Predictive Auto-Scaling in Cloud Environments: Evaluating Generalisability, Predictive Accuracy, and Computational Efficiency of Deep Learning Models (LSTM vs. Transformers) using Heterogeneous Cluster Traces

Manuchimso Excel Oliver

R2307D16807305

Mental Wealth; Professional Life (Dissertation)

UEL-CN-7000-107104

August 2026

Acknowledgements

I would like to thank Mahmoud Elbattah, my supervisor, for guidance and critical feedback throughout the design and writing of this dissertation. I am grateful to the University of East London for the resources and support that made this research possible. Thanks also to my family and friends for their patience and encouragement over the course of this degree.

Table Of Contents

Acknowledgements 1
Table Of Contents 2
List of Abbreviations 5
Abstract 6
Chapter 1 — Introduction 7
1.1 Problem Context 7
1.2 Research Problem 8
1.3 Aim and Objectives 8
1.4 Research Questions 9
1.5 Contributions 9
1.6 Dissertation Structure 10
Chapter 2 — Literature Review 12
2.1 Cloud Auto-Scaling and the Case for Prediction 12
2.2 Time-Series Forecasting for Cloud Workloads 13
2.3 LSTM for Sequence Modelling 14
2.4 Transformer for Time-Series 16
2.5 The "Are Transformers Effective?" Critique 17
2.6 Cross-Provider Generalisation: The Gap 19
2.7 Cluster Traces as Benchmarks 20
Chapter 3 — Methodology 22
3.1 Research Paradigm and Design 22
3.2 Ethical Considerations and Data Licensing 22
3.3 Datasets and Rationale 23
3.4 Alibaba Preprocessing 23
3.5 Google Preprocessing 25
3.6 Normalisation 26
3.7 Windowing and Chronological Splits 26
3.8 Model Architectures 27
3.9 Training Regime 29
3.10 Evaluation Methodology 30
3.11 Cross-Provider Setup 31
3.12 Environmental Setup 31
Chapter 4 — Findings 33
4.1 Overview of Experimental Results 33

4.2 Persistence Baselines 33
4.3 In-Provider Prediction Accuracy (RQ1) 35
4.4 Cross-Provider Generalisation (RQ2) — Headline 37
4.5 Computational Cost (RQ3) 41
4.6 Training Dynamics and Loss Curves 43
4.7 Statistical Treatment: Variance Across Seeds 45
Chapter 5 — Discussion 46
5.1 RQ1: On-Provider Accuracy 46
5.2 RQ2: Asymmetric Cross-Provider Transfer 47
5.3 RQ3: Computational Cost 49
5.4 RQ4: Synthesis — No Clear Winner 49
5.5 Implications for Auto-Scaling Practice 51
5.6 Limitations 52
Chapter 6 — Conclusion and future work 52
6.1 Summary of Findings 52
6.2 Future Work 53
6.3 Closing Remarks 55
References 56

List of Tables and Figures

List of Tables

Table 1. Persistence baselines by target provider and forecast horizon.

Table 2. In-provider trained-model results (Alibaba → Alibaba, Google → Google), mean ± sample standard deviation across three seeds.

Table 3. Cross-provider zero-shot trained-model results (Alibaba → Google, Google → Alibaba), mean ± sample standard deviation across three seeds.

Table 4. The four (source, target, model, horizon) cells that beat the persistence baseline in normalised RMSE.

List of Figures

Figure 1. Normalised per-unit RMSE by source-target condition and model, one panel per forecast horizon (5, 15, 30 minutes ahead).

Figure 2. RMSE against forecast horizon for each of the four (source, target) combinations, with the persistence baseline for the corresponding target shown as a dashed grey reference.

Figure 3. Distribution of median per-forward-pass latency (batch size 32) across all evaluation records for each model.

Figure 4. Accuracy-versus-cost trade-off (mean in-provider RMSE against mean per-unit training seconds), one point per (model, horizon) combination.

Figure 5. Training and validation MSE curves for LSTM seed=42 across each (source, horizon) combination, on a log-scale y-axis.

List of Abbreviations

ARIMA — Autoregressive Integrated Moving Average

AWS — Amazon Web Services

CC-BY — Creative Commons Attribution (licence)

CPU — Central Processing Unit

CSV — Comma-Separated Values

EC2 — Elastic Compute Cloud

GAN — Generative Adversarial Network

GPU — Graphics Processing Unit

LSTM — Long Short-Term Memory

MAE — Mean Absolute Error

ms — Millisecond(s)

MPS — Metal Performance Shaders

PID — Proportional–Integral–Derivative

RMSE — Root Mean Squared Error

RNN — Recurrent Neural Network

RQ — Research Question

SLA — Service-Level Agreement

VM — Virtual Machine

Abstract

Reactive cloud auto-scalers respond only after a threshold is breached, and provisioning delay can exceed the window in which a workload spike exhausts capacity. Predictive auto-scaling addresses this lag by forecasting near-term demand ahead of provisioning, but published comparisons of the two dominant forecasting architectures, Long Short-Term Memory (LSTM) networks and Transformers, rarely match parameter budgets or test generalisation beyond a single trace. This dissertation evaluated LSTM and Transformer forecasters at a matched budget of approximately 30,000 parameters, trained and tested on heterogeneous Alibaba and Google cluster traces at 5-, 15-, and 30-minute horizons, against a persistence baseline, across three random seeds.

In-provider, only one of twelve trained cells beat persistence in normalised RMSE: the Alibaba-source LSTM at 5 minutes, by 1.6%. Cross-provider zero-shot transfer proved directional rather than symmetric: every model trained on Alibaba and evaluated on Google lost to persistence at all three horizons, by four to ten times the baseline RMSE, while at every horizon at least one model trained on Google and evaluated on Alibaba beat persistence, the Transformer at 5 and 15 minutes and the LSTM at 30. Median inference latency overlapped almost entirely between architectures, but the Transformer took roughly 1.2 to 3 times longer to train than the LSTM. No single architecture dominated across the conditions tested: the direction of cross-provider transfer, not the choice between LSTM and Transformer, determined whether prediction added value over the naive baseline. The findings bound proactive resource-provisioning claims to the two providers, three horizons, and parameter budget evaluated here.

Chapter 1 — Introduction

1.1 Problem Context

Cloud platforms allocate virtual machines and containers elastically in response to fluctuating demand. The default mechanism for this allocation is reactive: a threshold-based, queue-length-based, or PID-controller-based scaler triggers a scaling action only once measured utilisation crosses a defined boundary (Lorido-Botran et al., 2014). The action then lags the demand that provoked it. Provisioning a new virtual machine, including boot time and application warm-up, typically takes 60–180 seconds (Aslanpour et al., 2020). A workload spike can exhaust existing capacity and breach a service-level agreement well before that new capacity comes online.

Predictive auto-scaling addresses this lag directly: rather than waiting for a threshold breach, a predictive scaler forecasts near-term demand and provisions resources ahead of it. Major cloud providers now ship predictive scalers as standard offerings — Google Cloud's Predictive Autoscaler and AWS's Predictive Scaling for EC2 both forecast demand from historical utilisation and pre-provision accordingly (AWS, n.d.; Google Cloud, n.d.). Their existence signals that operators want prediction, not merely faster reaction. What the vendor documentation does not settle is which forecasting architecture delivers that prediction most reliably; that question is empirical, not a matter of product design.

Time-series forecasting research has moved toward learned neural models over the past decade, driven by large-scale benchmarking work such as DeepAR (Salinas et al., 2020). Long short-term memory (LSTM) networks were the first neural architecture to dominate this space; Transformer variants, built on self-attention rather than recurrence, followed and are now widely treated as the field's current default (Wen et al., 2023). Cloud workload forecasting has adopted both architectures individually, but rarely compares them directly under conditions that isolate architecture from model capacity. That comparison, and what it implies for proactive resource provisioning, is the starting point for this dissertation.

1.2 Research Problem

Published comparisons of LSTM and Transformer forecasters for cloud workloads rarely control for model capacity. A Transformer with more parameters than its LSTM counterpart may outperform it for reasons of scale rather than architecture, yet parameter budgets are seldom matched or even reported in this literature (Prachitmutita et al., 2018). Without a matched-budget comparison, a claim that one architecture forecasts more accurately than the other cannot be separated from a claim that it was simply given more capacity to work with.

A second gap concerns generalisation across providers. Almost every published cloud-workload forecasting study trains and evaluates on a single trace, so whether a model fitted on one provider's data forecasts usefully on another provider's data, without retraining, remains comparatively under-examined. This is not a marginal question: operators who migrate workloads, expand into new regions, or run multi-cloud deployments frequently lack historical data from the target environment. A model that generalises zero-shot across heterogeneous traces would be more deployable than one that requires provider-specific retraining, but the evidence for whether either architecture actually does so is thin.

1.3 Aim and Objectives

This dissertation evaluates the generalisability, predictive accuracy, and computational efficiency of two deep-learning architectures — long short-term memory (LSTM) networks and Transformers — for cloud CPU workload forecasting, using heterogeneous cluster traces from Alibaba and Google. The aim translates into four objectives, each addressed by a specific part of the methodology.

The first objective is to implement LSTM and Transformer forecasters at a matched parameter budget, so that any accuracy difference between them can be attributed to architecture rather than model capacity (Chapter 3, §3.7). The second is to preprocess the Alibaba and Google traces into comparable, chronologically ordered slices despite their different measurement units and granularities (Chapter 3, §§3.3–3.4). The third is to evaluate both in-provider accuracy and zero-shot cross-provider transfer at three forecast horizons — 5, 15, and 30 minutes (Chapter 3,

§§3.6 and 3.10). The fourth is to measure training duration and inference latency for both architectures under the same matched-budget condition (Chapter 3, §3.9).

1.4 Research Questions

Four research questions structure the investigation, moving from accuracy to generalisability to cost and finally to synthesis.

RQ1 (accuracy): How do LSTM and Transformer compare in normalised RMSE for cloud CPU forecasting at short horizons of 5, 15, and 30 minutes?

RQ2 (generalisability): To what extent do the two architectures maintain accuracy when transferred zero-shot across heterogeneous cloud traces?

RQ3 (computational efficiency): What is the training-duration and inference-latency trade-off between the two architectures at matched parameter budget?

RQ4 (synthesis): Which architecture presents the most viable predictive framework for proactive resource provisioning at these horizons?

RQ1 establishes whether either architecture forecasts accurately enough to be worth deploying at all. RQ2 asks whether that accuracy, if present, survives a change of provider without retraining. RQ3 sets the accuracy findings against their computational cost. RQ4 draws the first three together into a single answer about deployment viability. Chapter 4 answers RQ1 to RQ3 with the empirical evidence; Chapter 5 synthesises RQ4, and Chapter 6 restates it within the dissertation's conclusions.

1.5 Contributions

The contributions below distinguish between what the study set out to establish and what the completed experiment actually supports. The original intention, following Assessment 1, was a controlled comparison of two forecasting architectures; the empirical contributions that follow are what the 78 recorded evaluations across Alibaba and Google actually show.

The first contribution is evidence on how strong a naive persistence baseline is at these horizons. Of the 24 trained (source, target, model, horizon) cells evaluated, only four beat the persistence baseline in normalised RMSE, and by a narrow margin in most cases — the largest improvement is 9.7 % (Google-source LSTM transferred to Alibaba at the 30-minute horizon), and the smallest is 1.6% (Alibaba-source LSTM at the 5-minute horizon). This matters methodologically: a complex forecasting model should demonstrate improvement over a persistence strategy — one that simply repeats the last observed value — before it can be considered useful for proactive provisioning. At these horizons, on these workloads, most trained cells do not clear that bar.

The second contribution concerns the direction of cross-provider transfer. Zero-shot transfer from Google to Alibaba beats persistence at all three horizons, with the Transformer winning at 5 and 15 minutes and the LSTM winning at 30 minutes. Transfer in the opposite direction, from Alibaba to Google, fails at every horizon for both architectures, with normalised RMSE 4–10 times the persistence baseline. The asymmetry, not the architecture, is the more informative variable: every winning cell in the study evaluates on the Alibaba target, and none evaluates on the Google target. This finding is bounded to the two providers, the preprocessing choices, and the horizons tested here; it does not establish a general rule about cross-provider transfer.

The third contribution concerns computational cost at matched parameter budget. Median inference latency overlaps heavily between the two architectures — 1.7–2.5 milliseconds per batch for the LSTM against 1.9–2.5 milliseconds for the Transformer — so the assumption that Transformer parallelism yields a decisive latency advantage does not hold at this scale. Training duration tells a different story: the Transformer takes roughly 1.2 to 3 times longer to train than the LSTM per unit. The absence of a latency advantage is a negative finding, but a useful one, since Transformer inference speed is often assumed rather than measured.

1.6 Dissertation Structure

Chapter 2 reviews the literature underlying this problem: reactive and predictive auto-scaling, time-series forecasting methods, the theoretical basis of LSTM and Transformer architectures,

the debate over whether Transformers offer a genuine advantage for time-series data, the gap in cross-provider generalisation research, and prior work using the Alibaba and Google traces.

Chapter 3 documents the research design: the two cluster traces and the rationale for treating them as heterogeneous, the preprocessing pipeline for each, the normalisation strategy, the windowing and chronological split procedure with its mechanical leakage test, the LSTM and Transformer architectures at matched parameter budget, the training regime, the evaluation methodology, and the zero-shot cross-provider setup.

