Queueing Systems (2025) 109:24
https://doi.org/10.1007/s11134-025-09952-3

NeuraliNQ: a neural network method for the transient performance analysis in non-Markovian Queues

Spyros Garyfallos[1,2] · Yunan Liu[3] · Pere Barlet-Ros[1] · Albert Cabellos-Aparicio[1]

Received: 22 January 2024 / Revised: 31 August 2025 / Accepted: 4 September 2025

Abstract

Many empirical studies have confirmed that service-time and patience-time distributions in service systems (e.g., call centers and health care) are far from exponentially distributed. Because non-Markovian queues are rarely amenable to analytic solutions, performance analysis often resorts to approximating methods such as heavy-traffic fluid limits or computer simulations. In this paper, we contribute to the literature on transient performance analysis of non-Markovian queues by developing a new *neural networks method*, dubbed *Neural network in non-Markovian Queue* (NeuraliNQ); we specifically focus on queues with customer abandonment. NeuraliNQ is an offline supervised learning method that uses synthetic training data to learn the system's intrinsic characteristics. In real-time applications, NeuraliNQ can recurrently estimate the transient system waiting time performance in a finite time window. Our results confirm that NeuraliNQ is able to achieve the proper balance between efficiency and accuracy: on the one hand, it is four orders of magnitude computationally more efficient than Monte-Carlo simulations; on the other hand, it yields higher solution accuracy than standard approximation methods such as the heavy-traffic fluid model, especially when the system scale is not too large.

The work performed in this paper does not relate to the first two authors' Amazon affiliation.

✉ Spyros Garyfallos
spyridon.garyfallos@upc.edu

Yunan Liu
yunanliu@amazon.com

Pere Barlet-Ros
pere.barlet@upc.edu

Albert Cabellos-Aparicio
alberto.cabellos@upc.edu

1 Department of Computer Architecture, Universitat Politécnica de Catalunya, Catalunya, Barcelona, Spain

2 Amazon, Amazon Web Services (AWS), New York City, NY, USA

3 Amazon, Supply Chain Optimization Technology, New York City, NY, USA

Published online: 16 October 2025

Keywords Non-Markovian queues · Transient performance analysis in queues · Neural networks in queues · Recurrent neural network

1 Introduction

In the realm of queueing systems, we encounter two primary types: (i) Markovian queues, characterized by Poisson arrivals and exponential service-time distributions, and (ii) non-Markovian queues, featuring nonexponential distributions. In general, the choice between these models involves a trade-off between tractability and practicality. Non-Markovian queues, the latter category, find application in a broader range of real-world scenarios due to their ability to incorporate complex and realistic model features. On the other hand, Markovian queues, the former type, offer mathematical tractability and ease of analysis through standard Markov chain methods.

Empirical studies consistently indicate that service times and abandonment times in contemporary service systems deviate significantly from exponential distributions. Instances of lognormal-like distributions are prevalent, such as observed call durations in customer contact centers [5] and patients' lengths of stay in hospitals [48]. Certain cases may allow effective approximations of non-Markovian queueing systems by their Markovian counterparts (with the nonexponential distributions replaced by exponential distributions). For example, steady states of the $M/G/n/n$ loss model are insensitive to the service distribution beyond the mean. However, recent research emphasizes the significance of non-Markovian features, extending beyond means, in analyzing both steady states [1, 55] and transient performance [37, 40] of these queueing systems. These revelations continue to motivate queueing theorists to explore more realistic queues with non-Markovian probability structure. The present work is part of the ongoing efforts in the performance analysis of non-Markovian queues.

1.1 Challenges of non-Markovian queueing systems

The analytical investigation of models with non-Markovian probability structure is often notoriously difficult and is rarely amenable to analytic solutions [1]. Moreover, describing the transient queueing performance necessitates capturing the nonstationary dynamics over time, introducing additional complexities. In general, analyzing a queueing system requires one to capture the (i) *stochastic variability*, which is determined by the complex probabilistic structure, and (ii) *time variability*, which is due to the nonstationary model parameters (e.g., arrival rate). Characterizing the stochastic variability in a non-Markovian queue is much more involved than its Markovian counterpart because a simple birth-and-death process can no longer represent the system dynamics. For example, to fully capture the dynamics of a multi-server $G/GI/n+G$ queue having nonexponential service and abandonment times, besides the total number of customers in the system, one needs to keep track of the elapsed waiting times of all waiting customers and the elapsed service times of those in service; see [39, 54] for the two-parameter age-and-time representations for the $G/GI/n+GI$ system. In addition, the transient analysis is more challenging than computing the steady-

state distributions since the transient performance (i.e., the waiting time at t) largely depends on its preceding values (i.e., the waiting time at $s < t$).

