Deep Learning Methods for Transient Performance Analysis and Optimization of Non-Markovian Nonstationary Queueing Systems

Spyros Garyfallos

December 2025

Universitat Politècnica de Catalunya (UPC)
Departament d'Arquitectura de Computadors (DAC)

Advisor:
Prof. Albert Cabellos-Aparicio

Co-Advisor:
Prof. Yunan Liu

The Service Economy Challenge

Global Scale

- $70-80T annual activity (67% GDP) [5, 9]
- 1.7B workers (50% employment) [3, 9]
- $3.1T lost annually in poor service [6]

The Challenge

- Match time-varying capacity to unpredictable demand
- Maintain service quality, control costs
- 1% improvement = $775M over 3 years for $1B company [8]

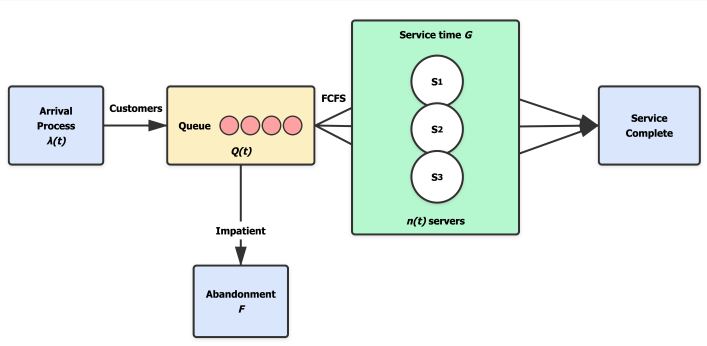
Critical Applications

Healthcare ED
Staff physicians over 12-hr shift
Patient wait times, outcomes

Contact Centers
50-500 agents, 50K daily calls
SLA: 80% within 20 seconds

Cloud Platforms
Thousands of service requests
99th percentile latencies

Queueing Systems: A trade-off between cost and customer experience



Components

- Arrivals: $\lambda(t)$ customers/time
- Queue: waiting customers
- Servers: $n(t)$ capacity
- Service & abandonment times

Kendall Notation

$$A/B/c/K/N/D$$

Components:

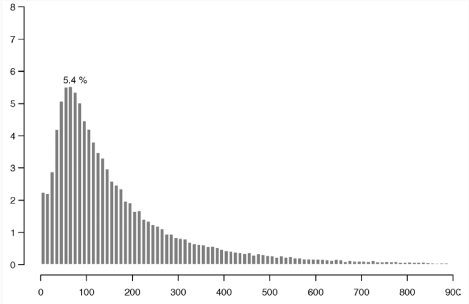
- A: Arrival process
- B: Service time distribution
- c: Number of servers
- K: System capacity
- N: Population size
- D: Service discipline

This thesis: $G_t/GI/n_t + GI$

- G_t: General time-varying arrivals
- GI: General service times
- n_t: Time-varying capacity
- $+GI$: General abandonment

Why Non-Exponential Distributions Matter

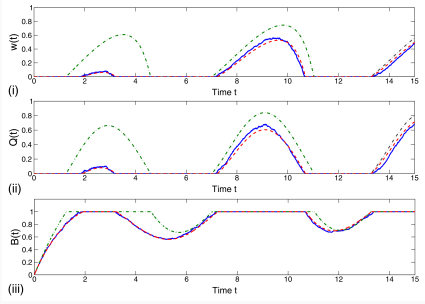
Empirical Evidence



Source: [1]

Service times from real call center

Impact on Performance



Source: [4]

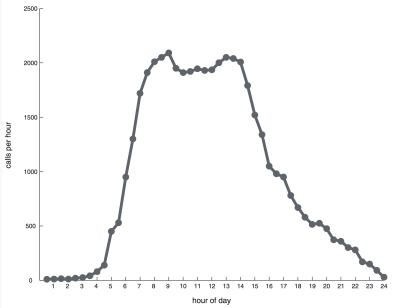
Same means, different variability

Dramatically different performance!

Variability matters: Cannot assume exponential distributions

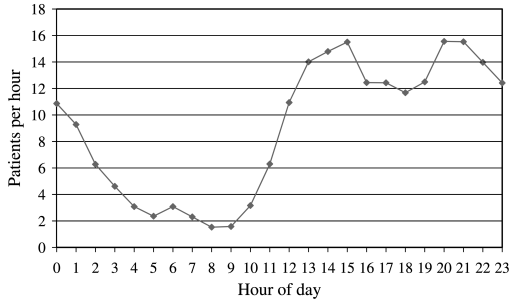
Time-Varying Demand is Everywhere

Call Center Arrivals



Source: [2]

Emergency Department



Source: [10]

The Fundamental Practical Problem

- What is the future performance of the system, e.g., waiting time, queue length, $\mathbb{P}(W(t) > \tau)$
- How many servers at each time to meet service-level targets?

Current Challenges

- Analytical: Simple cases only
- Simulation: Hours per scenario
- Heuristics: Inaccurate

The Three Traditional Approaches

Analytical Methods

- Heavy-traffic theory
- Fluid approximations

Pros:

- Fast (seconds)
- Theoretical insights

Cons:

- Large-scale only
- Complex for practitioners

Simulation

- Monte Carlo
- 10K+ replications

Pros:

- Accurate
- General

Cons:

- Slow (hours)
- Infeasible for optimization

Heuristics

- Square-root staffing
- Rules of thumb

Pros:

- Simple
- Fast

Cons:

- Limited accuracy
- No guarantees

Our Goal


Accuracy
Simulation
Our Goal
The Research Opportunity
Analytical
Heuristics
Speed

The Computational Bottleneck

Capacity Planning Workflow

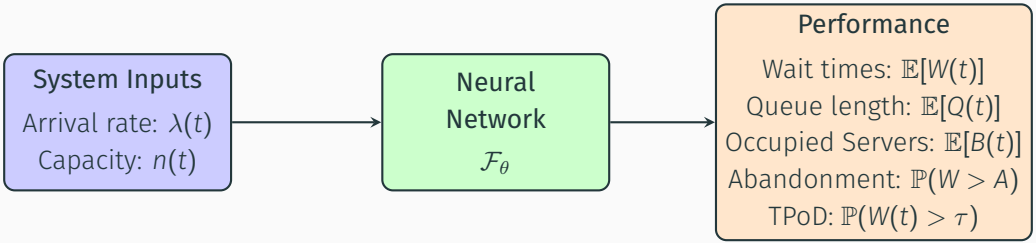
1. Forecast demand: $\lambda(t)$
2. Find staffing: $n(t)$ to meet constraints
3. Optimize: Minimize cost

The Problem

- Each candidate schedule requires full performance evaluation
- Simulation: 2-3 hours per scenario
- Need 100s-1000s of evaluations
- **Total: Weeks to months**

Can neural networks solve this?