Chapter 4 presents the empirical findings: persistence baselines, in-provider accuracy (RQ1), cross-provider transfer (RQ2), and computational cost (RQ3), together with training dynamics and the variance observed across random seeds. This chapter reports what was measured; it does not interpret it at length.

Chapter 5 interprets those findings, relates them to the literature reviewed in Chapter 2 — in particular the debate over Transformer effectiveness for time-series forecasting — considers alternative explanations, and states the limitations of the study before answering RQ4.

Chapter 6 synthesises the three contributions, states the conclusions within the scope the evidence supports, and sets out directions for future work, including the log1p transform deferred from the normalisation strategy used here.

Chapter 2 — Literature Review

2.1 Cloud Auto-Scaling and the Case for Prediction

This chapter reviews the literature that motivates and constrains the experimental design set out in Chapter 3. It moves from the operational case for predictive auto-scaling, through workload forecasting and the two architectures compared here, to the debate over whether Transformer models improve on simpler alternatives, and finally to cross-provider generalisability, where the clearest gap in the literature emerges. The review is organised by argument rather than by author, and each section closes by stating what remains unresolved. It begins with the operational problem that predictive scaling is meant to solve.

Cloud platforms manage resource allocation predominantly through reactive auto-scaling. Lorido-Botran et al. (2014) group these mechanisms into three families: threshold-based scalers that trigger on a fixed utilisation boundary, queue-length-based scalers that respond to backlog, and control-theoretic scalers that adjust provisioning proportionally to the error between measured and target load. Each family shares a structural limitation: the trigger fires only after the monitored metric has already moved, so the scaling action always responds to a change that has already happened rather than anticipating one. Aslanpour et al. (2020) quantify the resulting latency, but the more general point is architectural rather than empirical: a reactive controller, by definition, cannot act before its input signal changes, whatever its response speed once it does.

Predictive scalers address this lag directly by forecasting demand ahead of the provisioning decision. Vendor implementations — Google Cloud's Predictive Autoscaler and AWS's Predictive Scaling for EC2 — and academic proposals ranging from statistical models to LSTM ensembles (Prachitmutita et al., 2018) share a common assumption: that near-term demand is sufficiently predictable from recent history to justify acting on the forecast rather than the observation. That assumption carries a cost of its own. A forecast that overstates demand provisions capacity that goes unused, converting a latency problem into a cost problem; a forecast that understates demand reproduces the reactive scaler's failure mode while adding the forecasting system's own computational overhead. Prediction does not eliminate the provisioning problem; it relocates it, from response speed to forecast accuracy, a question that depends on

model choice, forecast horizon, and workload volatility, and one that vendor documentation does not resolve for any specific model or workload.

The horizons evaluated in this dissertation — 5, 15, and 30 minutes — follow from the provisioning lag itself rather than from forecasting convenience. If instance boot and application warm-up together consume on the order of 1–3 minutes (Aslanpour et al., 2020), a forecast reaching only a few minutes ahead offers little lead time once model inference and provisioning orchestration are accounted for, while a forecast reaching tens of minutes ahead risks losing accuracy to workload volatility before the extra lead time can be used. The three horizons bracket this trade-off rather than assume any point within it is correct in advance; whether accuracy holds as the horizon lengthens is the empirical question RQ1 addresses directly, in Chapter 4.

2.2 Time-Series Forecasting for Cloud Workloads

Time-series forecasting predates deep learning by decades, and the classical toolkit remains a relevant point of comparison. Autoregressive integrated moving-average (ARIMA) models and exponential-smoothing methods such as Holt-Winters decompose a series into trend, seasonal, and residual components and forecast by extrapolating each (Box & Jenkins, 1970; Hyndman & Athanasopoulos, 2021). These methods perform well when a series is close to stationary and its seasonal structure is stable, but they degrade when the underlying process shifts — a regime change, a workload migration, a change in traffic pattern — because the fitted coefficients no longer describe the process generating new observations.

A simpler alternative dominates smooth series more often than the sophistication of ARIMA or neural methods might suggest. The persistence (naive) forecast — predicting that the next value equals the most recently observed one — makes no distributional assumption and requires no fitting. Makridakis et al. (2020), reporting on the M4 competition's evaluation of 61 forecasting methods across 100,000 real-world series, found that naive and simple-exponential-smoothing baselines were difficult for many more complex methods to beat consistently, particularly at short horizons and on series without strong seasonal structure. This finding matters beyond the M4 dataset itself: it establishes persistence not as a strawman but as a benchmark that a

forecasting method must clear before any claim of genuine skill is credible. Cloud CPU utilisation, sampled every few minutes, is exactly the kind of short-horizon, weakly seasonal series for which this result is most relevant — a point returned to when this dissertation's own persistence baseline is introduced in Chapter 3.

Applying either classical or neural methods to a long series requires reframing continuous history as a supervised learning problem: a fixed window of past observations becomes the input, and one or more future steps become the target, with the window slid forward to enumerate training examples. This formulation is what allows a single model to be trained across many machines or tasks rather than fitted separately to each series, and it is the framing adopted by the neural forecasting literature reviewed below. The choice of window length is itself a modelling decision with consequences: a window too short discards relevant history, while a window unnecessarily long adds parameters and training cost without a guaranteed accuracy benefit, particularly for architectures whose per-step cost scales with input length. This trade-off recurs directly in the model architectures reviewed in §2.3 and §2.4.

Neural forecasting methods moved from a research curiosity to the field's default over roughly the last decade, driven by large-scale, multi-series benchmarking rather than any single architectural breakthrough. DeepAR (Salinas et al., 2020) demonstrated that a single recurrent network, trained jointly across thousands of related series, could outperform per-series classical models when enough series shared underlying structure. Wen et al. (2023), surveying the subsequent shift toward Transformer-based forecasters, describe this as a continuation of the same logic — more shared structure, more data, larger models — rather than a departure from it. Long short-term memory (LSTM) networks and Transformer variants are the two architectural families that this progression has left as the current default, and each is examined in turn below.

2.3 LSTM for Sequence Modelling

The long short-term memory (LSTM) network was introduced by Hochreiter and Schmidhuber (1997) to address a specific failure of the recurrent networks that preceded it: gradients propagated backward through many time steps either vanished or exploded, making long-range dependencies effectively unlearnable. Gers et al. (2000) added a forget gate to the original

design, allowing the network to reset its memory rather than being forced to accumulate it indefinitely — a refinement now considered part of the standard LSTM formulation rather than a variant of it.

An LSTM cell maintains two state vectors — a cell state that carries information across time steps with minimal transformation, and a hidden state that is exposed as the cell's output — regulated by three gates. The forget gate determines what proportion of the existing cell state to retain; the input gate determines what new information to write; the output gate determines what proportion of the updated cell state to expose as the hidden state (Sherstinsky, 2020). This gating mechanism is what allows the network to preserve a signal across many steps without the exponential decay that afflicts a plain recurrent unit, and it is why LSTM became the default recurrent architecture for sequence problems for the following two decades.

Applied to cloud workload prediction specifically, the published evidence is thinner and less uniform than the volume of proposed methods suggests. Song et al. (2018) apply LSTM directly to host-load prediction on a Google data-centre trace and report state-of-the-art accuracy relative to their comparison methods. Other papers commonly grouped with LSTM cloud-forecasting work turn out, on inspection, to address adjacent but distinct problems: proposals that use a neural component inside a larger auto-scaling framework (Prachitmutita et al., 2018) are not RMSE-benchmarking studies in the same sense as a forecasting paper, and at least one paper sometimes cited in this context concerns the scheduling of VM migration rather than forecasting accuracy at all. This is an informative finding in its own right: LSTM cloud-workload results are less directly comparable across studies than a single quoted error range would suggest, because dataset, horizon, hidden size, and evaluation protocol vary together and are rarely reported in a form that supports aggregation. Any claim about a typical LSTM error rate for cloud workloads needs the originating paper's specific conditions stated alongside it — a discipline this dissertation applies to its own reported figures in Chapter 4.

LSTM's sequential recurrence is also its principal constraint. Because each hidden state depends on the one before it, computation cannot be parallelised across the time dimension during training, and the fixed-size hidden state must compress an arbitrarily long history into a constant

number of values — a bottleneck that grows more binding as sequence length increases. Empirically, LSTM performance on very long dependencies remains inconsistent even with gating, since the gates mitigate rather than eliminate the underlying compression problem. These two limitations, not any specific empirical failure, motivated the attention-based alternative reviewed next.

2.4 Transformer for Time-Series

Vaswani et al. (2017) proposed self-attention as a replacement for recurrence: rather than passing a compressed state forward one step at a time, every position in a sequence attends directly to every other position within a single layer. This removes the sequential dependency that prevents LSTM training from parallelising across time, at the cost of a computational complexity that scales quadratically with sequence length rather than linearly. For the input windows used in cloud workload forecasting — typically tens of time steps rather than hundreds — this quadratic cost is a minor practical concern, though it becomes material for the much longer sequences Transformer variants were originally designed to handle.

A Transformer encoder block combines multi-head self-attention with a position-wise feed-forward network, each wrapped in a residual connection and layer normalisation. Multiple attention heads allow the model to represent several distinct relationships between positions simultaneously — one head might weight recent steps heavily, another might weight periodic structure — rather than forcing a single attention pattern to capture everything. The feed-forward block, typically expanding the model dimension by a factor of four in the original design, provides the non-linear transformation that attention alone does not; attention is fundamentally a weighted averaging operation, and without an intervening non-linearity the network could collapse toward a linear function of its inputs. How many blocks, heads, and how wide a feed-forward expansion a given study uses is a capacity decision independent of the attention mechanism itself, and it is this decision — not the presence of attention — that determines how directly two Transformer-based studies can be compared.

Because self-attention has no inherent notion of order — every position is treated symmetrically unless told otherwise — positional information must be injected explicitly. Vaswani et al. (2017)

proposed a fixed sinusoidal encoding, added to each input embedding, as an alternative to a learned positional embedding trained alongside the rest of the network. The two approaches trade off differently depending on scale: a learned encoding can in principle fit whatever positional structure a large dataset exhibits, but it consumes parameters and requires enough training data to fit reliably; a fixed encoding costs no parameters and imposes a smooth, predictable structure regardless of dataset size. For architectures operating at a small input length and a constrained parameter budget, a fixed encoding is the more defensible default, since the parameters a learned encoding would consume are not obviously better spent there than elsewhere in the network — a design question returned to directly in Chapter 3.

Later work has revised the vanilla encoder for time-series data specifically. Zhou et al. (2021) introduced ProbSparse attention in Informer to reduce the quadratic cost for long sequences; Nie et al. (2023) proposed PatchTST, which attends over patches of contiguous time steps rather than individual points, and reported consistent improvements over both LSTM and earlier Transformer variants on long-horizon benchmarks. Wen et al. (2023), surveying this body of work, treat patch-based and sparsity-based modifications as the field's current frontier, positioning the original encoder-only architecture as a reference point rather than a competitive endpoint.

The comparison in this dissertation uses that reference-point formulation rather than a variant: an encoder-only Transformer with no autoregressive decoder, consuming a fixed input window and producing a direct multi-step forecast from the final position's representation. This is a deliberate simplification relative to PatchTST or Informer, chosen so that the architecture under test differs from the LSTM baseline primarily in its use of attention rather than in a collection of task-specific modifications whose individual contribution would be difficult to isolate.

2.5 The "Are Transformers Effective?" Critique

The premise that attention-based architectures forecast time series better than simpler alternatives has been directly challenged. Zeng et al. (2023) showed that a single linear layer applied to the lookback window — a model they term LTSF-Linear, with no attention, no recurrence, and orders of magnitude fewer parameters than the Transformer variants it was

compared against — matched or exceeded published Transformer results on nine widely used long-term forecasting benchmarks. The claim is not that Transformers cannot forecast; it is that, on the benchmarks tested, their reported advantage over a linear baseline did not hold up under a controlled comparison.

Zeng et al. (2023) offer a mechanistic account for why this occurs. Self-attention is built to capture long-range, content-dependent relationships between positions, but many real-world time series — smooth, weakly seasonal, dominated by local trend — do not contain relationships of that kind for the model to find. Where the true generating process is close to linear in the recent past, attention's additional representational flexibility becomes a liability rather than an asset: it has more capacity to fit noise in the training data, and the permutation-invariant treatment of positions that positional encoding only partially corrects can degrade the ordering information a purely sequential or linear model preserves automatically. The critique is not that attention is a flawed mechanism in general; it is that attention is mismatched to a specific and common class of signal — one that includes many operational, sensor-derived, and infrastructure-monitoring series.

Subsequent work has qualified rather than overturned this result. Nie et al. (2023) found that PatchTST, which restricts attention to operate over contiguous patches instead of individual time steps, recovered a measurable advantage over LTSF-Linear on the same class of benchmarks — suggesting that the problem lies less with attention itself than with applying it at the granularity of single points on smooth series. The debate has narrowed: not whether attention can ever help, but at what granularity and on what kind of series it does.

Cloud CPU utilisation traces sit close to the type of signal Zeng et al. (2023) describe: short-horizon, weakly seasonal, and dominated by recent history rather than long-range structure, particularly at the 5-to-30-minute horizons this dissertation evaluates. Whether that similarity is enough to reproduce the linear-model result specifically for cloud workload forecasting — and whether it holds equally for LSTM, which shares none of attention's permutation-invariance but is also not a linear model — is not established by the general time-series literature; the benchmarks Zeng et al. (2023) and Nie et al. (2023) use are drawn from energy, traffic, and

weather domains, not cloud infrastructure telemetry. This dissertation tests both architectures against a persistence baseline on cloud CPU traces specifically, at matched parameter budget, which is the direct way of establishing whether the debate above extends to this domain rather than assuming it does. Chapter 5 returns to this question once the results are available.