1.2 On heavy-traffic methods for non-Markovian queues

In order to analyze non-Markovian queueing systems, researchers often resort to approximating methods arising from large-scale limits. Fluid limit is a predominately used model [35, 37, 38, 55]. As the first-order approximation for the corresponding queueing system, a fluid model intends to focus on characterizing the system's temporal dynamics while omitting its stochastic variability. Fluid limits are established via the *functional law of large numbers* (FLLN), which requires sufficiently scaling up the queueing system's demand and service capacity. Conceptually, in a fluid model, all customers are "shrunk to atoms of fluid" so that system-level deterministic functions can capture their aggregated behavior. The computation of fluid models of non-Markovian queues often involves numerically solving a system of differential equations [37]. Consider a system with scale n (e.g., n is the number of servers), the mean queue length $\mathbb{E}[Q_n(t)]$, server's occupancy $\mathbb{E}[B_n(t)]/n$ (i.e., mean number of busy servers $\mathbb{E}[B_n(t)]$ over the number of servers n), and waiting time $\mathbb{E}[W_n(t)]$ can be approximated by their fluid counterparts as below:

$$\frac{1}{n}\mathbb{E}[Q_n(t)] \approx Q(t), \quad \frac{1}{n}\mathbb{E}[B_n(t)] \approx B(t) \quad \text{and} \quad \mathbb{E}[W_n(t)] \approx w(t) \quad \text{for } n \text{ large}, \tag{1}$$

where $Q(t)$, $B(t)$, and $w(t)$ are the corresponding fluid limits of the queue length, occupancy, and waiting-time processes.

Fluid limits can help approximate the system's "average" performance as a function of time, such as the mean waiting time $\mathbb{E}[W(t)]$ and mean queue size $\mathbb{E}[Q(t)]$. Nevertheless, its deterministic nature hinders its ability to capture performance functions encompassing distributional information beyond the mean. For example, this limitation is particularly evident when dealing with the tail probability of delay $\mathbb{P}(W(t) > \tau)$, a predominantly used service-level target in modern service systems that accounts for the fraction of customers experiencing wait times exceeding a target $\tau > 0$ [32]. Another major drawback of fluid methods is the degradation of solution accuracy in medium- and low-traffic systems. In addition, fluid models may become inaccurate when the system constantly alternates between low-traffic and heavy-traffic scenarios [37] (e.g., practical queueing systems with natural daily cycles transition between night-time low traffic levels and rush hour peaks). Also, solving fluid functions of non-Markovian models is not straightforward and can be challenging for practitioners.

Diffusion limits arising from the *functional central limit theorem* (FCLT) can be used as a stochastic refinement of the deterministic fluid model. In many cases, these FCLT limits can effectively account for the random fluctuations of the system processes around their mean values. Nevertheless, the FCLT approximations may fall short in the following cases: First, FCLT limits are primarily available for Markovian queueing systems [41]; there are several recent successes in the development of FCLT limits for queues with nonexponential abandonments, see [21, 32, 33, 39] and references

therein. Unfortunately, the development of such a limit becomes extremely challenging for queues with nonexponential service times [1, 40]. Next, the analysis and solution formulas of diffusion limits are often quite involved [39] and not easy to implement in practice. Last, when the system scale is smaller, similar to the fluid model, its solution accuracy degrades.

In this paper, we contribute to the performance analysis of non-Markovian queues by proposing a novel neural network approach. This approach is inspired by the recent successes of neural networks in computer vision and natural language processing [15, 18], where deep learning became the state-of-the-art, exceeding all past approaches based on more conventional methods. Our new method, called *neural network in non-Markovian queue* (NeuraliNQ), will produce effective predictions for transient queueing performance functions for the non-Markovian nonstationary queueing systems. In particular, we focus on queues with customer abandonment, which are predominant in service systems such as call centers and healthcare [17].