Research Hypothesis: Neural Networks as Performance Oracles



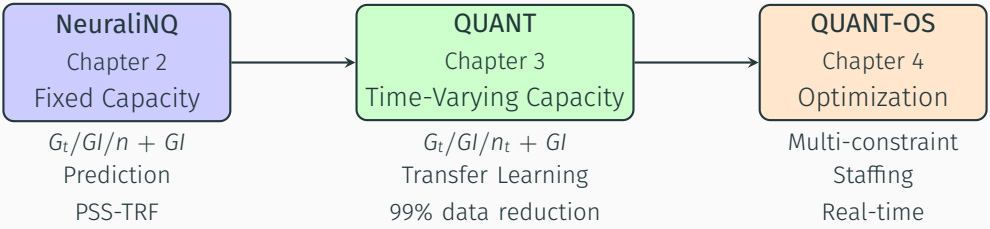
Why Neural Networks?

- Universal approximation: Learn complex mappings
- Fast inference: Milliseconds per prediction
- Differentiable: Enables sensitivity analysis and gradient-based optimization

The Trade-off:

- Expensive training (days, offline) → Cheap prediction (ms, online)

Our Research Journey: Three Gradual Contributions



Progression:

Ch 2: Can we predict?

Ch 3: Can we scale?

Ch 4: Can we optimize?

Roadmap

1. **NeuraliNQ (Ch 2):** Fixed capacity prediction
   - PSS-TRF decomposition
   - 10,000× speedup vs simulation
2. **QUANT (Ch 3):** Time-varying capacity
   - Transfer learning: Markovian → Non-Markovian
   - 99% reduction in training data
3. **QUANT-OS (Ch 4):** Optimization
   - Multi-constraint capacity planning
   - Seconds to optimize

NeuraliNQ: Neural Network for Inference in Nonstationary Queues

Problem: $G_t/GI/n + GI$

- Time-varying arrivals, general service times
- Fixed capacity, customer abandonment
- Predict transient performance

Challenges

- Non-Markovian dynamics
- Nonstationary arrivals
- Error accumulation

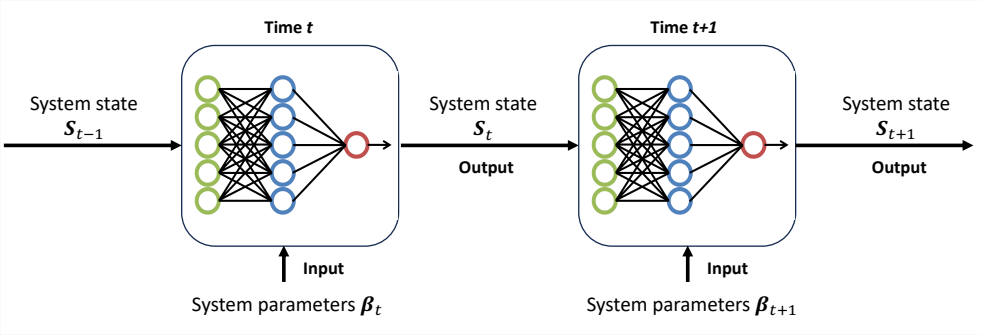
Goals

- Simulation-level accuracy
- Millisecond-level speed
- Interpretable predictions

Benchmarks

- *Ground truth*: Monte Carlo Simulation
- *Comparison*: Fluid approximation [4]

A Recurrent NN (RNN) with carefully constructed state



Key Insight: Parsimonious State Representation

Conventional Wisdom
Non-Markovian systems require tracking age distributions of all customers
$\rightarrow$ Infinite-dimensional state space

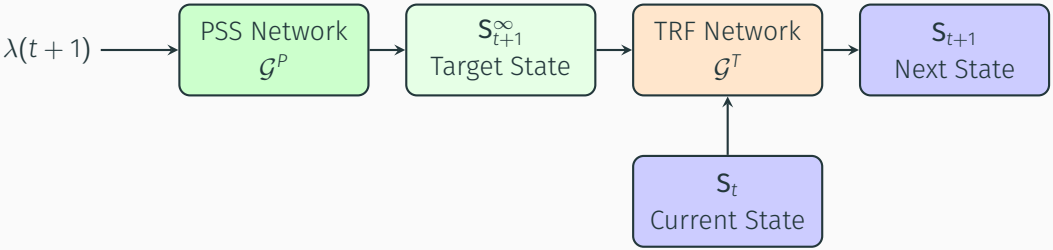
Our Discovery
For *expected* performance prediction, a 2D snapshot state suffices:

$$S_t = (\mathbb{E}[W(t)], \mathbb{E}[B(t)])$$

- $\mathbb{E}[W(t)]$: Mean waiting time (queue side)
- $\mathbb{E}[B(t)]$: Mean busy servers (server side)

Why does this work?
- We predict *expectations*, not sample paths
- Current state aggregates implicitly capture needed information
- Networks learn relationships between means from training data

Architecture Innovation: PSS-TRF Decomposition



PSS: Pointwise Steady State

- Where would the system converge if $\lambda(t+1)$ stayed constant?
- Provides target/reference

TRF: Transient Response

- How does system move from $\mathbf{S}_t$ toward $\mathbf{S}^{\infty}_{t+1}$?
- Captures gradual adjustment

Why PSS-TRF Decomposition?

Interpretability
- PSS: Long-run equilibrium
- TRF: Transient dynamics
- Aligns with queueing theory

Training Efficiency
- PSS: Smooth, monotonic
- Separate objectives
- Easier optimization

Error Correction
- PSS provides target
- Prevents accumulation
- Self-correcting

Stepwise Prediction as Function Approximation

The core problem: Learn the mapping

$$\mathsf{S}_{t+1} = \mathcal{F}(\mathsf{S}_t, \beta_{t+1}), \quad t = 0, 1, 2, \ldots$$

where:

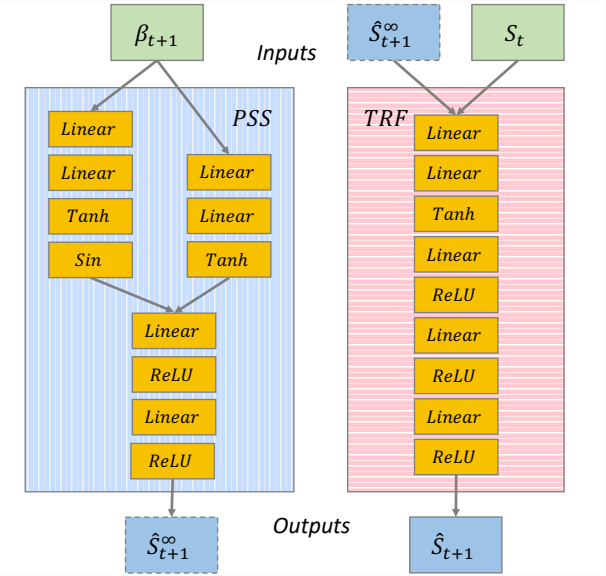
- $\mathsf{S}_t = (\mathbb{E}[W(t)], \mathbb{E}[B(t)])$ - System state (mean wait time, mean busy servers)
- $\beta_{t+1} = \lambda(t+1)$ - External input (arrival rate)
- $\mathcal{F}$ - Unknown transition function (learned via neural networks)

Two-Stage Decomposition:

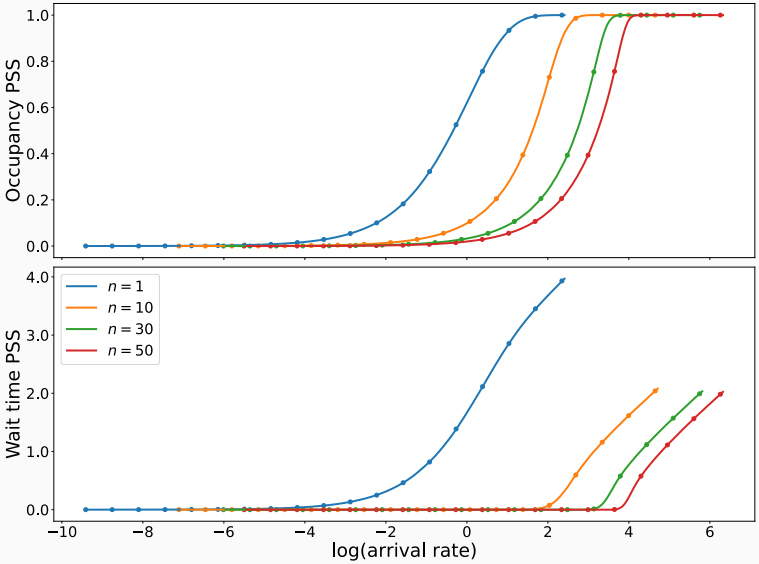
$$\widehat{\mathsf{S}}^{\infty}_{t+1} \leftarrow \mathcal{G}^{P}(\beta_{t+1}) \quad \text{(PSS Network)}$$
$$\widehat{\mathsf{S}}_{t+1} \leftarrow \mathcal{G}^{T}(\mathsf{S}_t, \widehat{\mathsf{S}}^{\infty}_{t+1}) \quad \text{(TRF Network)}$$

Key insight: Parsimonious 2D state suffices for expected performance prediction in non-Markovian systems

NeuraliNQ: The NNs architecture



The fitted PSS curves for server occupancy and waiting time



PSS Network: Dual-Branch Architecture Design

Challenge: Capture both smooth and non-smooth characteristics of steady-state curves

Architecture Rationale

Branch 1 (Smooth):

- Tanh activations
- Captures monotonic trends
- Models gradual transitions

Branch 2 (Non-smooth):

- Sinusoidal activations [7]
- Captures critical loading point
- Models rapid transitions

Fusion:

- Concatenation + ReLU layers
- Combines complementary features

Mathematical Form

Branch 1:

$$\mathbf{h}_1 = \text{Tanh}(\mathbf{W}_1\lambda + \mathbf{b}_1)$$

Branch 2:

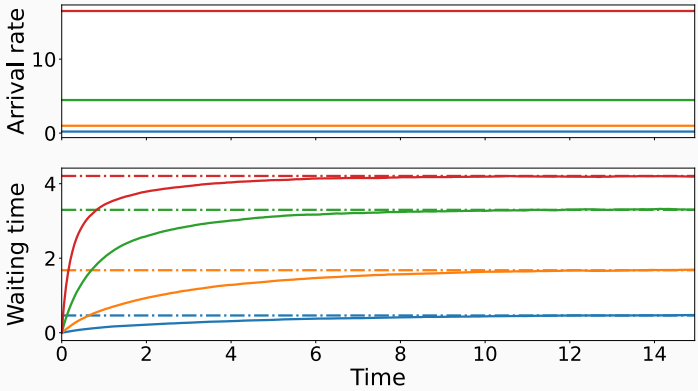
$$\mathbf{h}_2 = \sin(\mathbf{W}_2\lambda + \mathbf{b}_2)$$

Fusion:

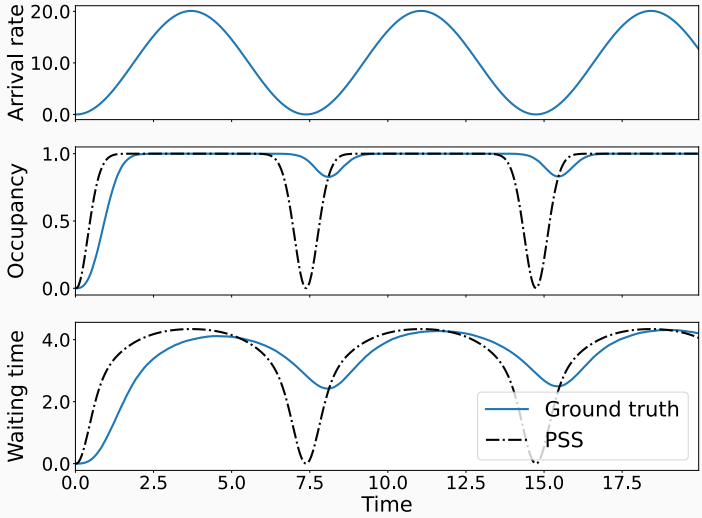
$$\mathbf{S}^{\infty} = \text{ReLU}(\mathbf{W}_3[\mathbf{h}_1; \mathbf{h}_2] + \mathbf{b}_3)$$

Design principle: Different activation functions capture different curve properties—combining them handles full complexity

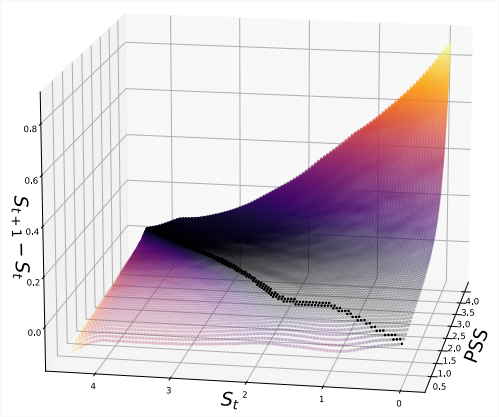
Extracting the PSS training data from simulation



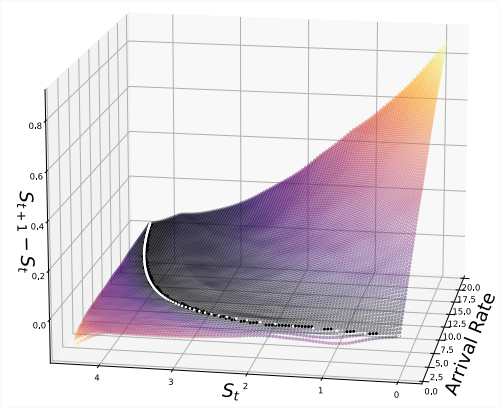
NeuraliNQ: PSS compared to transient performance



A 3D visualization of the TRF NN

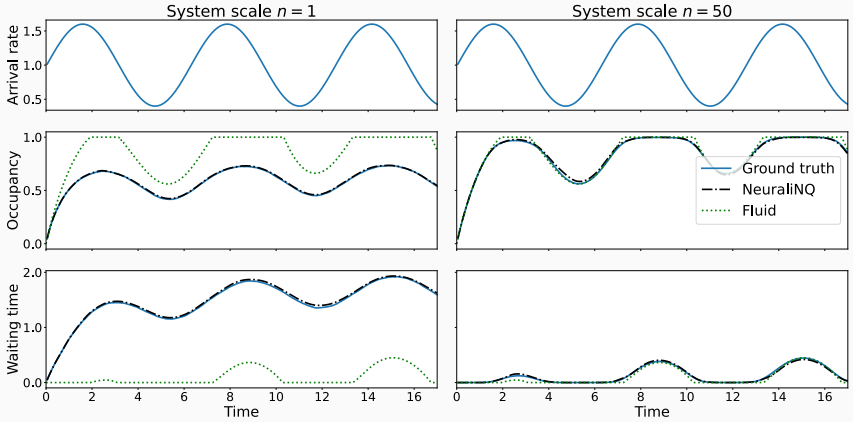


(a) The trained TRF surface.

