ignores the severity of the under prediction. A more suitable approach would be to use a weighted loss function that penalizes under predictions more heavily which would be more in line with the goal of reducing latency.

Future improvements: In the future I could first focus on fine tuning the Chronos model on serverless specific datasets to help it learn the volatile nature of this traffic. I would also explore larger model variants such as Chronos-large to see if increased capacity yields better accuracy. Additionally, a hybrid approach might have a more optimal solution. I could rely on the computationally cheaper baseline for stable traffic periods and only switch to the AI model when traffic spikes which might get the best of both worlds.

References

[1] Shahrad, M., Fonseca, R., Goiri, I., Chaudhry, G., Batum, P., Cober, J., & Bianchini, R. (2020). Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. *USENIX ATC*.

[2] Ansari, A. F., Stella, L., Turkmen, C., Zhang, X., Mercado, P., Shen, H., ... & Wang, Y. (2024). Chronos: Learning the Language of Time Series. *Transactions on Machine Learning Research*.