1.3 Queueing systems vs. recurrent neural networks

Among the large volume of neural network literature, we are particularly inspired by *recurrent neural networks* (RNNs). Distinct from traditional deep neural networks which assume that inputs and outputs are independent of each other, RNNs have the ability to maintain internal state (also called hidden representation or simply, memory) of past inputs, and to characterize how they affect future system states. Their ability to process sequential data and track temporal dependencies makes them widely applicable to time series prediction tasks, including forecasting and system dynamics modeling.

Similar to time series prediction, in queueing systems, the transient trajectory of the performance function (e.g., waiting times) $\boldsymbol{S}_{t-1}$, $\boldsymbol{S}_t$, $\boldsymbol{S}_{t+1}$, ..., is a sequence of inter-correlated system states indexed by time t. The distribution of $\boldsymbol{S}_{t+1}$ is jointly determined by (i) the **external system input** $\boldsymbol{\beta}_t$ (e.g., the arrival rate λ_t), and (ii) the **internal memory** of the previous state of the system (e.g., the waiting time at t). For example, the system's congestion level at $t+1$ is determined by both the congestion level at t and the new demand between t and $t+1$.

At the heart of NeuraliNQ is a one-step lookahead learning procedure that learns to recurrently predict the next-step system state (at time $t+1$) using the latest state (at time t), utilizing the intrinsic correlation of the system dynamics between the two time steps. Two neural networks trained using simulated data within a local problem space accomplish this stepwise prediction. Figure 1 provides a schematic illustration of NeuraliNQ formed as an unrolled (unfolded) computational graph into a full network.

In contrast to conventional RNN models, where hidden representations are constructed by sequentially replaying past information in order to provide the past context up until the point of inference, we explore the option of using *snapshot state information* (information only from the previous time frame) and experiment with various state feature combinations that are descriptive enough to remove the need for contextual reconstruction of hidden representations. We show that the state snapshot comprising of the latest waiting time and service occupancy are descriptive enough

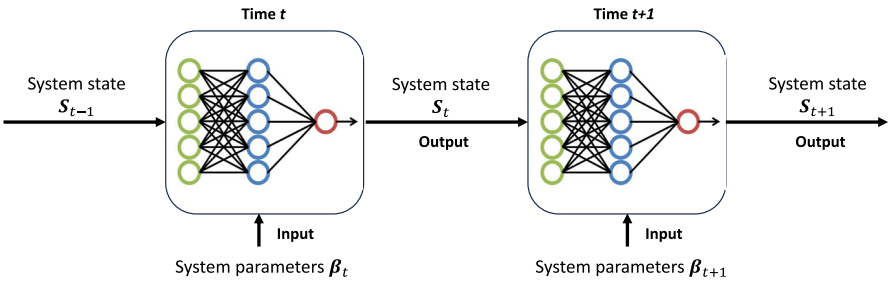


Fig. 1 NeuraliNQ: the feedforward topology of time-indexed RNNs

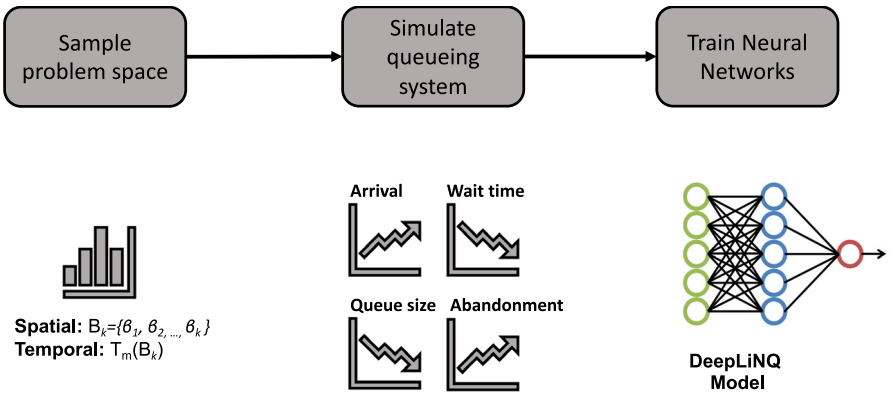


Fig. 2 Flow chart of NeuraliNQ

for high-accuracy predictions. The previous statement of this minimum snapshot state information holds even for predicting multiple common performance metrics, such as the *tail probability of delay* (TPoD), queue length and abandonment probability. This discovery allows us to construct a training data generation strategy that is straightforward and computationally parsimonious, reduces our neural network's architecture size and complexity by removing the need for internal memory, and simplifies the inference data inputs to only the most recent state snapshot.