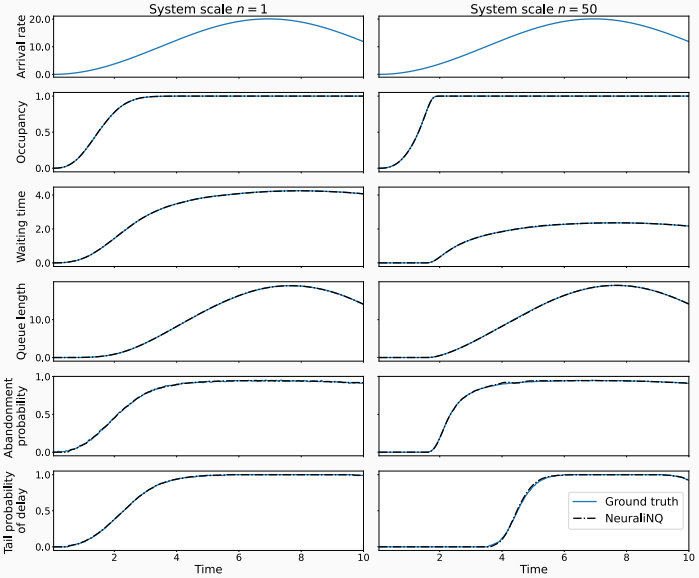


(b) The trained prediction curve.

NeuraliNQ: Prediction Quality



NeuraliNQ: Multiple Performance Metrics



NeuraliNQ: Key Results

Accuracy (*time-averaged mean squared error* - TAMSE)

- TAMSE: 10^{-4} to 10^{-3}
- 3-4 orders of magnitude better than fluid (small scale)
- Multiple metrics: wait time, queue, abandonment, tails

Speed

- Prediction: 200-350ms
- Simulation: 2-3 hours
- **Speedup: $>$ 10,000×**

Robustness

- Scales: $n = 1$ to $n = 50$
- Distributions: Erlang to Hyperexponential
- Patterns: Slow to rapid variation

NeuraliNQ: Takeaways

What We Demonstrated

1. Neural networks can predict non-Markovian queues accurately
2. Parsimonious 2D states suffice
3. PSS-TRF provides interpretability & stability
4. 4 orders of magnitude speedup

Limitation

Fixed capacity only (n constant)
PSS requires abandonment

Next: Time-varying capacity?

QUANT: The Time-Varying Capacity Challenge

New Problem: $G_t/GI/n_t + GI$
Both $\lambda(t)$ and $n(t)$ vary → Realistic operations

The Dimensionality Explosion
Fixed: $[\lambda_{min}, \lambda_{max}]$ (1D)
Time-varying: $[\lambda_{min}, \lambda_{max}] \times [n_{min}, n_{max}]$ (2D)

- Need 10K trajectories for coverage
- Each: 2-3 hours simulation
- **Total: 2-3 CPU-years!**

Challenge: Train without this computational burden

Two Architectural Changes in QUANT

1. Eliminate PSS Dependency

Problem with PSS:

- Need steady states at every capacity level
- $n \in [1, 150]$ requires 150x simulations
- Cost scales with capacity range

QUANT solution:

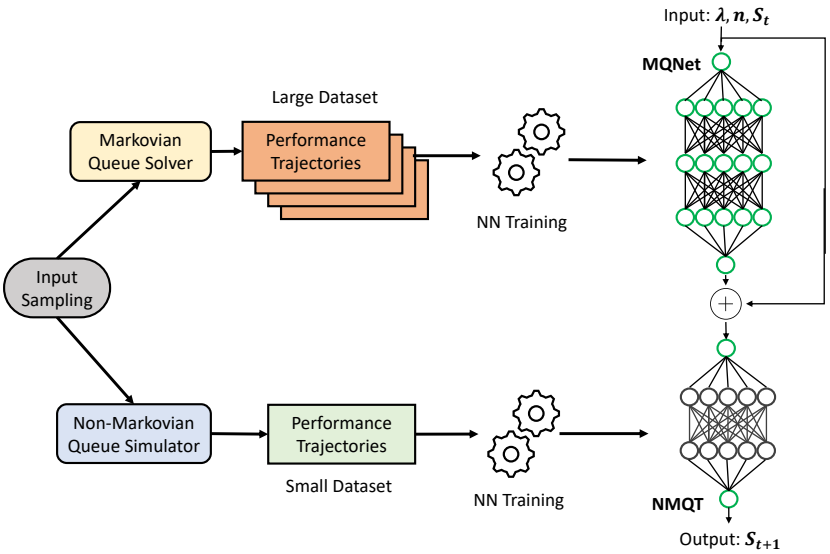
- Direct prediction of dynamics
- Deeper NNs
- More training data

2. Transfer Learning

QUANT approach:

- Pre-train on 10K Markovian trajectories
- Fine-tune on 100 non-Markovian
- **99% reduction in simulation!**
- Accuracy loss < 5%

The Transfer Learning Strategy



Why Transfer Learning Works: A Queueing Theory Perspective

Four Features Characterizing Any Queueing System:

1. System Structure
- Topology: single, parallel, networked
- Capacity: number of servers $n(t)$

2. Operational Rules
- Service discipline (FCFS)
- Customer abandonment behavior

3. Random Elements: Means
- Arrival rate $\lambda(t)$
- Mean service time $1/\mu$
- Mean patience $1/\theta$

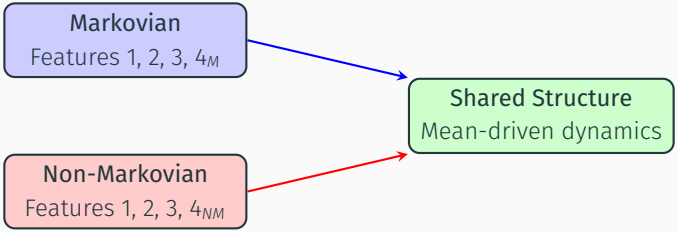
4. Distributions Beyond Means
- Service time distribution
- Abandonment time distribution

Key Insight

Features 1-3 are shared between Markovian and non-Markovian systems.

Only Feature 4 differs—and this is what we transfer!