2.6 Cross-Provider Generalisation: The Gap

A model trained on one data distribution and evaluated on another is performing transfer, and the theoretical literature on domain adaptation gives a general account of when this should be expected to work. Pan and Yang (2010) frame the question as one of source and target distributions sharing enough structure that a model fitted to the source retains predictive value on the target; Ben-David et al. (2010) formalise this in terms of a measurable distance between the two distributions, with generalisation bounds that loosen as that distance grows. Neither framework specifies in advance how much distance two cloud providers' CPU-utilisation traces exhibit, or whether that distance is small enough for the bounds to be practically informative rather than merely theoretically true.

Empirical evidence on cross-provider transfer for cloud workload forecasting specifically is limited. The great majority of published cloud-forecasting studies train and evaluate on a single trace, so the question of whether a fitted model retains accuracy on a different provider's data is simply not addressed by most of the literature reviewed in §2.3 and §2.4 — not because it was tested and found unimportant, but because it falls outside the scope those studies set for themselves. Where cross-dataset evaluation does appear, it is typically accompanied by some form of adaptation. Arbat et al. (2022), for instance, use an adversarial training scheme (a Wasserstein GAN paired with a Transformer generator) to align source and target representations before evaluating across datasets, which tests a materially different question from zero-shot transfer, where no target-domain data or fine-tuning is available at all. Cross-provider transfer without adaptation remains comparatively under-examined as a result.

The practical stakes of this gap are not hypothetical. An operator expanding into a new region, migrating workloads between providers, or running a genuinely multi-cloud deployment frequently has abundant historical data from one environment and none from the other at the

point a forecasting model is needed. Fine-tuning is only available where target-domain data already exists in sufficient quantity — precisely the condition that does not hold in these scenarios. If a model trained on one provider's traces cannot forecast usefully on another's without that data, the practical value of the model is confined to the single environment it was trained on, which is a considerably narrower claim than "this architecture forecasts cloud workloads well" would suggest. The distinction between in-distribution accuracy and cross-domain generalisability is not a technicality — it determines whether a published result says anything about deployability outside the study's own dataset.

This dissertation addresses the gap directly rather than by proxy: both architectures, once trained on one provider's trace, are evaluated zero-shot on the other provider's trace, with no fine-tuning and no shared normalisation statistics between source and target beyond what each provider's own scaler establishes. This design isolates the question the literature above leaves open — whether either architecture, trained under a matched parameter budget, retains forecasting value when the input distribution changes — as a directly testable empirical question rather than a theoretical one. Chapter 4 reports what this test found; the interpretation is developed in Chapter 5.

2.7 Cluster Traces as Benchmarks

The Google 2011-2 cluster trace records 29 days of scheduling and resource-usage data from a cluster of roughly 12,500 machines, with task-level usage tables reported at a nominal five-minute granularity (Reiss et al., 2011). Reiss et al. (2012) characterise the workload running on this cluster as highly heterogeneous — a mix of long-running services and short batch tasks with markedly different resource footprints — which the trace format supports by recording usage per task rather than only per machine, at the cost of a substantially larger and more fragmented dataset than a machine-level trace of the same duration.

The Alibaba v2018 machine-usage trace covers a shorter window — eight days — across around 4,000 machines, reporting per-machine CPU utilisation as a percentage rather than task-level resource consumption (Guo et al., 2019). Cheng et al. (2018) describe the co-located batch and long-running workloads on this cluster as a deliberate design choice by the operator to improve

utilisation, which produces a different volatility profile from a cluster where task placement is not similarly co-optimised.

Both traces are widely used as forecasting benchmarks, but almost always separately. The instrumentation itself is not directly comparable: Alibaba's cpu_util_percent is a bounded, machine-level percentage, while Google's task-level usage figures are reported in CPU-core-seconds per second, a different physical quantity measured at a different aggregation level. A model trained on one is not, by construction, being trained on a like-for-like measurement of the other, which is precisely why treating the two traces as a genuine cross-provider test — rather than as two convenient but interchangeable datasets — matters for what a transfer result can be said to show.

Google-trace forecasting studies report normalised RMSE figures broadly consistent with the range discussed in §2.3, using the task-usage tables at varying levels of temporal aggregation (Song et al., 2018; Kumar & Singh, 2018). Alibaba-trace work has more often focused on characterising the workload itself — co-location patterns, utilisation distributions, scheduling behaviour (Cheng et al., 2018) — than on forecasting accuracy specifically, which leaves fewer directly comparable published forecasting results for this trace than for the Google release. This asymmetry in what has been published, not any property of the traces themselves, is one further reason a controlled comparison run on both traces under identical preprocessing and evaluation is more informative than relying on the separately published results for each.

Chapter 3 — Methodology

3.1 Research Paradigm and Design

This dissertation adopted a quantitative, computational, experimental methodology conducted entirely on secondary data, continuous with Assessment 1's own commitment to "a quantitative computational experimental methodology based entirely on secondary data." The paradigm is positivist in orientation: the research questions concern measurable properties of trained models, normalised RMSE, training duration, and inference latency, evaluated against falsifiable predictions rather than an interpretive criterion. RQ1 to RQ3 each asked for a quantity to be measured and compared; RQ4 synthesised those measurements.

Several design commitments followed. Random seeds were fixed and results reported across three repeated runs per condition, distinguishing reported differences from sampling variation. Model capacity was matched at the parameter-count level between architectures, isolating architecture as the variable under comparison for RQ1 and RQ3. Splits were constructed chronologically rather than at random, preserving temporal ordering. Evaluation metrics were computed per unit and then averaged, rather than pooled globally, so units contributing more windows did not dominate the reported error. The persistence baseline, the three horizons, and the zero-shot cross-provider evaluation follow the same controlled-comparison logic, described in §§3.7, 3.8, and 3.11. Each commitment narrows what a result can be attributed to.

3.2 Ethical Considerations and Data Licensing

This study uses only secondary data from two publicly released cluster traces. The Google 2011-2 task_usage data is distributed by Google under the Creative Commons CC-BY 4.0 licence (Google, n.d.). The Alibaba v2018 machine_usage data is released by Alibaba Group for academic research and study purposes only, via a short survey acknowledgement, under the terms published in the Alibaba Cluster Trace Program repository (Alibaba Cluster Trace Program, n.d.). Both traces are fully anonymised at source: machine, container, task, job, and user identifiers are opaque, and the data contains no personally identifiable information, no application content, and no access-pattern information about end users. No human-subjects ethics review was required.

3.3 Datasets and Rationale

This dissertation drew on two publicly available cluster traces: the Alibaba Cluster Trace v2018 machine_usage table (Guo et al., 2019) and the Google Cluster Data 2011-2 task_usage table (Reiss et al., 2011). Both traces record resource-utilisation telemetry from production compute clusters over multi-day windows and are widely used as forecasting benchmarks, established in Chapter 2. Assessment 1 proposed forecasting cloud CPU and memory workloads jointly; this study narrowed that to CPU utilisation only, a scope decision made to keep preprocessing and the matched-parameter comparison tractable across two traces that already differ in instrumentation, rather than a finding forced by the data itself. Memory forecasting remains available as future work.

The two traces are heterogeneous by construction, and this heterogeneity is a deliberate feature of the design rather than an incidental limitation. Alibaba's cpu_util_percent is a bounded percentage of machine capacity, sampled per machine; Google's cpu_usage_mean is a mean CPU-core-seconds-per-second rate, sampled per task. The two quantities are not interchangeable measurements of the same underlying process, and Assessment 1 was explicit that the datasets would "not be treated as semantically identical." Testing whether a model trained on one provider's trace retains forecasting value on the other's is only a meaningful test of generalisability if the two traces genuinely differ in instrumentation, granularity, and workload composition; two traces measuring the same quantity the same way would collapse the cross-provider question into a same-distribution test, which is not what RQ2 asks. Per-dataset normalisation, described in §3.6, is the mechanical acknowledgement of this heterogeneity.

3.4 Alibaba Preprocessing

The Alibaba machine_usage table decompresses to approximately 15 GB, a size that risked exceeding the disk space available during preprocessing. Rather than extracting the archive fully before reading it, the raw CSV was streamed directly out of the compressed tarball and processed in fixed-size chunks, so that only a bounded portion of the file was held in memory at any point and the full 15 GB was never written to disk uncompressed.

This streaming approach used Python's tarfile module to read the compressed member directly, decoded as a text stream and parsed with pandas in chunks of 1 million rows at a time. Each chunk was filtered before the next was read, so that only rows meeting the study's inclusion criteria were retained in memory, and the output was written incrementally to a single compact file rather than accumulated in full before writing. This kept peak memory use bounded regardless of the size of the source archive, which was necessary given that the available working memory was small relative to the uncompressed table.

Assessment 1 left the specific contiguous window unspecified, committing only to selecting a representative period according to available computational resources. The most direct reading of that commitment, a period starting at the beginning of the trace, was tested first and rejected on empirical grounds. A coverage diagnostic across all eight days of the Alibaba trace found that no machine had complete five-minute-bin coverage across the first two days: the maximum number of bins present for any single machine was 271 out of 288 on day 0 and 225 out of 288 on day 1, with total row counts around 40 % lower than the steady-state rate observed from day 2 onward. This pattern is consistent with the early period of the trace capturing a cluster still ramping up its own monitoring coverage, rather than steady operation. Day 2 was the first day for which complete coverage was available at scale: 3,988 of the 4,018 machines present in that window had all 288 bins recorded. Selecting day 2 satisfied Assessment 1's requirement for a representative period more directly than the naive first-day default would have, by using a period the data could actually support without gap imputation.

Within the selected day-2 window, machines were retained only if all 288 five-minute bins were present for the full period, a completeness gate that avoided the need for gap imputation entirely. Of the machines present in that window, 3,988 met this criterion. The first 500 of these, ordered lexicographically by machine identifier, were selected as the working sample. This selection procedure was deterministic: given the same trace and the same completeness gate, any reader repeating the procedure would arrive at the same 500 machines, without needing to store or share the specific identifiers separately. Deterministic selection was preferred over random sampling from the eligible pool because it removed an additional source of seed-dependent variation from a step that the study's random-seed controls, described in §3.1, were not designed to cover.

3.5 Google Preprocessing

The Google 2011-2 task_usage data is distributed in a large number of separate compressed parts, each covering only a fraction of the full trace. An initial candidate set of six parts was evaluated first and found to span approximately 8.36 hours of trace time, insufficient to construct a complete 24-hour forecasting window with enough surrounding data for a chronological 70/15/15 split at the input-window-plus-horizon level. To provide sufficient contiguous coverage for this requirement, 20 parts, covering approximately 27.84 hours in total, were acquired from the trace's public repository.

As with the Alibaba trace, the specific window within this broader span was selected empirically rather than assumed. A coverage diagnostic across the available 27.84 hours found that the final, partial day, covering only 14 hours, had no task with complete five-minute-bin coverage for a full 24-hour period. The preceding complete day, day 28 of the trace, had 82,568 tasks with full coverage, more than 165 times the 500-task sample needed. The window used was the 24-hour period corresponding to day 28, selected because it was the most recent complete day available within the acquired parts and offered ample choice of fully covered tasks, avoiding both partial-day gaps and the need to select from a thin pool of eligible tasks.

The target variable was identified by direct verification against the trace's official schema documentation rather than assumed from a column position alone. The 19-column task_usage table places two distinct CPU-related fields at different positions: a mean CPU usage over each measurement bin at index 5, and a maximum CPU rate within the bin at index 13. These are different quantities: the mean describes typical utilisation over the bin, while the maximum describes a peak burst rate. Index 5, the mean, is the field used throughout this study and is referred to as cpu_usage_mean; it is the closer counterpart to Alibaba's own bin-averaged utilisation figure, since both describe typical rather than peak load, which is the appropriate quantity for a study concerned with routine demand forecasting rather than burst detection. Index 13, the peak-rate field, would have answered a different question and was not used.

A sampling-quality filter based on the trace's own recorded sample-portion field was evaluated as a candidate preprocessing step, since a low sample portion would indicate a bin built from an

incomplete set of underlying observations. A full scan of the acquired parts, covering 45,791,550 rows, found this field uniformly zero across every row, meaning it carries no discriminating information in this particular release of the trace and cannot be used to distinguish well-sampled bins from poorly-sampled ones. Applying a filter that excluded rows below a sample-portion threshold would, in this release, have discarded the entire dataset. Rows were instead filtered only for validity: retained where the CPU usage figure was finite and non-negative, and dropped otherwise. This is a narrower filter than a sampling-quality check would have been, and the finding that the field is uninformative in this release is itself worth noting for any future work drawing on the same data.

3.6 Normalisation

Each provider's data was normalised using Min-Max scaling to [0, 1], fitted separately for Alibaba and Google on the training portion only, then applied without refitting to that provider's validation and test data. Fitting on training data alone prevented statistics from the validation and test periods shaping the scaler, which would otherwise let information the model was not meant to see during training influence how every observation was scaled. A normalised value of 0.5 on Alibaba and 0.5 on Google still describes two different physical quantities; per-dataset scaling is a modelling convenience, not a claim of measurement equivalence.