The development of NeuraliNQ follows three steps: First, we sample from the selected local space of the model parameters and conduct offline simulations to generate data from the system's stationary and transient samples. Through these extracted features, we train two neural networks. The first neural network is used to predict a sequence of t-indexed pointwise steady state (PSS) assuming that the system is operated under stationary model parameters at each t (e.g., stationary arrival rate, service-time, and patience-time distributions). We subsequently use the estimated PSS and the current state (at time t) to predict the next-step state (at $t+1$) via our second neural network. The stepwise nature of the model enables recurrent predictions on arbitrarily long horizons. See Fig. 2 for an illustration of these steps.

We position NeuraliNQ within the broader landscape of computational methods aimed at solving complex system dynamics by leveraging domain knowledge and

system structure. Recent advances in machine learning, such as physics-informed neural networks (PINNs) [44], have shown promise in solving differential equations that arise in physical systems. However, while Markovian queueing models can often be characterized by relatively simple differential equations, non-Markovian models are substantially more complex. Although it is theoretically possible to derive the governing equations for queue lengths by largely expanding the state space of the system - incorporating auxiliary variables such as the elapsed ages of customers in service and in queue - the resulting models remain generally complicated. NeuraliNQ acts as a queueing-specialized neural operator [28] that learns the mapping from the current system state to its future evolution, without requiring explicit formulations of the underlying equations. Its architecture is guided by PSS as a supervisory structure. One can think of PSS as a time-indexed boundary condition. Since the differential operator is time-varying, we are "solving" an equation whose boundary condition is changing at each point in time, and is unknown. A key limitation, however, is that the PSS requires the presence of customer abandonment when the pointwise arrival rate exceeds the service rate (PSS is not well-defined for overloaded queues without customer abandonment).

This paper presents an initial attempt to train neural networks to capture the temporal dynamics of queueing systems under nonstationary arrival patterns. While NeuraliNQ is not restricted to specific service or abandonment time distributions because it is trained on simulated queueing data, the current framework focuses on predicting transient waiting time dynamics under varying arrival rates, assuming fixed service and abandonment distributions. To accommodate a different combination of service and abandonment characteristics, the model needs to be retrained on new simulated training data using these distributions. The current NeuraliNQ framework is particularly well-suited for scenarios in which customers belong to a single class, meaning their service and abandonment behaviors remain consistent, while their arrival patterns may fluctuate from day to day.

1.4 An example

We first give a quick illustration of the performance of NeuraliNQ. We consider a non-Markovian multi-server $M_t/H_2/n + E_2$ model, having a nonhomogeneous Poisson arrival process with n-scaled rate $\lambda_n(t) = n(1 + 0.6\sin(t))$ (top panel of Fig. 3), service times following a hyperexponential (H_2) distribution with mean $1/\mu = 1$, and abandonment times following an Erlang-2 (E_2) distribution with mean $1/\theta = 1$. Specifically, the *probability density functions* (PDFs) for the abandonment and service times are respectively:

$$f_a(x) = 4\theta^2 x e^{-2\theta x} \quad \text{and} \quad g(x) = p\mu_1 e^{-\mu_1 x} + (1-p)\mu_2 e^{-\mu_2 x},$$

where $p = 0.5(1 - \sqrt{0.6})$, $\mu_1 = 2p$, and $\mu_2 = 2(1-p)$.