The Transfer Learning Rationale



MQNet Learns (Pre-training):

- Queue buildup/drain dynamics
- $\lambda(t)$, $n(t) \rightarrow$ congestion mapping
- Distribution-independent behavior

NMQT Learns (Fine-tuning):

- Distributional corrections only
- How variability affects performance
- The *difference*, not everything

Result: 99% less simulation data needed

MQSolver: Fast Markovian Oracle

Purpose
Generate 10K Markovian trajectories cheaply for pre-training

Approach:

- Exploit birth-death process structure
- Uniformization + matrix methods
- Exact, deterministic solutions

Performance:

- 60 seconds per trajectory
- vs 900 seconds for simulation
- **15× speedup**

Enables cheap generation of abundant Markovian training data

MQSolver: Mathematical Formulation

Discrete-time birth-death process with time step $h = T/M$:

Transition Probabilities

For state $i \in \{0, 1, \ldots, N\}$:

$$P_{i,j}(t_k) = \begin{cases} \lambda(t_k)h & j = i+1,\ i < N \\ \delta(t_k, i)h & j = i-1,\ i > 0 \\ 1 - a_i(t_k)h & j = i \\ 0 & \text{otherwise} \end{cases}$$

where

$$\delta(t_k, i) = \min(i, n(t_k))\mu + (i - n(t_k))^+\theta$$
$$a_i(t_k) = \lambda(t_k)\mathbf{1}_{i<N} + \delta(t_k, i)$$

Transient Distribution

$$\boldsymbol{\pi}(t_k) = \boldsymbol{\pi}(0)\prod_{j=0}^{k-1}\mathsf{P}(t_j)$$

Performance Metrics

From $\boldsymbol{\pi}(t_k) = [\pi_0(t_k), \ldots, \pi_N(t_k)]$:

$$\mathbb{E}[X(t_k)] = \sum_{i=0}^{N} i \cdot \pi_i(t_k)$$

$$\mathbb{E}[Q(t_k)] = \sum_{i=0}^{N} (i - n(t_k))^+\pi_i(t_k)$$

$$\mathbb{E}[B(t_k)] = \sum_{i=0}^{N} \min(i, n(t_k)) \cdot \pi_i(t_k)$$

$X(t)$: customers in system
$Q(t)$: customers in queue
$B(t)$: busy servers
N: truncation level

MQSolver: Waiting Time via Phase-Type Distribution

Key idea: Model waiting time $W(t)$ as absorption time in a pure death process

Phase-Type Construction

For a customer finding q others in queue:

- State space: $\{0, 1, \ldots, q+1\}$
- State 0: absorbing (service begins)
- State j: j customers ahead

Transition rates from state j:

$$\gamma_j(t) = n(t)\mu + (j-1)\theta, \quad j \geq 1$$

Deaths occur via:

- Service completion (rate $n(t)\mu$)
- Abandonment of customers ahead (rate $(j-1)\theta$)

Tail Probability of Delay

$$\mathbb{P}(W(t) > \tau) = \sum_{x=n(t)}^{N} \mathbb{P}(W(t) > \tau \mid X(t) = x) \cdot \pi_x(t)$$

where the conditional probability is:

$$\mathbb{P}(W(t) > \tau \mid X(t) = x) = \mathbf{e}_q^\top \prod_{k=0}^{M_\tau - 1} \mathsf{P}(t_k, q) \cdot \mathbf{1}$$

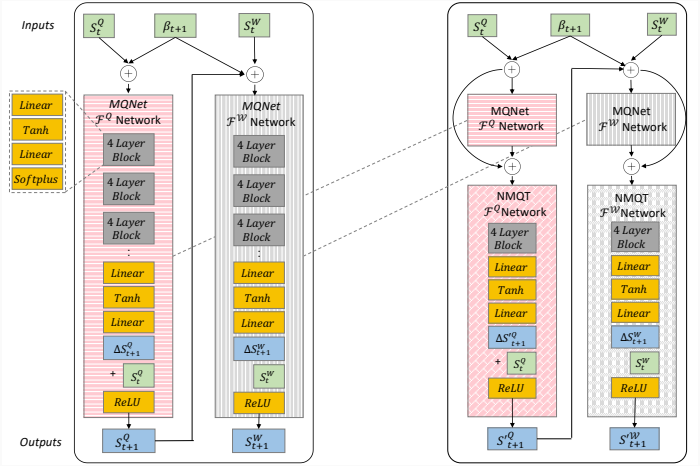
with $q = x - n(t)$ and $M_\tau = \tau/h$

Mean Waiting Time

$$\mathbb{E}[W(t)] = \sum_{x=n(t)}^{N} \mathbb{E}[W(t) \mid X(t) = x] \cdot \pi_x(t)$$

Conditional means computed via backward recursion

QUANT Architecture: Tandem Networks



Two specialized networks process different aspects of queueing dynamics

QUANT Architecture: Mathematical Design

Tandem Network Structure: Separate processing for count vs. time metrics

Count Network ($\mathcal{F}^Q$)

Predicts changes in discrete metrics:

$$\mathbf{S}_{t+1}^Q = \mathbf{S}_t^Q + \mathcal{F}^Q(\mathbf{S}_t, \beta_{t+1})$$

where $\mathbf{S}_t^Q = (\mathbb{E}[Q(t)], \mathbb{E}[B(t)])$

Handles: Sharp, discrete transitions in queue length and server occupancy

Time Network ($\mathcal{F}^W$)

Maps queue state to waiting time:

$$\mathbf{S}_{t+1}^W = \mathcal{F}^W(\mathbf{S}_{t+1}^Q, \mathbf{S}_t^W, \beta_{t+1})$$

where $\mathbf{S}_t^W = (\mathbb{E}[W(t)])$

Handles: Smooth, continuous evolution of waiting times

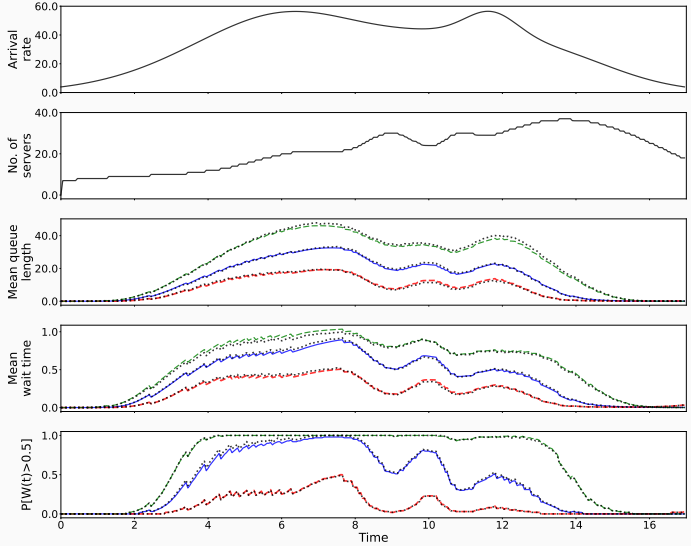
4-Layer Repeating Blocks

- Stabilizes gradients through depth
- Enables deeper networks (16 layers)
- Better learning capacity