The Google target's raw distribution is heavily compressed toward zero (mean 0.0123, standard deviation 0.0278, maximum 0.381), motivating consideration of a log1p transform before Min-Max scaling to spread the distribution further. The supporting configuration was implemented but not activated for the reported experiment; evaluating a log1p-transformed Google target remains future work, discussed in Chapter 6.

3.7 Windowing and Chronological Splits

Each forecasting example used 24 five-minute steps of history, corresponding to 2 hours, as input, and produced a forecast at one of three horizons: 1 step (5 minutes), 3 steps (15 minutes), or 6 steps (30 minutes) ahead. These horizons were fixed before evaluation began and applied identically across both providers and both trained architectures, so that any accuracy difference

reported in Chapter 4 reflects the model or the provider rather than an inconsistency in what was being forecast.

Each unit's 288-bin timeline (one five-minute bin per interval across the full 24-hour window) was split chronologically into training, validation, and test portions in a 70/15/15 ratio before any forecasting windows were constructed, giving 201 training bins, 43 validation bins, and 44 test bins per unit. Splitting was applied to the raw timeline first, and windows were then built separately within each portion, rather than building windows across the full timeline and splitting the resulting set of windows afterward. This ordering matters: building windows first and splitting afterward can place a window's input steps in one portion and its target step in another, allowing a training example to look ahead into data meant to represent the future. Splitting the timeline before windowing rules this out by construction.

This guarantee was also verified mechanically rather than assumed from the split logic alone. An automated check examined every training window constructed across both providers and all three horizons and confirmed that no window's target step fell beyond the boundary of the training portion; the equivalent check was applied to the validation and test windows against their own boundaries. This check covered 261,000 training windows in total, which is a materially stronger guarantee than inspecting a handful of examples by hand, since a boundary error affecting only a small fraction of windows could easily be missed by manual spot-checking but would be caught by an exhaustive automated pass. Chronological splitting without random shuffling is standard practice for time-series evaluation, since random splitting would let a model train on data from after the point it is being asked to forecast; the automated check makes that standard practice auditable rather than simply asserted.

3.8 Model Architectures

Three forecasting approaches were implemented and compared under an identical input–output contract: a persistence baseline, a long short-term memory (LSTM) network, and a Transformer encoder. Each model consumed the same 24-step univariate input window described in §3.6 and produced a forecast spanning the relevant horizon, so that no model received more context, a different input representation, or a different target definition than the others. This shared contract

meant that any difference in reported accuracy could be attributed to the forecasting approach itself rather than to an inconsistency in what each model was given to work with.

The persistence baseline forecast the value at each future step as equal to the most recently observed value, $\hat{y}[t+h] = y[t]$, for every horizon h. It had no trainable parameters and required no training: given the same test data, its output was fully deterministic. Persistence served as the reference against which the two trained models needed to demonstrate improvement; a trained model that could not do better than repeating the last observation was not adding forecasting value.

The LSTM forecaster used two recurrent layers with a hidden size of 48 and a dropout rate of 0.1 between layers, processing input sequences batch-first. The final layer's hidden state at the last input step was passed to a linear layer that projected it to the forecast horizon. At the longest horizon evaluated, 6 steps, this architecture had 28,902 trainable parameters, a figure verified by direct measurement of the constructed model rather than calculated by hand.

The Transformer forecaster used an encoder-only architecture with two encoder blocks, four attention heads, a model dimension of 44, and a feed-forward dimension of 88, with the same 0.1 dropout rate and ReLU activation, layer normalisation applied after each sub-block. Positional information was added using a fixed sinusoidal encoding rather than a learned one, contributing no trainable parameters of its own. At the longest horizon evaluated, this architecture had 32,302 trainable parameters, again verified by direct measurement rather than assumed from the configuration alone.

Matching parameter count between the two trained architectures was a deliberate design commitment, introduced in §3.1, intended to isolate architecture as the variable under comparison rather than allowing one model to simply have more capacity than the other. An initial specification targeted a hidden size of 64 for the LSTM and a model dimension of 64 for the Transformer, aiming for roughly comparable budgets of around 30,000 and 35,000 parameters respectively. Direct measurement of models built to these widths found 50,822 parameters for the LSTM and 67,462 for the Transformer, both substantially above their intended targets and 33 % larger for the Transformer than the LSTM, an imbalance that would have

undermined a matched-budget comparison had it gone unnoticed. The widths were reduced to the values reported above, bringing the two architectures to 28,902 and 32,302 parameters respectively, a ratio of 1.12 rather than 1.33.

3.9 Training Regime

Both trainable models were optimised using mean squared error as the training loss and the Adam optimiser (Kingma & Ba, 2015) with a learning rate of 0.001. Mean squared error was chosen as the training objective because it penalises larger errors more heavily than mean absolute error, which suits a regression task where large deviations in predicted CPU utilisation are more operationally costly than small ones; RMSE and MAE, both derived from the same residuals, were retained as the metrics reported for evaluation.

Training used early stopping on validation loss, with a maximum of 50 epochs and a patience of 10 epochs without improvement before training halted (Prechelt, 1998). This bounded total training time while allowing models that continued to improve to keep training beyond an arbitrary fixed epoch count, and prevented models whose validation loss had plateaued from training long past the point where further epochs added value.

The model retained for evaluation was not necessarily the model produced by the final training epoch. Instead, the state of the model at whichever epoch had achieved the lowest validation loss during training was restored before evaluation, whether or not early stopping had triggered by that point. This distinction matters because a model can continue training past its point of best generalisation, with training loss still falling while validation loss rises, a pattern indicating overfitting to the training data specifically. Evaluating the terminal-epoch model in that situation would report worse test-set error than the model was actually capable of, inflating the apparent gap to persistence for reasons unrelated to the architecture itself. This policy was verified mechanically: an automated check re-evaluated the returned model on the validation set and confirmed its loss matched the recorded best validation loss for that run, rather than the loss at the final epoch.

Each trainable configuration was run with three fixed random seeds, 42, 123, and 2024, following Assessment 1's commitment to fixed seeds and repeated runs. Reporting a mean and

sample standard deviation across these three runs, rather than a single run, allowed reported differences to be distinguished from ordinary run-to-run variation.

No hyperparameter search was performed. Architectural and optimisation hyperparameters were fixed at the values described above. A tuning sweep would have confounded the matched parameter-budget comparison central to RQ1 and RQ3, and was outside the scope this study committed to from the outset. The width adjustment described above was a parameter-budget correction to hit a stated target, not a hyperparameter search for the best-performing configuration.

3.10 Evaluation Methodology

Prediction accuracy was measured using RMSE and MAE, the two metrics Assessment 1 committed to for evaluating prediction error. Rather than pooling every prediction from every machine or task into a single calculation, each unit's residuals were flattened across its own forecasting windows and horizon steps, and RMSE and MAE were computed for that unit alone; the mean and sample standard deviation of these per-unit values were then reported across the full set of units. This differs from concatenating every unit's residuals into one global calculation, which would let units with more forecasting windows dominate the result. Per-unit-then-averaged aggregation instead weights each machine or task equally, regardless of how many windows it happened to contribute.

Inference latency was measured separately from accuracy, using 1,000 timed forward passes at a fixed batch size of 32, preceded by 50 discarded warm-up iterations to avoid measuring one-off initialisation cost. Because the hardware backend used for training and evaluation can execute operations asynchronously, each timed pass was bracketed by an explicit synchronisation step before and after timing, ensuring that the recorded duration reflected the computation itself rather than the time taken merely to queue it. Median latency and the 25th and 75th percentiles were recorded for each evaluation.

Median and interquartile range were reported for latency rather than mean and standard deviation, a deliberate choice given how GPU-style kernel scheduling behaves under sustained use. Occasional scheduling delays and kernel-launch variability produce latency distributions

with a long tail of atypically slow passes rather than a symmetric spread around a central value; a mean calculated over such a distribution is pulled upward by these outliers in a way that misrepresents typical performance, while the median and interquartile range are largely unaffected by them. This choice reports what a typical forward pass costs, rather than a figure inflated by infrequent scheduling anomalies unrelated to the model itself.

3.11 Cross-Provider Setup

Zero-shot cross-provider evaluation, the second stage of the two-stage design set out in Assessment 1, required a decision about how the untrained provider's data should be scaled at evaluation time. Every trained model, regardless of which provider it was trained on, was evaluated against both providers' test sets, with each test set normalised using that provider's own Min-Max scaler rather than the scaler the model was trained under. A model trained on Alibaba data sees Google's test inputs scaled to Google's own value range at evaluation time, not remapped through Alibaba's scaler. This tests whether learned temporal structure generalises to a target provider's normalised distribution shape, rather than testing whether the model can also cope with an unfamiliar scaling convention layered on top of an unfamiliar workload. Given that Alibaba's utilisation percentage and Google's core-seconds-per-second measure genuinely different quantities, conflating scale mismatch with generalisation failure would have made the two providers harder to compare, not easier.

This design is also the source of the study's seventy-two trained evaluation records: each of the thirty-six training units (two architectures, two source providers, three horizons, three seeds) is evaluated against both provider test sets under this scheme, producing one in-provider and one cross-provider record per unit.

3.12 Environmental Setup

All training and evaluation were performed on a MacBook Pro with an Apple M1 chip, using PyTorch's Metal Performance Shaders (MPS) backend for GPU acceleration; data preprocessing ran on CPU. The full experimental run totalled 5 hours 28 minutes of wall-clock time across all 78 records. Three intermittent multi-minute stalls were observed during sustained MPS load: gaps of 9, 27, and 12 minutes between epochs in three separate training units. The process

remained active throughout each stall, with no errors logged and no malformed output, consistent with macOS thermal throttling or MPS scheduling delays rather than a fault in the training loop. Cumulative stall time added roughly 15% to total wall clock without affecting result correctness.

Chapter 4 — Findings

4.1 Overview of Experimental Results

Chapter 4 reports the results of the experimental run described in Chapter 3, without yet interpreting what those results mean; interpretation is reserved for Chapter 5. The run produced 78 result records: six persistence-baseline evaluations, one per target-provider/horizon combination, and 72 trained-model evaluations from 36 unique training runs, each evaluated against both providers' test sets under the zero-shot scheme described in §3.11. The full run took 5 hours, 28 minutes and 8 seconds of wall-clock time. Of the 36 training runs, 34 (94%) stopped early on validation loss; the remaining two ran the full 50-epoch schedule because the flat Google target signal reset their patience counters late in training, discussed further in §4.6.

The chapter proceeds in six stages. Section 4.2 establishes the persistence baseline as the reference floor for every trained-model result. Section 4.3 reports in-provider accuracy (RQ1); section 4.4 reports cross-provider zero-shot accuracy (RQ2), containing the chapter's headline finding; section 4.5 reports training duration and inference latency (RQ3). Section 4.6 examines training dynamics across the two providers, and section 4.7 reports variance across the three random seeds and the study's reproducibility guarantees. Throughout, the aim is to state what was measured; why it takes this shape is addressed in Chapter 5.

4.2 Persistence Baselines

The persistence baseline forecasts each future value as equal to the most recently observed value, and requires no training. Its purpose in this study is not descriptive but evaluative: it is the reference every trained-model result in this chapter is measured against, and a trained model's forecasting value is established by beating it, not merely by achieving a low RMSE in isolation. Persistence results are reported once per target-provider-horizon combination, since the baseline is deterministic and produces no variation across seeds to average over. Table 1 summarises all six persistence evaluations.

Target	Horizon (min)	RMSE (norm)	MAE (norm)	RMSE (denorm)	MAE (denorm)	Latency median (ms)
Alibaba	5	0.04193	0.03332	4.1523	3.2995	0.272
Alibaba	15	0.04962	0.03899	4.9139	3.8612	0.268
Alibaba	30	0.05659	0.04557	5.6046	4.5126	0.271
Google	5	0.00165	0.00124	0.0006	0.0005	0.249
Google	15	0.00199	0.00149	0.0008	0.0006	0.260
Google	30	0.00216	0.00163	0.0008	0.0006	0.275

Table 1. Persistence baselines by target provider and forecast horizon.

On the Alibaba target, persistence RMSE grew monotonically with horizon: 0.04193 (normalised) at 5 minutes ahead, 0.04962 at 15 minutes, and 0.05659 at 30 minutes. On the scale of the original cpu_util_percent measurement, these correspond to RMSEs of 4.15, 4.91, and 5.60 percentage points respectively. Persistence error growing with horizon is the expected pattern: the further ahead the forecast, the more the utilisation value can plausibly have moved from where it last stood.

On the Google target, persistence RMSE was two orders of magnitude smaller: 0.00165 at 5 minutes, 0.00199 at 15 minutes, and 0.00216 at 30 minutes (normalised). This is a genuine property of the Google cpu_usage_mean distribution rather than an artefact of the normalisation procedure: the raw distribution on the selected window has a mean of 0.0123 and a standard deviation of 0.0278, meaning the signal barely moves from bin to bin regardless of how it is scaled. A model, or a persistence baseline, forecasting a series that rarely departs from a narrow band around a low value will report a small RMSE almost by construction.

Persistence forward-pass latency, measured identically to the trained models' latency in §3.10, ranged from 0.249 to 0.275 milliseconds per batch of 32 across the six evaluations. The Alibaba-target evaluations clustered tightly, at 0.272, 0.268, and 0.271 milliseconds for the 5, 15, and 30-minute horizons respectively; the Google-target evaluations were slightly more spread, at 0.249, 0.260, and 0.275 milliseconds. Because persistence has no parameters and performs no computation beyond indexing the most recent observation, this figure is close to the minimum achievable latency on this hardware for producing a forecast at all. Any trained model's inference

latency, reported in §4.5, is measured against this floor rather than against zero, since some overhead is unavoidable even for the simplest possible forecasting rule.