In Fig. 3 we graph the transient trajectories of the expected mean waiting time and service occupancy (in form of (1)) generated from (i) NeuraliNQ, (ii) fluid approximation using [37], and (iii) crude Monte-Carlo (MC) simulation (the "ground truth"), for

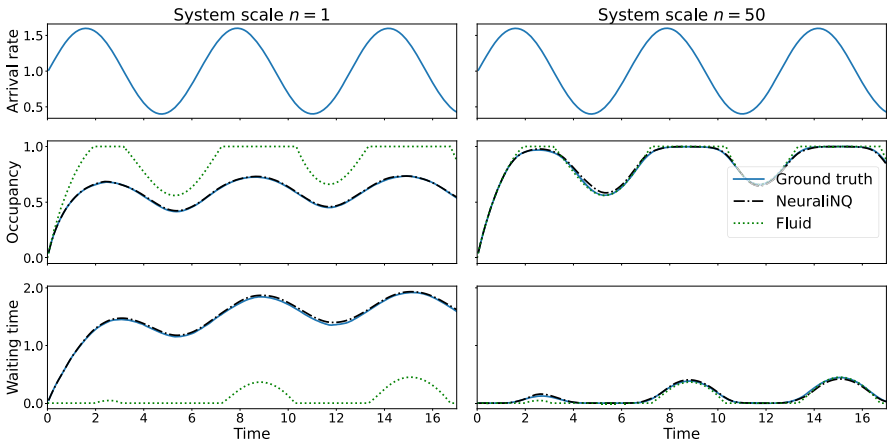


Fig. 3 NeuraliNQ vs. fluid model: Expected waiting time and service occupancy (mean number of busy servers divided by total servers) of an $M_t/H_2/n + E_2$ queue at two different scales

low- and large-scale systems. We observe that both NeuraliNQ and the fluid approximation work effectively when the system scale is large, while NeuraliNQ provides a more accurate prediction than the fluid model when the scale is small. We will conduct additional numerical experiments in Sect. 4 to evaluate the performance of NeuraliNQ further. Also, we will provide in-depth discussions on additional advantages of NeuraliNQ over the fluid model and heavy-traffic methods; see Sect. 5.

Remark 1 (The descriptive power of server occupancy) In large-scale systems (e.g., right-hand panel in Fig. 3), the waiting time alone can be used to describe the system's congestion level during an overloaded interval, because the occupancy is close to 100%, giving redundant information. However, in small-scale systems (e.g., left-hand panel in Fig. 3), the system exhibits a larger stochastic variability, so that there is no clear separation of underloaded and overloaded intervals. To see this, note that throughout the entire interval, the waiting time is strictly positive and the occupancy is strictly less than 100%. In this case, both the waiting time and server occupancy are needed, which will jointly describe the system's evolutionary state.

1.5 Contributions and organization

We summarize the main contributions of this paper:

(i) To the best of our knowledge, this is the first attempt at applying neural networks to study the transient performance of non-Markovian nonstationary queueing systems. As a supervised learning method, NeuraliNQ is able to recurrently predict the system's transient performance functions (e.g., expected waiting time) in a step-wise lookahead manner, taking advantage of the intrinsic characteristics of the system dynamics in the temporal domain. Distinct from typical black-box neural network models, NeuraliNQ stands out as a customized model with queueing-specific interpretability: First, NeuraliNQ introduces a novel approach

of integrating transient and stationary performance, two aspects often studied independently in queueing theory. It achieves this by learning the system's transient performance over time, treating the pointwise steady state as a "supervisor". Next, the stepwise lookahead RNN topology embedded in NeuralinQ empowers it to predict transient performance by leveraging previous system dynamics. This not only ensures its high efficiency but also enhances its robustness.

(ii) Our findings offer useful insights that can help improve the modeling and management of queueing systems. NeuralinQ helps identify the essential state information needed for predicting and analyzing useful performance metrics in both temporal and spatial dimensions: First, our proposed latest snapshot state representation ensures sufficiency for modeling non-Markovian systems, eliminating the necessity for retaining additional memory of past states. Next, NeuralinQ demonstrates its capability to predict queueing service-level metrics using simple state information from both the queue side (e.g., mean waiting time) and the server side (e.g., mean number of busy servers).