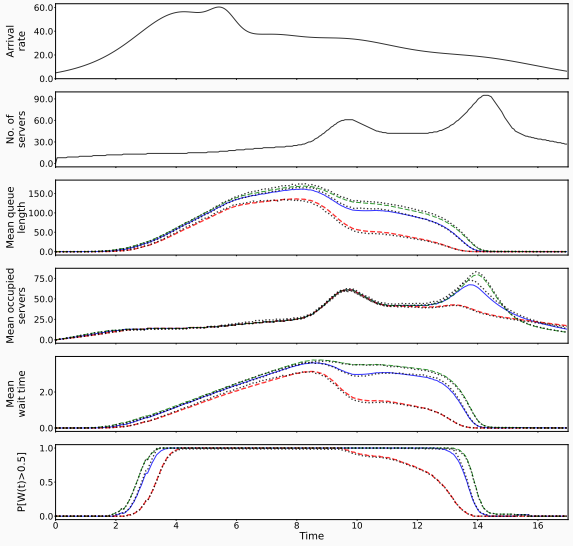
Dual Activation Functions

- Tanh: Zero-centered, smooth gradients
- Softplus: Continuous, differentiable ReLU
- Combined: Stability across many recurrent steps

QUANT: Prediction Quality Example



QUANT: No abandonment



Transfer Learning Data Efficiency

The Core Question
How much non-Markovian simulation data do we really need?

Baseline (No Transfer)

- Train from scratch
- 10,000 trajectories
- 20K-30K CPU-hours
- TAMSE: 0.0703

QUANT (Transfer Learning)

- Pre-train on Markovian
- 100 trajectories (1%)
- 200-300 CPU-hours
- TAMSE: 0.0710 (within 5%)

Result

99% reduction in simulation requirements
with $<$**5% accuracy degradation**

QUANT: Key Results

Accuracy

- TAMSE < 0.1 across test scenarios
- Consistent: Markovian, Erlang, Hyperexponential
- Robust: 10-100 servers

Training Efficiency

- Baseline: 10K trajectories, 3 CPU-years
- QUANT: 100 trajectories, 8 CPU-days
- **99% reduction, <5% accuracy loss**

Extended Capabilities

- Works without abandonment
- Arbitrary time-varying schedules
- Single framework, multiple distributions

QUANT: Takeaways

What We Demonstrated

1. Time-varying capacity adds huge complexity
2. Transfer learning is highly effective
3. 99% reduction in simulation requirements
4. Eliminates PSS bottleneck

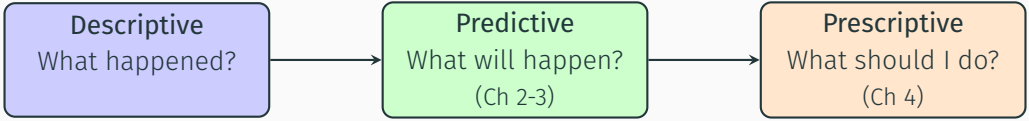
Key Innovation

Structural similarity exploitation:

- Markovian & non-Markovian share mean-driven dynamics
- Learn general dynamics from cheap oracle
- Adapt to specifics with limited data

Next: Can we optimize?

The Ultimate Goal: Prescriptive Analytics



The Capacity Planning Problem
Given:

- Arrival forecast $\lambda(t)$
- Service-level constraints (multiple, simultaneous)
- Cost per server

Find:

- Optimal capacity schedule $n^*(t)$
- Minimize cost
- Satisfy *all* constraints at *all* times

Multi-Constraint Optimization

Typical Requirements

- Mean wait time: $\mathbb{E}[W(t)] \leq \tau_W$
- Queue length: $\mathbb{E}[Q(t)] \leq \tau_Q$
- Tail probability: $\mathbb{P}(W(t) > w) \leq \tau_P$
- Abandonment: $\mathbb{P}(W(t) > A) \leq \tau_A$

Some must hold simultaneously!

Challenge

- Constraints often conflict
- Each candidate needs full evaluation
- 100s-1000s evaluations to find optimal
- Traditional: Weeks of computation

QUANT-OS: Three Optimization Methods

Binary Search

Exploits monotonicity

Pros:

- Guaranteed convergence
- Robust
- 7-10 iterations/step

Time: 1.2 seconds

Batch Evaluation

Tensor parallelization

Pros:

- Fastest method
- No iteration
- Guaranteed feasibility

Time: 0.8 seconds

Gradient-Based

Automatic differentiation

Pros:

- Sensitivity analysis
- Exploit differentiability
- Gradient descent

Time: 10 seconds

All methods: **Seconds** vs **Days** with simulation-based optimization

Optimization Framework

Problem Formulation

$$\begin{aligned} \min_{n(t)} \quad & \sum_{t=0}^{T} c \cdot n(t) \\ \text{s.t.} \quad & \Pi(\mathsf{S}_t) \leq 0, \quad \forall t \\ & \mathsf{S}_{t+1} = \mathcal{F}(\mathsf{S}_t, \lambda(t+1), n(t+1)) \end{aligned}$$

Recurrent Strategy:

1. At each time t: Find minimum $n^*(t+1)$ satisfying constraints
2. Compute next state using QUANT
3. Proceed to next time-step

QUANT-OS: Convergence & Performance

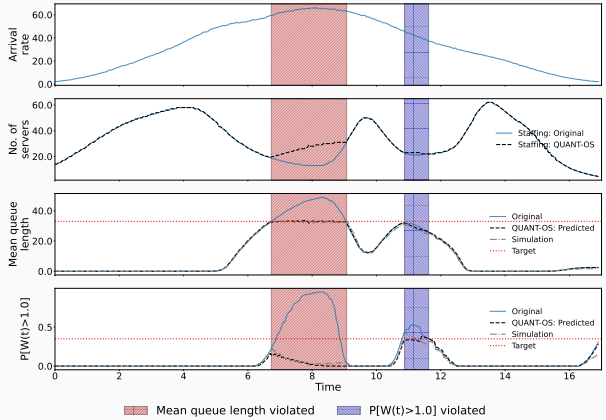
Method	Iterations/Step	Time (sec)	Convergence
Binary Search	8.2 ± 0.5	1.2	100%
Batch Evaluation	1.0 ± 0.0	0.8	100%
Gradient-Based	34.5 ± 17.7	10.1	97.2%

Speedup Analysis

- QUANT-OS: 0.8-10 seconds for full optimization
- Simulation would require years!
- **Speedup: multiple orders of magnitude**
- Enables real-time replanning, interactive exploration

Results across 200 test scenarios, horizon T=17 (340 steps)

Case Study: Multi-Constraint Optimization



Dual constraints: $\mathbb{E}[Q(t)] \leq 33$ AND $\mathbb{P}(W(t) > 1) \leq 0.35$

Result: QUANT-OS identifies two intervention periods, 96.7% satisfaction

QUANT-OS: Key Results

Performance

- Time: 0.8-10 seconds (vs hours for simulation)
- Speedup: 3-4 orders of magnitude
- Constraint satisfaction: 94-98%

Impact

Transforms capacity planning:

- Static advance → Dynamic real-time
- Single constraint → Multi-constraint
- Batch offline → Interactive continuous

Completes the journey: Predict → Scale → Optimize

Computational Performance: The Big Picture

Task	Traditional	Our Method	Speedup
Single prediction	2-3 hours	0.2-0.35 sec	$>10{,}000\times$
Training data (non-Mark.)	3 CPU-years	8 CPU-days	$\sim 100\times$
Capacity optimization	multiple years	0.8-10 sec	$>10{,}000\times$

What This Enables

- **Real-time replanning:** As forecasts update throughout the day
- **Interactive exploration:** "What if demand increases 10%?"
- **Continuous optimization:** Intraday capacity updates
- **Large-scale deployment:** Practical for routine operations

Paradigm shift: From offline analysis to interactive real-time

Key Contributions

1. **NeuraliNQ:** First neural framework for $G_t/GI/n + GI$
   - PSS-TRF decomposition
   - 10,000× speedup
2. **QUANT:** Transfer learning for $G_t/GI/n_t + GI$
   - 99% data reduction
   - Works without abandonment
3. **QUANT-OS:** Multi-constraint optimization
   - Seconds to optimize
   - 94-98% constraint satisfaction

Methodological Impact

Machine Learning
- Domain-informed architectures
- Transfer learning for stochastic systems
- Integration with optimization

Operations Research
- ML for core OR tasks
- Real-time decision support
- Multi-constraint handling

Queueing Theory
- Parsimonious states work
- Alternative to heavy-traffic
- Simulation accuracy + analytical speed

Practice
- Democratizes analysis
- New operational paradigms
- Computationally feasible

Limitations & Future Directions

Future Work

- Parametric distributions
- Multi-class systems
- Queueing networks

Vision: Expand to a general modeling tool for queueing systems

What We've Shown

Neural networks predict and optimize complex queueing systems with simulation-level accuracy at millisecond-level speed

1. Neural networks work for queueing systems
2. Transfer learning makes training practical
3. Fast prediction enables real-time optimization
4. 94-98% constraint satisfaction in validation

Not just faster simulation—enables new capabilities

Publications

Journal Papers

- Spyros Garyfallos and Yunan Liu and Pere Barlet-Ros and Albert Cabellos-Aparicio, "NeuraliNQ: A Neural Network Method for the Transient Performance Analysis in non-Markovian Queues." **Queueing Systems: Theory and Applications**. Paper accepted on September, 4th, 2025.
- Spyros Garyfallos and Yunan Liu and Xianzhe Wang and Pere Barlet-Ros and Albert Cabellos-Aparicio, "Performance Analysis and Control in Nonstationary Non-Markovian Queues: A Transfer Learning Neural Network Framework" **Informs Journal on Computing** (Under review).

Conference Papers

- Spyros Garyfallos and Yunan Liu and Pere Barlet-Ros and Albert Cabellos-Aparicio, "Service Level Prediction in Non-Markovian Nonstationary Queues: A Simulation-Based Deep Learning Approach." **Winter Simulation Conference (WSC), 2024, pp. 2655-2666**.
- Spyros Garyfallos and Yunan Liu, "A deep learning approach for performance analysis and capacity planning of time-varying queues." **1st Amazon Machine Learning Conference Workshop on End-to-End Capacity Planning with Machine Learning, Optimization, and LLMs)**, Seattle, WA, USA, 2025

Bridging Two Research Paradigms

The Operations Research Community Context

- Decades of rigorous mathematical analysis and theoretical foundations
- Strong tradition of analytical proofs and asymptotic guarantees
- Queueing theory: problems that have taken researchers years to solve analytically

Earning the Community's Trust

- Very long cycles from submission to acceptance
- Multiple review cycles without major technical objections
- Reviewers' questions focused on: *methodology validity, generalizability, comparison to theoretical approaches*

Reflection: Paradigm Shifts Take Time

- Data-driven approaches complement, not replace, theoretical work
- Natural skepticism: decades of analytical work vs. fast and flexible data-driven solutions
- Both paradigms offer unique value—our work aims to demonstrate their synergy
- Community gradually embracing hybrid methodologies

Thank You!

Questions?

References i

[1] Lawrence Brown, Noah Gans, Avishai Mandelbaum, Anat Sakov, Haipeng Shen, Sergey Zeltyn, and Linda Zhao. **"Statistical analysis of a telephone call center: A queueing-science perspective".** In: *Journal of the American Statistical Association* 100.469 (2005), pp. 36–50.

[2] Linda V Green, Peter J Kolesar, and Joao Soares. **"Queueing systems with retrials: A survey".** In: *Manufacturing & Service Operations Management* 9.3 (2007), pp. 242–264.

[3] International Labour Organization. ***Number of employees worldwide from 1991 to 2022, by broad sector.*** ILO Statistics, via Statista. 1.7 billion people employed in services sector globally in 2022. 2023. URL: `https://www.statista.com/statistics/1259198/global-employment-figures-by-sector/`.

[4] Yunan Liu and Ward Whitt. **"The $G_t/GI/s_t + GI$ many-server fluid queue".** In: *Queueing Systems* 71.4 (2012), pp. 405–444.

References ii

[5] NITI Aayog. ***Working Paper: Identifying Potential Service Sub-Sectors: Insights from GVA.*** Tech. rep. Services contribute around 67-70% of global GDP, worth approximately USD 67 trillion in 2022. National Institution for Transforming India, 2024. URL: https://www.niti.gov.in/sites/default/files/2024-12/Working%20Paper_Identifying%20Potential%20Service%20Sub-Sectors%20Insights%20from%20GVA_New.pdf.

[6] Qualtrics XM Institute. ***The global cost of poor customer experience.*** Tech. rep. Organizations risk $3.1 trillion globally due to poor customer experiences. Qualtrics, 2020. URL: https://www.jindalx.com/blog/impact-of-call-abandonment-rate-in-call-center-services/.

[7] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. ***Implicit Neural Representations with Periodic Activation Functions.*** 2020. arXiv: 2006.09661 [cs.CV].

References iii

[8] Temkin Group. ***The Financial Impact of Customer Experience.*** Tech. rep. A moderate improvement in CX would impact the revenue of a typical $1 billion company an average of $775 million over three years. Temkin Group, 2017.

[9] World Trade Organization. ***Trade in services for development.*** Tech. rep. Services sector generates 50% of global employment and 67% of global GDP. World Trade Organization, 2023. Chap. 12, pp. 1–50. URL: https://www.wto.org/english/res_e/booksp_e/trade_in_serv_devpt_chp1_e.pdf.

[10] Galit B. Yom-Tov and Avishai Mandelbaum. **"Erlang-R: A Time-Varying Queue with Reentrant Customers, in Support of Healthcare Staffing".** In: *Manufacturing & Service Operations Management* 16.2 (May 2014), pp. 283–299. ISSN: 1526-5498. DOI: 10.1287/msom.2013.0474. URL: http://dx.doi.org/10.1287/msom.2013.0474.