4.3 In-Provider Prediction Accuracy (RQ1)

This section addresses RQ1: how do LSTM and Transformer compare in normalised RMSE for cloud CPU forecasting at 5, 15, and 30-minute horizons? It considers only the twelve in-provider cells, where the model was trained and evaluated on the same provider's data (Alibaba → Alibaba and Google → Google), leaving the twelve cross-provider cells to §4.4. Each cell reports the mean normalised RMSE across three seeds, compared against the persistence baseline established in §4.2. Table 2 reports the full set of in-provider results; Figure 1 plots them by condition and horizon.

Condition	Model	Horizon (min)	RMSE norm (mean ± std)	MAE norm (mean ± std)
Alibaba → Alibaba	LSTM	5	0.04125 ± 0.00145	0.03389 ± 0.00155
Alibaba → Alibaba	LSTM	15	0.05266 ± 0.00338	0.04525 ± 0.00327
Alibaba → Alibaba	LSTM	30	0.05949 ± 0.00793	0.05082 ± 0.00716
Alibaba → Alibaba	Transformer	5	0.04358 ± 0.00052	0.03636 ± 0.00080
Alibaba → Alibaba	Transformer	15	0.05044 ± 0.00330	0.04289 ± 0.00323
Alibaba → Alibaba	Transformer	30	0.05684 ± 0.00382	0.04805 ± 0.00353
Google → Google	LSTM	5	0.00219 ± 0.00030	0.00180 ± 0.00031
Google → Google	LSTM	15	0.00275 ± 0.00022	0.00229 ± 0.00026
Google → Google	LSTM	30	0.00293 ± 0.00017	0.00246 ± 0.00019
Google → Google	Transformer	5	0.00178 ± 0.00021	0.00140 ± 0.00021
Google → Google	Transformer	15	0.00228 ± 0.00016	0.00177 ± 0.00010
Google → Google	Transformer	30	0.00253 ± 0.00009	0.00201 ± 0.00008

Table 2. In-provider trained-model results (Alibaba → Alibaba, Google → Google), mean ± sample standard deviation across three seeds.

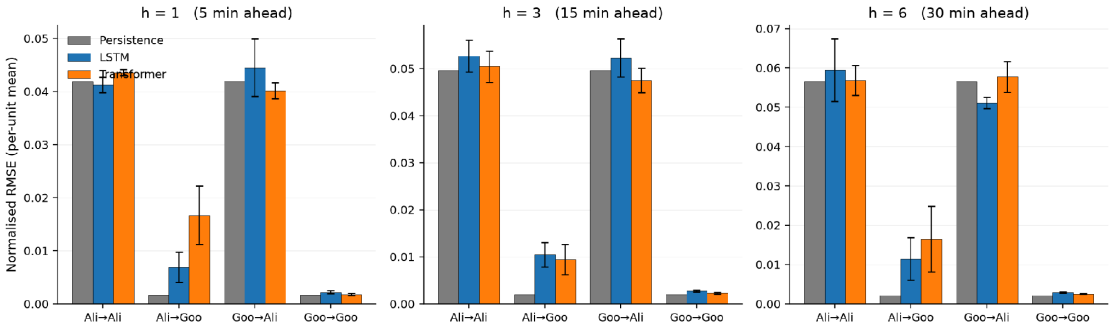


Figure 1. Normalised per-unit RMSE by source-target condition (x-axis) and model (grouped bars), one panel per forecast horizon (5, 15, 30 minutes ahead). Bars show the mean over the three random seeds {42, 123, 2024}; error bars are one sample standard deviation. Persistence has no seed variation (deterministic given the test series). The persistence baseline is roughly matched by the trained models in-provider (Ali→Ali, Goo→Goo) but is decisively better than either trained model on the cross-provider Alibaba→Google direction at every horizon.

On the Alibaba target, only one of the six in-provider cells beat persistence: the LSTM at 5 minutes ahead, with a mean RMSE of 0.04125 (4.09 percentage points denormalised) against a persistence baseline of 0.04193 (4.15 percentage points), an improvement of 0.0007, or approximately 1.6%. This is the only cell among all twelve in-provider combinations in which a trained model outperformed persistence. At the two longer horizons the LSTM lost to persistence, by 0.0030 at 15 minutes (mean RMSE 0.05266, a loss of 6.0% relative to persistence) and by 0.0029 at 30 minutes (mean RMSE 0.05949, a loss of 5.1%).

The Transformer did not beat persistence at any Alibaba in-provider horizon, though its distance from persistence narrowed as the horizon lengthened. At 5 minutes its mean RMSE was 0.04358, a loss of 0.0016, or 3.8% relative to persistence; at 15 minutes, 0.05044, a loss of 0.0008, or 1.6%; at 30 minutes, 0.05684, a loss of only 0.0003, or 0.5%, within a fraction of a percentage point of the persistence baseline in both absolute and relative terms.

On the Google target, no trained model beat persistence at any horizon, across either architecture. The six in-provider deltas ranged from 0.0001, an effective statistical tie for the Transformer at 5 minutes (mean RMSE 0.00178 against a persistence baseline of 0.00165, a relative loss of 7.9%), to 0.0008 for the LSTM at both 15 and 30 minutes (mean RMSEs of 0.00275 and 0.00293 against persistence baselines of 0.00199 and 0.00216, relative losses of 38.2% and 35.6% respectively). The Transformer's losses at 15 and 30 minutes were smaller in both absolute and relative terms, 0.0003 (14.6%) and 0.0004 (17.1%) respectively (mean RMSEs 0.00228 and

0.00253). The LSTM's relative losses on Google, all above 30%, stand out against how small the absolute numbers look: because the persistence baseline itself is so small, a trained model that is only fractionally worse in absolute RMSE can still be a third worse in relative terms.

Because the persistence baseline on Google is already very small, as established in §4.2, the absolute deltas above are numerically tiny even though the trained models never close the small remaining gap. Across all twelve Google-target combinations considered in this chapter (six in-provider here, six cross-provider in §4.4), zero cells beat persistence; this pattern is addressed further in §4.4.

Comparing the two trained architectures directly within each provider, model preference is not fixed across horizons. On Alibaba, the LSTM had the lower mean RMSE at 5 minutes (0.04125 against 0.04358), but the Transformer had the lower mean RMSE at both 15 minutes (0.05044 against 0.05266) and 30 minutes (0.05684 against 0.05949). On Google, the Transformer had the lower mean RMSE than the LSTM at every in-provider horizon, by a wider relative margin than on Alibaba, though both remained above persistence throughout. Which architecture is preferable in-provider depends on both the target provider and the horizon, not on a single dominant model; on this evidence alone, the Transformer is the more consistent in-provider performer of the two, even though it never actually beats persistence.

4.4 Cross-Provider Generalisation (RQ2) — Headline

This section addresses RQ2: to what extent do the two architectures maintain accuracy when transferred zero-shot across heterogeneous cloud traces? It reports the twelve cross-provider cells, in which a model trained on one provider's data was evaluated, without any further training, against the other provider's test set, using that provider's own scaler as described in §3.11. This is the headline result of the dissertation: transfer performance in one direction bears almost no resemblance to transfer performance in the other. Table 3 reports the full set of cross-provider results, Table 4 isolates the cells that beat persistence, and Figure 2 plots RMSE against horizon for all four source-target combinations against their respective persistence baselines.

Condition	Model	Horizon (min)	RMSE norm (mean ± std)	MAE norm (mean ± std)
Alibaba → Google	LSTM	5	0.00690 ± 0.00285	0.00655 ± 0.00288
Alibaba → Google	LSTM	15	0.01045 ± 0.00257	0.00944 ± 0.00275
Alibaba → Google	LSTM	30	0.01148 ± 0.00543	0.01056 ± 0.00522
Alibaba → Google	Transformer	5	0.01671 ± 0.00553	0.01642 ± 0.00552
Alibaba → Google	Transformer	15	0.00941 ± 0.00324	0.00741 ± 0.00341
Alibaba → Google	Transformer	30	0.01645 ± 0.00835	0.01459 ± 0.00744
Google → Alibaba	LSTM	5	0.04452 ± 0.00542	0.03683 ± 0.00540
Google → Alibaba	LSTM	15	0.05233 ± 0.00402	0.04463 ± 0.00408
Google → Alibaba	LSTM	30	0.05111 ± 0.00147	0.04289 ± 0.00168
Google → Alibaba	Transformer	5	0.04016 ± 0.00149	0.03189 ± 0.00143
Google → Alibaba	Transformer	15	0.04750 ± 0.00260	0.03863 ± 0.00280
Google → Alibaba	Transformer	30	0.05771 ± 0.00393	0.04816 ± 0.00359

Table 3. Cross-provider zero-shot trained-model results (Alibaba → Google, Google → Alibaba), mean ± sample standard deviation across three seeds.

Source → Target	Model	Horizon (min)	Δ vs persistence (norm)	% better
Alibaba → Alibaba	LSTM	5	-0.0007	1.6%
Google → Alibaba	LSTM	30	-0.0055	9.7%
Google → Alibaba	Transformer	5	-0.0018	4.2%
Google → Alibaba	Transformer	15	-0.0021	4.3%

Table 4. The four (source, target, model, horizon) cells that beat the persistence baseline in normalised RMSE.

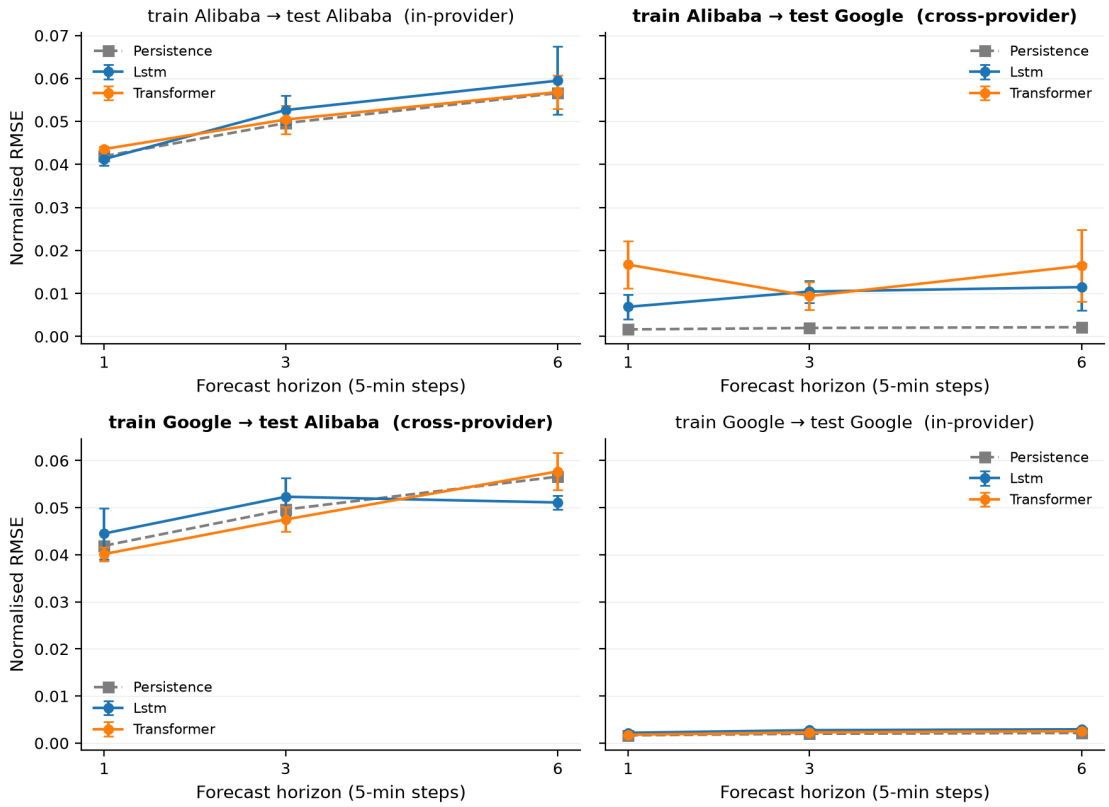


Figure 2. RMSE against forecast horizon for each of the four (source, target) combinations, with the persistence baseline for the corresponding target shown as a dashed grey reference. In-provider panels (upper-left, lower-right) show trained models at or near persistence. Cross-provider panels reveal an asymmetry: Alibaba→Google (upper-right) degrades sharply for both trained models, whereas Google→Alibaba (lower-left, bold title) sees the Transformer beat persistence at h=1 and h=3, and the LSTM beat persistence at h=6. This is the headline RQ2 finding — cross-provider transfer is asymmetric and model-dependent by horizon.

Transfer from Alibaba to Google failed at every horizon, for both architectures, without exception. All six Alibaba → Google cells produced RMSE several times larger than the corresponding Google persistence baseline. The LSTM's cross-provider RMSE rose with horizon, from 0.00690 at 5 minutes (4.2 times the persistence baseline of 0.00165) to 0.01045 at 15 minutes (5.3 times) and 0.01148 at 30 minutes (5.3 times).

The Transformer's degradation was more severe and less regular across horizons. At 5 minutes its mean RMSE was 0.01671, roughly ten times the persistence baseline and the single worst cell in the entire experiment. At 15 minutes its RMSE fell to 0.00941, around 4.7 times persistence; at 30 minutes it rose again to 0.01645, around 7.6 times persistence. This non-monotonic pattern coincides with a large seed spread at 30 minutes (standard deviation 0.00835 against a mean of

0.01645); seed variance across all cells is examined in §4.7. Regardless of the exact shape between horizons, the direction fails comprehensively: all six Alibaba → Google cells sit well above their persistence baseline.

Transfer from Google to Alibaba produced a markedly different picture. At 5 minutes, the Transformer achieved a mean RMSE of 0.04016 against a persistence baseline of 0.04193, an improvement of 0.0018, or 4.2%; the LSTM lost at this horizon, with a mean RMSE of 0.04452, a loss of 0.0026, or 6.2%. At 15 minutes, the Transformer again won, with a mean RMSE of 0.04750 against persistence's 0.04962, an improvement of 0.0021, or 4.3%; the LSTM again lost, with a mean RMSE of 0.05233, a loss of 0.0027, or 5.5%. At 30 minutes the pattern reversed: the LSTM won, with a mean RMSE of 0.05111 against persistence's 0.05659, an improvement of 0.0055, or 9.7%, the largest margin over persistence anywhere in the study; the Transformer lost at this horizon, with a mean RMSE of 0.05771, a loss of 0.0011, or 2.0%.

Three of the six Google → Alibaba cells beat persistence, and three did not: the LSTM lost at 5 and 15 minutes, and the Transformer lost at 30 minutes. Every horizon in this direction had exactly one winning architecture and one losing architecture, rather than either model dominating the direction outright.

Taken together, the two transfer directions could not be more different. Every one of the four cells in the whole experiment where a trained model beat persistence, across both in-provider and cross-provider evaluation, has Alibaba as the target: the one in-provider Alibaba win from §4.3, plus the three Google → Alibaba wins reported above. No cell, in-provider or cross-provider, beats persistence when the test target is Google; this holds across all twelve Google-target combinations spanning both directions and both architectures. Direction of transfer, not choice of architecture, is what separates a cell that beats persistence from one that does not: an architecture that wins comfortably when transferring into Alibaba can lose badly when transferring into Google, and vice versa within the same horizon. This asymmetry is the chapter's headline finding for RQ2 and is examined further in Chapter 5.

Within the Google → Alibaba direction specifically, no single architecture dominates even though the direction as a whole works. The Transformer wins the two shorter horizons, 5 and 15

minutes, by margins of 4.2% and 4.3%; the LSTM wins the longest horizon, 30 minutes, by 9.7%, the largest margin recorded anywhere in the study. A reader looking for a single "better" cross-provider architecture within this winning direction would need to specify the horizon before the question has an answer.

4.5 Computational Cost (RQ3)

This section addresses RQ3: what is the training-duration and inference-latency trade-off between the two architectures at matched parameter budget? Training duration is reported per training unit (one architecture, one source provider, one horizon, averaged across three seeds); inference latency is reported per evaluation, following the protocol in §3.10. Persistence, having no trainable parameters, required no training time, and its latency was already reported in §4.2 as the reference floor. Figure 4 plots accuracy against training cost for every model-horizon combination; Figure 3 plots the latency distributions directly.

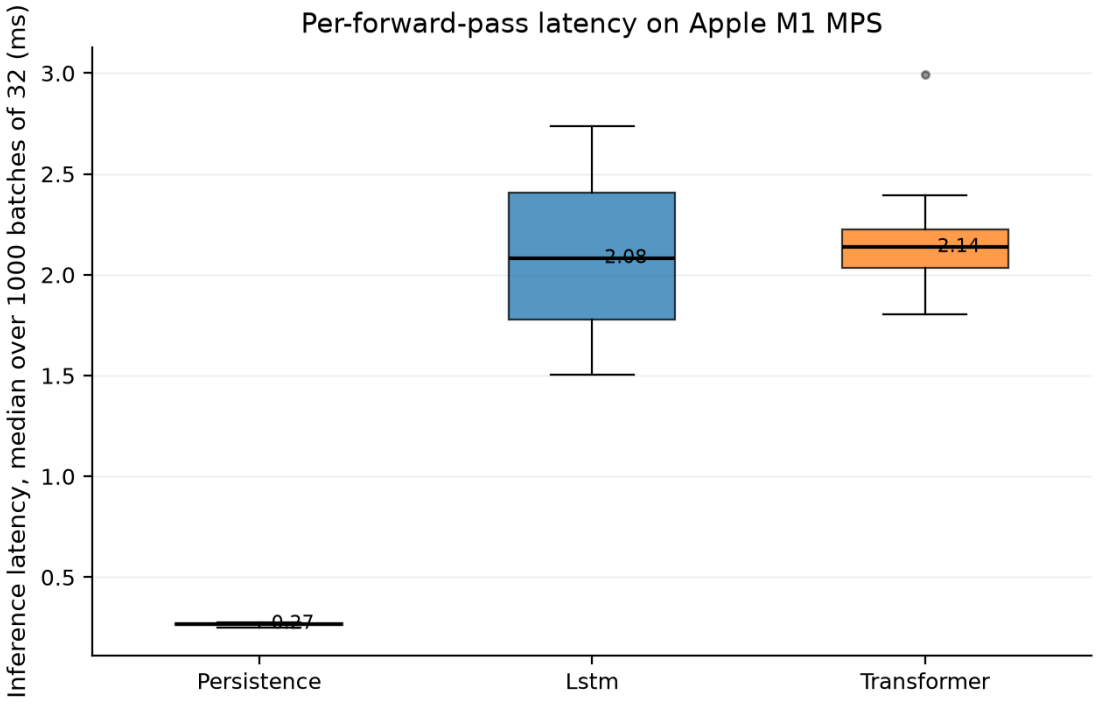


Figure 3. Distribution of median per-forward-pass latency (batch size 32) across all evaluation records for each model. Persistence sits around 0.27 ms; both trained models occupy the 1.7–2.5 ms band with heavy overlap, meaning there is no meaningful latency advantage of LSTM over Transformer at this parameter budget on this hardware. The ~8× gap to persistence is the cost of running a neural forward pass at all.

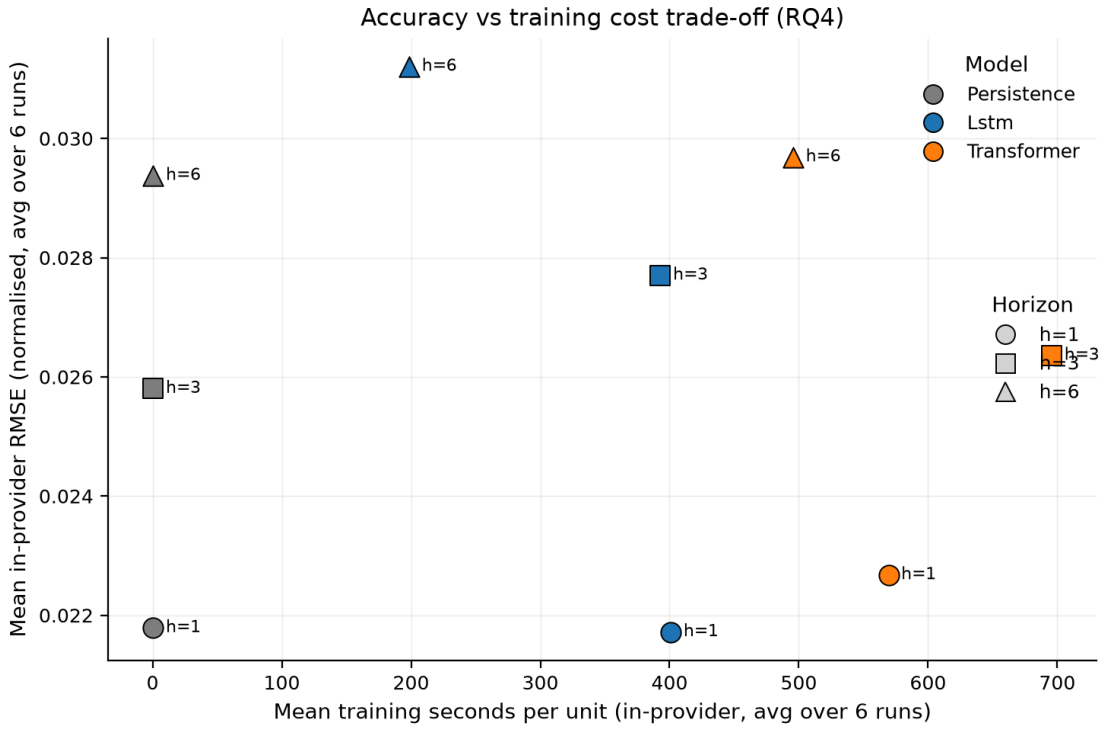


Figure 4. Accuracy-versus-cost trade-off (mean in-provider RMSE against mean per-unit training seconds), one point per (model, horizon) combination. Persistence is plotted near the origin with training cost effectively zero. LSTM occupies the low-cost side of the trade-off; Transformer takes 2–3× longer to train per unit for comparable or slightly better in-provider RMSE. There is no clear Pareto-dominant architecture at this budget: the answer to RQ4 depends on whether the operator values training cost or the modest in-provider RMSE gap at longer horizons.

Training duration varied by source provider more than by architecture alone. Alibaba-source LSTM units trained in 147 to 209 seconds per unit across the three horizons (206 seconds at 5 minutes, 209 at 15 minutes, 147 at 30 minutes); every Alibaba-source LSTM run terminated via early stopping within roughly 11 to 12 epochs. Alibaba-source Transformer units took longer at every horizon, 246 to 426 seconds per unit (426, 339, and 246 seconds at 5, 15, and 30 minutes respectively), between roughly 1.6 and 2.1 times the LSTM's training time at matched horizons.

Google-source training took substantially longer for both architectures. Google-source LSTM units trained in 250 to 596 seconds per unit (596, 577, and 250 seconds at 5, 15, and 30 minutes), with the longer end of that range reflecting runs that did not early-stop as quickly, discussed further in §4.6. Google-source Transformer units were the slowest in the experiment, training in 713 to 1,052 seconds per unit (713, 1,052, and 746 seconds at 5, 15, and 30 minutes), roughly five to seven times the duration of the fastest Alibaba-source LSTM unit.

Inference latency showed a very different pattern: architecture, not source provider, was the dominant factor, though even between architectures the difference was small. Across all 72 trained evaluations, LSTM median latency ranged from 1.755 to 2.509 milliseconds per batch of 32, and Transformer median latency ranged from 1.913 to 2.484 milliseconds. These two ranges overlap almost entirely; at no horizon or provider combination does one architecture separate cleanly from the other.

Both trained architectures were substantially slower than persistence, whose median latency across the six baseline evaluations was 0.249 to 0.275 milliseconds, reported in §4.2. The gap from persistence to either trained model, roughly 1.5 to 2.3 milliseconds, is the fixed cost of running a forward pass through a trained network at all; the gap between the two trained architectures, by contrast, is small enough that the LSTM and Transformer latency distributions cannot be reliably distinguished at this parameter budget on this hardware.

4.6 Training Dynamics and Loss Curves

Alibaba-source training runs converged quickly and then plateaued. The LSTM at 5 minutes, seed 42, reached its best validation loss of 0.002383 at epoch 1, the very first epoch of training; validation loss then rose noisily for the following 10 epochs without improving further, and early stopping fired at epoch 11. Figure 5 plots this run's training and validation curves alongside the corresponding Google run discussed below. This is representative of the broader pattern across Alibaba-source runs: an initial one-epoch descent to a low validation loss, followed by noisy horizontal or slightly upward drift that the patience mechanism, described in §3.9, is designed to catch. Because the best epoch typically arrived so early, most Alibaba-source training runs terminated well before the 50-epoch ceiling, consistent with the training-duration figures reported in §4.5.

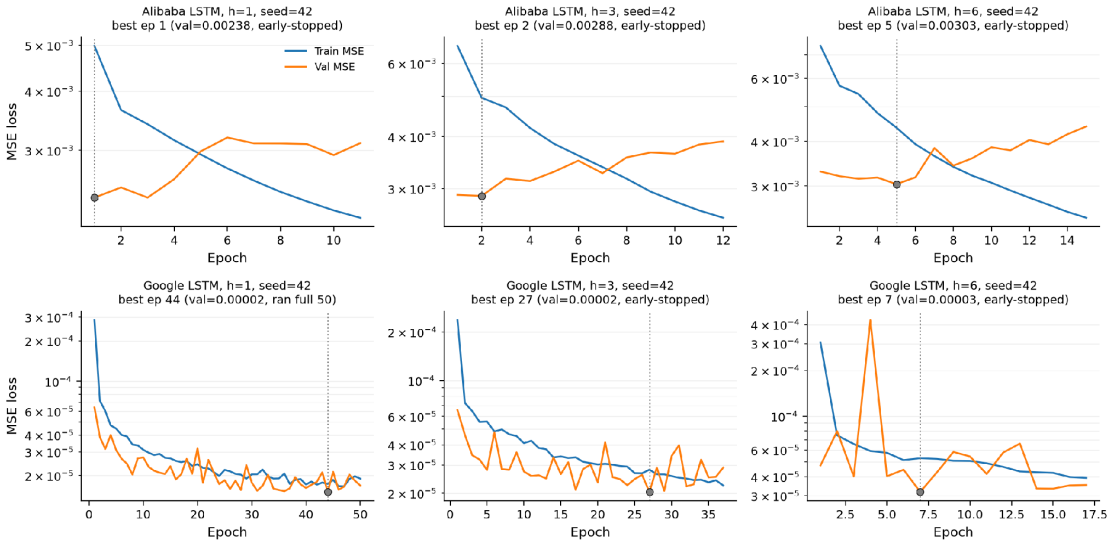


Figure 5. Training and validation MSE curves for LSTM seed=42 across each (source, horizon) combination, on a log-scale y-axis. Grey dotted line marks the best-val-loss epoch — the checkpoint the final evaluation uses. Alibaba-source runs typically reach best val within the first 5 epochs and are early-stopped by ≤ epoch 15. Google-source runs, in contrast, exhibit an extremely small val-loss range (~10^{-5} scale) and keep noisily improving over dozens of epochs — a symptom of the flat-signal problem in the Google cpu_usage_mean target.