Backup Slides

NeuraliNQ Architecture Details

PSS Network:

- 11 layers, width 500
- Dual-branch: Tanh + Sinusoidal activations
- Captures smooth and non-smooth steady-state curves
- Input: $\lambda \rightarrow$ Output: $(\mathbb{E}[W^{\infty}], \mathbb{E}[B^{\infty}])$

TRF Network:

- 9 layers, width 2000
- Tanh + ReLU activations
- Input: $(\mathbf{S}_t, \mathbf{S}_{t+1}^{\infty}) \rightarrow$ Output: $\Delta \mathbf{S}_{t+1}$
- Larger capacity for complex mapping

Training:

- L1 loss (robust to outliers)
- Adam optimizer, very small learning rates (10^{-7})
- 5000 epochs (PSS), 2000 epochs (TRF)
- Sequential training

QUANT Tandem Architecture

Count Network ($\mathcal{F}^Q$):

- Predicts changes: $\Delta\mathbb{E}[Q]$, $\Delta\mathbb{E}[B]$
- 16 layers (deeper for complex dynamics)
- Handles discrete state transitions
- Tanh + Softplus + ReLU activations

Time Network ($\mathcal{F}^W$):

- Maps queue state to waiting time
- 12 layers (shallower, smoother function)
- Continuous predictions
- Depends on predicted queue state from $\mathcal{F}^Q$

Why Tandem?

- Count metrics: Sharp, discrete transitions
- Time metrics: Smooth, continuous evolution
- Specialized processing improves accuracy

MQSolver Algorithm

Problem: Compute exact performance for $M_t/M/n_t + M$ systems

Approach:

1. Birth-death process with time-varying rates
2. Uniformization: Convert to discrete-time
3. Select rate ν exceeding all transition rates
4. Matrix-vector multiplication: $\boldsymbol{\pi}(t + \Delta t) = \mathsf{P}(t)\boldsymbol{\pi}(t)$
5. Compute performance from $\boldsymbol{\pi}(t)$

Complexity:

- State space truncation at $X_{\max}$
- Sparse tridiagonal matrix
- Linear in time steps and state space
- Independent of stochasticity

Result: 60 seconds per trajectory vs 900 for simulation ($15\times$ speedup)

Transfer Learning in Detail

Four-Phase Process:

1. **Develop MQSolver**
   - Exact Markovian performance oracle
   - 60 sec per trajectory
2. **Generate 10K Markovian data**
   - Diverse $\lambda(t), n(t)$ patterns
   - 7 CPU-days total
3. **Train MQNet**
   - Learn general dynamics
   - Depends on mean rates
4. **Fine-tune NMQT**
   - 100 non-Markovian trajectories
   - Learn distributional corrections
   - Very small learning rate (10^{-7})

Optimization Algorithm Pseudocode

for $t \leftarrow 0$ to $T-1$ **do**
  if $\Pi(\mathcal{F}(\mathbf{S}_t, \lambda(t+1), n_{\text{plan}}(t+1))) \leq 0$ **then**
    $n^* \leftarrow n_{\text{plan}}(t+1)$ ▷ Baseline satisfies
  else
    if method = Binary **then**
      $n^* \leftarrow \text{BinarySearch}(\mathbf{S}_t, \lambda(t+1))$
    else if method = Batch **then**
      $n^* \leftarrow \text{BatchEval}(\mathbf{S}_t, \lambda(t+1))$
    else
      $n^* \leftarrow \text{GradientDescent}(\mathbf{S}_t, \lambda(t+1))$
    end if
  end if
  $\mathbf{S}_{t+1} \leftarrow \mathcal{F}(\mathbf{S}_t, \lambda(t+1), n^*)$
end for
return $\{n^*(t)\}$

Key: Each time step solved independently using fast QUANT evaluations

Experimental Setup

Test Scenarios

- 200 diverse scenarios
- Arrival rates: 0-100 per time unit
- Capacity range: 1-150 servers
- Distributions: Markovian, Erlang-4, Hyperexp-2
- Horizon: T=17 time units (340 steps)

Validation

- **Stage 1:** QUANT predictions
- **Stage 2:** Independent simulation (10K replications)
- Metrics: TAMSE, constraint satisfaction rate
- Hardware: 96-core cluster, Tesla V100 GPU

Additional Experimental Results

System Configuration	TAMSE	Satisfaction
Markovian	0.121	98.5%
Erlang-4 (low var)	0.224	97.0%
Hyperexp-2 (high var)	0.360	95.5%
Small scale ($n < 20$)	0.156	96.5%
Medium scale ($20 \leq n < 50$)	0.140	97.0%
Large scale ($n \geq 50$)	0.132	98.5%

Observations:

- Consistent performance across distributions
- Improves with scale (as expected)
- All above 95% satisfaction threshold

Comparison to Traditional Methods

Method	Accuracy	Speed	Applicability
Heavy-traffic (Fluid/Diffusion)	Medium-High (scale-dependent)	Fast (seconds)	Large scale only Markovian preferred
MOL	Medium (steady-state)	Fast (seconds)	Markovian Single metric
Simulation (ISA)	High (affects speed)	Very Slow (days-weeks)	General Any system
QUANT-OS	**High (94-98%)**	**Fast (seconds)**	**General Non-Markovian**

QUANT-OS: Best of all worlds

Markovian vs. Non-Markovian: The Key Distinction

What makes the difference?

Markovian systems ($M_t/M/n_t + M$):

- Memoryless property: Future depends only on current state
- State = simple queue length
- Exact solutions via uniformization and matrix methods
- Computationally tractable

Non-Markovian systems ($G_t/GI/n_t + GI$):

- Must track customer ages, residual service times
- State space becomes infinite-dimensional
- No closed-form solutions
- Heavy-traffic approximations are complex
- Simulation is slow and expensive

The scientific challenge: How do we analyze infinite-dimensional systems efficiently?

Two-Stage Validation

Stage 1: Prediction Validation

1. QUANT-OS determines optimized $n^*(t)$
2. QUANT predicts performance under $n^*(t)$
3. Verify constraints satisfied in *predictions*

Result: 100% satisfaction (by construction)

Stage 2: Simulation Validation

1. Simulate actual system with $n^*(t)$
2. Monte Carlo: 10K replications
3. Measure *true* constraint satisfaction

Result: 94-98% satisfaction (accounts for prediction errors)

Gap between stages quantifies robustness to prediction errors

Constraint Satisfaction in Simulation

Constraint Type	Prediction Error	Violation Rate
$\mathbb{E}[W] \leq w$ (loose)	0.082	2.5%
$\mathbb{E}[W] \leq w$ (tight)	0.107	3.5%
$\mathbb{E}[Q] \leq q$ (loose)	0.080	2.0%
$\mathbb{E}[Q] \leq q$ (tight)	0.096	3.0%
$\mathbb{P}(W > w) \leq \tau$	0.131	5.0%
Multi-constraint	0.140	6.0%

Interpretation

- **94-98% satisfaction:** High reliability for deployment
- **2-6% violations:** Modest, due to prediction errors
- **Tighter constraints:** Slightly higher violation rates
- **Multi-constraint:** Most challenging, still 94% success

The impact of abandonment