Google-source training runs behaved very differently. The corresponding Google run, LSTM at 5 minutes, seed 42, trained for the full 50 epochs without early stopping: its best validation loss of 0.000015 was reached at epoch 44, leaving only six further epochs, too few for 10 consecutive non-improving epochs to accumulate before training reached its ceiling. This reflects the very small scale on which Google validation loss operates, a consequence of the compressed target distribution discussed in §4.2 and §3.6: with validation loss values on the order of 10^{-5}, small fluctuations that would be negligible on the Alibaba scale are large enough, relative to the loss itself, to repeatedly reset the patience counter and delay the point at which training is judged to have stopped improving.

Across the full set of training runs, this contrast in convergence speed was systematic rather than confined to the two seed-42 examples above. Alibaba-source models typically reached their best validation loss within the first five epochs of training; Google-source models typically reached it much later, between epoch 20 and epoch 45. This difference in best-epoch position is the direct mechanism behind the longer Google-source training durations reported in §4.5: a model that

keeps finding small improvements late into training runs for more epochs before its patience counter can accumulate 10 consecutive non-improving epochs.

4.7 Statistical Treatment: Variance Across Seeds

Reported means throughout this chapter are averages across three random seeds, with standard deviation as the associated spread. That spread was not uniform across the twenty-four trained cells. For the twelve in-provider cells (§4.3) and the six Google → Alibaba cross-provider cells (§4.4), standard deviation was mostly a modest fraction of the mean, typically in the low single digits to around 14% at most; the previous sections should be read with that level of run-to-run variation in mind, but the reported means are broadly stable.

The six Alibaba → Google cross-provider cells were markedly different. Every one of these six cells had a standard deviation between roughly 25% and 51% of its own mean, far larger than any in-provider or Google → Alibaba cell. This was not confined to a single horizon or architecture; it held across both models and all three horizons in that direction, and is the same seed instability already noted alongside the Transformer's non-monotonic degradation pattern in §4.4.

Two cells illustrate the practical consequence of this spread. The Alibaba → Google Transformer at 5 minutes had a mean RMSE of 0.01671 with a standard deviation of approximately 0.00553, a spread larger than several of the deltas-versus-persistence reported elsewhere in this study. The same architecture and direction at 30 minutes was more extreme still: a mean RMSE of 0.01645 against a standard deviation of 0.00835, roughly half the mean itself. In both cases, the reported mean remains the best available point estimate, but any claim resting specifically on the size of the Alibaba → Google degradation, rather than simply on its direction, should be read with this seed sensitivity in mind.

Separately from seed-to-seed variation in results, the pipeline's own reproducibility was verified directly. The full test suite comprised 71 unit tests, covering data handling, model construction, training, and evaluation; all 71 passed on the reference machine. A dedicated reproducibility test re-ran a two-epoch training with the same seed twice on the same hardware and compared the resulting validation-loss trajectories; the observed difference at every epoch was zero, within a

stated tolerance of 1×10^{-5}. Variation between seeds in this study reflects genuine sensitivity to initialisation and data order in the trained models themselves, not measurement noise in the pipeline that produced them.

Chapter 5 — Discussion

5.1 RQ1: On-Provider Accuracy

Chapter 4 established that, in-provider, only one of twelve trained cells beat the persistence baseline in normalised RMSE: the Alibaba-source LSTM at the 5-minute horizon, by 1.6%. Every other in-provider cell either lost to persistence or matched it within a fraction of a percentage point. That result answers RQ1 directly: at a matched parameter budget of roughly 30,000 trainable parameters, neither architecture reliably forecasts cloud CPU utilisation in-provider more accurately than repeating the last observed value, at any of the three horizons tested, with the single exception noted above.

The pattern is not confined to one provider. On Alibaba, the LSTM's single win at 5 minutes gave way to losses of 6.0% and 5.1% at 15 and 30 minutes; the Transformer never won, though its loss narrowed to 0.5% at the longest horizon. On Google, no cell won at all, and the Transformer's losses (7.9–17.1%) were consistently smaller in relative terms than the LSTM's (32.7–38.2%), even though the absolute deltas on both were small because the persistence baseline itself is already tiny on that target. Read together, the twelve cells describe a forecasting problem in which the naive baseline is difficult to displace regardless of architecture, echoing Makridakis et al.'s (2020) M4 finding that persistence-family baselines are hard for far more complex methods to beat consistently at short horizons. The finding also corroborates Zeng et al.'s (2023) argument that attention-based models do not reliably outperform much simpler alternatives on smooth, weakly seasonal series; this dissertation extends that argument from the nine general-purpose benchmarks Zeng et al. tested to cloud CPU utilisation specifically, and to an even simpler alternative than the linear model they used: persistence itself.

A reasonable objection is that the ~30,000-parameter budget is too small: published LSTM cloud-forecasting studies commonly report gains over baselines at hidden sizes of 128 or more,

several multiples of the 48 used here. That is a fair scope condition rather than a flaw in the comparison. The budget was fixed deliberately, and reduced from an initial specification, precisely so that any accuracy difference between the two architectures could be attributed to architecture rather than capacity (§3.8); the same constraint that enables that comparison also bounds what the in-provider result can claim. The result shows that at this budget, on these two traces, at these three horizons, persistence is not easily beaten, though it does not rule out an LSTM or Transformer configuration beating it at a larger budget; that replication is the natural next test, taken up in §6.3.

5.2 RQ2: Asymmetric Cross-Provider Transfer

Chapter 4 reported the study's headline result: cross-provider zero-shot transfer is directional, not symmetric. At every horizon, at least one model trained on Google and evaluated zero-shot on Alibaba beat the Alibaba persistence baseline: the Transformer at 5 and 15 minutes, the LSTM at 30. Every model trained on Alibaba and evaluated zero-shot on Google, by contrast, lost to the Google persistence baseline at all three horizons, by four to ten times the baseline RMSE. Together with the single in-provider win reported in §5.1, every cell in the whole study that beat persistence evaluates on Alibaba as the target; none evaluates on Google. RQ2 asked to what extent the two architectures maintain accuracy when transferred zero-shot across heterogeneous traces; the answer is that the direction of transfer, not the choice of architecture, decides whether zero-shot transfer adds value at all. This is consistent with the general transfer-learning literature's expectation that generalisation depends on how close source and target distributions are (Pan & Yang, 2010; Ben-David et al., 2010), though neither framework predicts in advance which of two specific traces would prove closer.

A plausible mechanism, offered here as an interpretation rather than a demonstrated cause, starts from how differently the two raw signals behave. Google's cpu_usage_mean is compressed near zero (mean 0.0123, standard deviation 0.0278, on the raw core-seconds-per-second scale reported in §3.6), reflecting a task-level trace that mixes long idle stretches with short bursts of activity. Alibaba's cpu_util_percent occupies a much wider, smoother mid-range (mean 39.7, standard deviation 14.3 percentage points, measured directly from the processed slice). A model

trained on Google's more heterogeneous signal may learn temporal structure, such as the shape of a transition between idle and active states, that still describes something real when applied to Alibaba's smoother range; a model trained on Alibaba's comparatively flat, mid-range signal has less structure of that kind to offer when applied to a target that spends most of its time near zero. This account is speculative: no ablation in this study isolates which property of the source distribution is doing the work, and it is offered as the most plausible reading of the asymmetry rather than an established cause.

A reasonable objection is that this is not evidence of asymmetric transfer so much as asymmetric baseline difficulty: persistence is already close to optimal on the flat Google target, so no source-trained model could add much beyond it, and the "failure" reflects the target's flatness rather than anything about Alibaba as a source. That objection is largely correct, and worth conceding directly, but it explains the finding rather than removing it. A model trained on Alibaba adds nothing when transferred to Google precisely because Google's target leaves almost no room for any source-trained model to improve on the last observed value; the mechanism and the direction-effect describe the same finding at different levels.

Practically, an operator choosing which provider to pretrain on, when the eventual deployment target is unknown or will vary, gains more from a source with richer within-series variance: here, that means Google generalises usefully to Alibaba, not the reverse. This is a narrow, direction-specific recommendation, not a general claim that any richer-variance source transfers well to any flatter target.

The finding is bounded by the study's design. Two providers cannot establish that transfer asymmetry is a general property of cross-provider cloud forecasting; the direction could reverse on a different pair of traces. Three seeds per cell is a thin basis for a causal claim, and the Alibaba→Google Transformer cell at 5 minutes has a seed standard deviation (0.00553) large enough to exceed several of the deltas reported elsewhere in this study, though the direction of failure across all six Alibaba→Google cells holds regardless of seed. The mechanistic account above is a hypothesis for future work, not something this design demonstrates.

5.3 RQ3: Computational Cost

RQ3 asked what training-duration and inference-latency trade-off separates the two architectures at matched parameter budget. On inference, the answer is that there is effectively no trade-off: LSTM median latency ranged 1.755–2.509 ms per batch of 32 across the 72 trained evaluations, Transformer median latency ranged 1.913–2.484 ms, and the two ranges overlap almost entirely. At this parameter budget, on this hardware, inference speed alone is not a meaningful basis for choosing between them.

Training duration is where the two architectures separate. Computed directly from the per-unit training seconds in §4.5, the Transformer took between roughly 1.2 and 3.0 times longer to train than the LSTM at matched source and horizon: as little as 1.2× on Google at 5 minutes, as much as 3.0× on Google at 30 minutes, with Alibaba-source runs sitting in a narrower 1.6–2.1× band throughout. Where deployment cost is dominated by training rather than serving, the LSTM is the cheaper choice at this budget; where only inference latency matters post-deployment, the two are interchangeable.

A natural objection is that this understates the Transformer's usual advantage: self-attention parallelises across the sequence dimension in a way recurrence cannot, an advantage well established for long sequences and large batches (Vaswani et al., 2017). The 24-step window and batch size of 32 used here, chosen to match the auto-scaling-relevant horizon rather than to stress-test throughput, are too short and too small to exercise that advantage. The finding is scoped to this input length and batch size on this hardware, not a general claim that Transformer inference is never faster than LSTM's.

The more consequential comparison is against persistence rather than between the two trained models. Persistence's forward pass costs roughly an eighth of either trained model's median latency. Persistence beat both trained architectures on eleven of the twelve in-provider cells (§5.1) and nine of the twelve cross-provider cells (§5.2); in most of the conditions tested here, the inference-latency cost of a trained model is paid for no corresponding accuracy gain.

5.4 RQ4: Synthesis — No Clear Winner

Across the twenty-four trained cells, no architecture wins consistently. The LSTM's two wins are the in-provider Alibaba result at 5 minutes (§5.1, an improvement of 0.0007, or 1.6%) and the Google→Alibaba cross-provider result at 30 minutes (§5.2, an improvement of 0.0055, or 9.7%, the largest margin in the study). The Transformer's two wins are both Google→Alibaba, at 5 and 15 minutes (improvements of 0.0018 and 0.0021, or 4.2% and 4.3%). Neither architecture beats persistence on a single Alibaba→Google cell, at any horizon. RQ4 asked which architecture presents the most viable predictive framework for proactive resource provisioning; the evidence does not support a horizon-invariant answer at this parameter budget. Which architecture wins depends on whether the setting is in-provider or cross-provider, which direction the transfer runs, and which horizon is forecast — three conditions that would each need specifying before "which is best" has a definite answer.

A deployment recommendation, then, has to be conditional rather than singular, and the two architectures do not divide along a simple winner-takes-all line. In-provider, the LSTM produced the study's only win, but the Transformer was the more consistent performer at every other in-provider horizon on both providers, losing to persistence by smaller margins throughout (§4.3); an operator who cares more about worst-case deviation from persistence than about occasionally beating it would reasonably prefer the Transformer in-provider, despite the LSTM holding the single win there. Cross-provider, the recommendation is squarely conditional on direction and horizon: transferring from Google to Alibaba, the Transformer wins at 5 and 15 minutes and the LSTM wins at 30; transferring from Alibaba to Google, neither architecture adds value at any horizon, and persistence is the only defensible choice.

A pragmatic reader might press for a single recommendation regardless: the LSTM's best margin (9.7%) exceeds either Transformer win, and it trains faster (§5.3), so why not recommend it outright? That preference is defensible operationally but not on the evidence as a whole: the Transformer's two short-horizon Google→Alibaba wins are as real as the LSTM's one long-horizon win, and recommending a single architecture across every condition tested would misstate what twenty of the twenty-four trained cells show: persistence winning regardless of architecture. The pattern is a specific instance of a broader result in learning theory, that no single algorithm dominates across every problem instance without prior assumptions about the

target (Wolpert, 1996). At this parameter budget, on these two traces, at these three horizons, the honest empirical answer to RQ4 is not "LSTM" or "Transformer" but persistence in most settings, and a horizon- and direction-specific choice between the two trained architectures in the few settings where either adds value.

5.5 Implications for Auto-Scaling Practice

The persistence-dominance finding running through §§5.1–5.4 is not a null result; it is a caution about when the cost of deep learning is worth paying. Across the twenty-four trained cells in this study, only four beat persistence, by modest margins (−0.0007 to −0.0055 in normalised RMSE), while producing them cost roughly 4.6 cumulative hours of training across 36 runs and an inference cost, once deployed, around eight times persistence's forward-pass latency (§4.2, §4.5). For an operator deciding whether to build a predictive scaler at these horizons on workloads resembling either trace, the evidence here argues for trying persistence, or a comparably cheap baseline, before committing to a trained model: at this parameter budget, most of the accuracy gain a trained model might promise does not materialise, and the gain that does appear is concentrated in one direction of one specific cross-provider transfer.

Where deep learning does help, the benefit is narrow rather than general. Zero-shot transfer from Google to Alibaba beat persistence at every horizon tested, by up to 9.7%; this is the one setting in the whole study where training a model, rather than deploying persistence directly, would have been the better operational choice. This result does not generalise into a broader endorsement of predictive scaling as commonly practised: it is one direction, between two specific traces, at three specific horizons, under one specific parameter budget.

A production system operated by a major cloud provider might reasonably object that this understates deep learning's value: Google's and AWS's own predictive autoscalers use trained models in production and are not being withdrawn on the strength of a null in-provider finding here. That objection is fair but addresses a different setting. Production predictive scalers typically use far larger models, richer multivariate inputs beyond a single CPU signal, and longer effective horizons than the ones tested here; this study's matched-budget, univariate design was

chosen specifically to make the architecture comparison fair, not to reproduce production conditions.

5.6 Limitations

Four scope conditions bound this study's claims, beyond the caveats already attached to each finding in §§5.1–5.5.

The parameter budget (~30,000 trainable parameters per architecture) was fixed to keep the two models comparable, not to maximise either one's accuracy (§3.8). Published cloud-forecasting LSTMs often use larger hidden sizes; whether either architecture behaves differently relative to persistence at a larger budget is untested here.

The cross-provider claims rest on two traces. Google and Alibaba differ enough to make the transfer test meaningful (§3.3), but two providers cannot establish that the Google-to-Alibaba/Alibaba-to-Google asymmetry is a general property of cross-provider forecasting rather than a property of this specific pair.

The forecasting target is univariate CPU only. Assessment 1 originally proposed CPU and memory jointly; this study narrowed to CPU to keep the matched-parameter comparison tractable (§3.3). Whether joint modelling changes the accuracy or transfer picture is not addressed here.

The three horizons tested cap at 30 minutes, chosen to match the boot-and-warm-up latency motivating predictive scaling (§2.1). Whether the persistence-dominance finding holds at horizons of hours, where a naive forecast has more room to drift, remains open.

Each boundary is a condition the design accepted in exchange for a controlled, matched-budget comparison, not a defect in it. Chapter 6 returns to three of them as directions for future work.

Chapter 6 — Conclusion and Future Work

6.1 Summary of Findings

Chapter 4 established that persistence was difficult to beat in-provider at these horizons: of the twelve in-provider cells tested, only one trained model beat the persistence baseline. The Alibaba-source LSTM at the 5-minute horizon achieved a normalised RMSE of 0.04125 against a persistence baseline of 0.04193, an improvement of 1.6%. Every other in-provider cell either lost to persistence or matched it within a fraction of a percentage point; on Google, persistence won all six in-provider cells outright. RQ1 is therefore answered directly: at a matched parameter budget of roughly 30,000 trainable parameters, neither the LSTM nor the Transformer reliably forecasts cloud CPU utilisation in-provider more accurately than the last observed value, at these three horizons.

Cross-provider zero-shot transfer proved directional rather than symmetric, the headline finding of Chapter 4. Every model trained on Alibaba and evaluated zero-shot on Google lost to the Google persistence baseline at all three horizons, by four to ten times the baseline RMSE. In the opposite direction, at every horizon at least one model trained on Google and evaluated zero-shot on Alibaba beat the Alibaba persistence baseline: the Transformer at 5 and 15 minutes (4.2% and 4.3% better than persistence), the LSTM at 30 minutes (9.7% better, the largest margin recorded in the study). RQ2's answer is that the direction of transfer, not the choice of architecture, determines whether zero-shot transfer adds value.

RQ3 asked what training-duration and inference-latency trade-off separates the two architectures at matched parameter budget. On inference, the answer is effectively none: LSTM and Transformer median latency, at 1.755–2.509 ms and 1.913–2.484 ms per batch respectively, overlap almost entirely across all 72 trained evaluations. On training duration, the two architectures separate clearly: at matched source and horizon, the Transformer took roughly 1.2 to 3 times longer to train than the LSTM, with Google-source Transformer runs the slowest in the experiment. Persistence remained roughly eight times cheaper than either trained model at inference, with no corresponding accuracy cost in most conditions tested.

6.2 Future Work

The clearest technical extension concerns the Google target's compressed distribution. An optional log1p transform ahead of Min-Max scaling was implemented but not activated for the reported experiment (§3.6). Re-running the Google-target evaluations with that transform active would test whether the persistence-dominance finding on Google reflects a genuine property of the underlying workload or an artefact of scaling a heavily compressed distribution, with a mean of 0.0123 and a standard deviation of 0.0278, directly into [0, 1].

A second extension concerns the Transformer architecture itself. The vanilla encoder-only Transformer tested here is the reference variant; specialised time-series Transformers such as Informer's sparse attention (Zhou et al., 2021) and PatchTST's patch-based tokenisation (Nie et al., 2023) have reported gains a standard encoder does not obtain elsewhere in the literature. Testing whether either variant closes the gap to persistence at this parameter budget would establish whether the persistence-dominance finding is specific to the vanilla architecture tested here or holds more broadly across the Transformer family.

A third extension concerns the forecasting target itself. Assessment 1 originally proposed forecasting CPU and memory jointly; this study narrowed to CPU alone to keep the matched-parameter comparison tractable across two already heterogeneous traces (§3.3). Joint modelling, particularly for the Transformer, could recover an advantage the univariate comparison does not show, if attention across CPU and memory channels captures cross-signal dependencies a single-channel model cannot represent.

A fourth extension concerns horizon length. The three horizons tested here cap at 30 minutes; whether the persistence-dominance finding holds at horizons of hours, where a naive last-value forecast has more room to drift from the true value, remains untested and would require a different windowing scheme than the one used here.

A fifth extension concerns provider coverage. The asymmetric-transfer finding rests on two providers; testing whether Google-source models generalise to a third, differently instrumented trace would establish whether that asymmetry reflects a Google-specific property or a more general relationship between source heterogeneity and transferability.

6.3 Closing Remarks

The three findings summarised above — persistence dominance at these horizons, asymmetric cross-provider transfer, and no meaningful inference-latency advantage of Transformer over LSTM at matched budget — constitute the dissertation's empirical contributions. The dissertation's methodological transparency is as much a contribution as its empirical findings. Every reported result is traceable to a specific, reproducible evaluation record, and every point at which the implemented study diverged from Assessment 1 is documented alongside the evidence that motivated the change, not presented as though it had been the original plan. A suite of 71 automated tests, a reproducibility check confirming bit-identical validation losses across repeated runs, and synchronisation-bracketed latency measurements make the computational-cost findings auditable rather than merely asserted. A comparison between architectures is only as trustworthy as the infrastructure used to produce it.

References

Arbat, S., Jayakumar, V. K., Lee, J., Wang, W., & Kim, I. K. (2022). Wasserstein adversarial transformer for cloud workload prediction. *Proceedings of the AAAI Conference on Artificial Intelligence, 36*(11), 12433–12439. https://doi.org/10.1609/aaai.v36i11.21509

Aslanpour, M. S., Gill, S. S., & Toosi, A. N. (2020). Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research. *Internet of Things, 12*, 100273. https://doi.org/10.1016/j.iot.2020.100273

Alibaba Cluster Trace Program. (n.d.). *Cluster trace v2018.* GitHub. https://github.com/alibaba/clusterdata/tree/master/cluster-trace-v2018

AWS. (n.d.). *Predictive scaling for Amazon EC2 Auto Scaling.* Amazon Web Services https://docs.aws.amazon.com/autoscaling/ec2/userguide/ec2-auto-scaling-predictive-scaling.html

Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., & Vaughan, J. W. (2010). A theory of learning from different domains. *Machine Learning, 79*(1–2), 151–175. https://doi.org/10.1007/s10994-009-5152-4

Box, G. E. P., & Jenkins, G. M. (1970). *Time series analysis: Forecasting and control.* Holden-Day.

Cheng, Y., Chai, Z., & Anwar, A. (2018). Characterizing co-located datacenter workloads: An Alibaba case study. In *Proceedings of the 9th Asia-Pacific Workshop on Systems* (Article 12). Association for Computing Machinery. https://doi.org/10.1145/3265723.3265742

Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural Computation, 12*(10), 2451–2471. https://doi.org/10.1162/089976600300015015

Google. (n.d.). *Cluster data.* GitHub. https://github.com/google/cluster-data

Google Cloud. (n.d.). *Scaling based on predictions.* Compute Engine documentation https://cloud.google.com/compute/docs/autoscaler/predictive-autoscaling

Guo, J., Chang, Z., Wang, S., Ding, H., Feng, Y., Mao, L., & Bao, Y. (2019). Who limits the resource efficiency of my datacenter: An analysis of Alibaba datacenter traces. In *Proceedings of the International Symposium on Quality of Service (IWQoS '19)* (Article 39). Association for Computing Machinery. https://doi.org/10.1145/3326285.3329074

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hyndman, R. J., & Athanasopoulos, G. (2021). *Forecasting: Principles and practice* (3rd ed.). OTexts. https://otexts.com/fpp3/

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations (ICLR 2015).* https://arxiv.org/abs/1412.6980

Kumar, J., & Singh, A. K. (2018). Workload prediction in cloud using artificial neural network and adaptive differential evolution. *Future Generation Computer Systems, 81*, 41–52. https://doi.org/10.1016/j.future.2017.10.047

Lorido-Botran, T., Miguel-Alonso, J., & Lozano, J. A. (2014). A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing, 12*(4), 559–592. https://doi.org/10.1007/s10723-014-9314-7

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2020). The M4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting, 36*(1), 54–74. https://doi.org/10.1016/j.ijforecast.2019.04.014

Nie, Y., Nguyen, N. H., Sinthong, P., & Kalagnanam, J. (2023). A time series is worth 64 words: Long-term forecasting with transformers. In *The Eleventh International Conference on Learning Representations (ICLR 2023)*. https://openreview.net/forum?id=Jbdc0vTOcol

Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering, 22*(10), 1345–1359. https://doi.org/10.1109/TKDE.2009.191

Prachitmutita, I., Aittinonmongkol, W., Pojjanasuksakul, N., Supattatham, M., & Padungweang, P. (2018). Auto-scaling microservices on IaaS under SLA with cost-effective framework. In *2018 Tenth International Conference on Advanced Computational Intelligence (ICACI)* (pp. 583–588). IEEE. https://doi.org/10.1109/ICACI.2018.8377525

Prechelt, L. (1998). Early stopping — But when? In G. B. Orr & K.-R. Müller (Eds.), *Neural networks: Tricks of the trade* (Vol. 1524, pp. 55–69). Springer. https://doi.org/10.1007/3-540-49430-8_3

Reiss, C., Tumanov, A., Ganger, G. R., Katz, R. H., & Kozuch, M. A. (2012). Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing* (Article 7). Association for Computing Machinery. https://doi.org/10.1145/2391229.2391236

Reiss, C., Wilkes, J., & Hellerstein, J. L. (2011). *Google cluster-usage traces: Format + schema* (Technical report). Google Inc. https://github.com/google/cluster-data/blob/master/ClusterData2011_2.md

Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2020). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting, 36*(3), 1181–1191. https://doi.org/10.1016/j.ijforecast.2019.07.001

Sherstinsky, A. (2020). Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena, 404*, 132306. https://doi.org/10.1016/j.physd.2019.132306

Song, B., Yu, Y., Zhou, Y., Wang, Z., & Du, S. (2018). Host load prediction with long short-term memory in cloud computing. *The Journal of Supercomputing, 74*(12), 6554–6568. https://doi.org/10.1007/s11227-017-2044-4

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)* (pp. 5998–6008). Curran Associates. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

Wen, Q., Zhou, T., Zhang, C., Chen, W., Ma, Z., Yan, J., & Sun, L. (2023). Transformers in time series: A survey. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI-23)* (pp. 6778–6786). https://doi.org/10.24963/ijcai.2023/759

Wolpert, D. H. (1996). The lack of a priori distinctions between learning algorithms. *Neural Computation, 8*(7), 1341–1390. https://doi.org/10.1162/neco.1996.8.7.1341

Zeng, A., Chen, M., Zhang, L., & Xu, Q. (2023). Are transformers effective for time series forecasting? *Proceedings of the AAAI Conference on Artificial Intelligence, 37*(9), 11121–11128. https://doi.org/10.1609/aaai.v37i9.26317

Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., & Zhang, W. (2021). Informer: Beyond efficient transformer for long sequence time-series forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence, 35*(12), 11106–11115. https://doi.org/10.1609/aaai.v35i12.17325