(iii) We confirm the effectiveness of NeuralinQ by conducting comprehensive numerical experiments. Besides the mean waiting time (which is the primary performance metric), NeuralinQ is able to compute other service-level metrics predominantly used in service systems, such as the probability of abandonment, server occupancy, mean queue length, and tail probability of delay. For service-level metrics that capture the average system values (e.g., mean waiting time and queue length), our results show that NeuralinQ yields high-fidelity solutions that outperform the heavy-traffic fluid approximations, especially when the system scale is not too large. For service-level metrics derived from waiting-time distributions beyond the means (e.g., probability of delay and tail probability of delay) where the fluid approximation is no longer applicable, NeuralinQ's predictions continue to demonstrate effectiveness. We also confirm that NeuralinQ is robust to various model inputs such as scale, arrival patterns, and distributions of the arrival and service times.

Organization of the paper. In Sect. 2 we review the relevant literature. In Sect. 3 we describe the main steps of NeuralinQ; we also discuss the interpretability of NeuralinQ. In Sect. 4 we evaluate the effectiveness of NeuralinQ by conducting a comprehensive set of numerical experiments. In Sect. 5 we provide in-depth discussions on how NeuralinQ compares to the other two conventional approaches for non-Markovian queues: heavy-traffic methods and computer simulations. Finally, we give concluding remarks in Sect. 6. We provide supplementary materials in the appendix.

2 Related literature

Three bodies of literature are relevant to the present work.

Non-Markovian queueing systems: The transient performance analysis of Markovian queueing networks has been developed by [41]. The analysis of non-Markovian queueing systems is, in general, much more challenging. To obtain tractable results

for queues having nonexponential service times and abandonment times, Whitt [54] introduced a new fluid $G/G/n+G$ model which adapts two-parameter performance functions to keep track of elapsed service and waiting time of customers. Whitt's pioneering work [54] opened a new line of research on non-Markovian queues; this representation has subsequently been extended to incorporate time-varying arrivals and staffing levels [34, 37], infinite-server queues [2], network structure [35, 38], and queues with transitory arrivals [23]. To further refine these fluid models, stochastic FCLT limits have been developed, including the time-varying OU process for the $G_t/M/n_t+GI$ system alternating between underloaded and overloaded time intervals [36, 39], the patience-time scaled diffusion approximation [21], and the Gaussian approximations for the stationary $G/GI/n+GI$ overloaded queues [1, 40]. Distinct from the above literature that studies non-Markovian models using heavy-traffic limits, the present paper proposes a neural network approach.

Machine learning in queueing systems: There is a growing literature on modeling and analyzing queueing systems using machine learning techniques. In recent studies, researchers proposed to apply online learning and reinforcement learning methodologies to support real-time decision-making in queueing systems, including pricing [24], capacity sizing [9, 10], and control policies such as routing [30, 45] and scheduling [14, 29]. Several recent works have explored using neural networks for queueing performance analysis. Baron et al. [4] developed a supervised learning approach for solving GI/GI/1 queues, focusing on stationary distributions of single-server systems without abandonments. Building on this, Sherzer [46] extends this approach to steady-state probabilities of G/GI/1 tandem queueing networks and Sherzer et al. [47] to non-stationary G(t)/GI/1 queues. While these works share our goal of applying machine learning to queueing systems, they differ in their focus on single-server queues without abandonments, whereas our approach targets multi-server queues with customer abandonment, which present additional modeling complexities. Ata et al. [3] apply neural networks to solve Hamilton-Jacobian-Bell equations of Markov decision processes in queueing control problems. Garbi et al. [16] apply neural networks to study deterministic telecommunication queueing networks. Ojeda et al. [42] develop an adversarial framework that enhances service time modeling by incorporating recurrent structures, leading to improved predictive accuracy in dynamic service environments. Raeis et al. [43] utilize mixture density networks to predict the distribution of waiting times in customer service systems. Cheng et al. [11] use transformer models to classify scheduling policies in queueing networks based on partial monitoring data. Neural networks have also been adopted for parameter estimations in queueing systems including the latent stochastic intensity of a doubly stochastic process [51] and infinite server queues driven by Cox processes [52]. More recent work from Che et al. [7] proposes a differentiable discrete event simulation for queuing network control, leveraging modern auto-differentiation frameworks from the ML domain. It is worth mentioning here there exist some computational and simulation platforms for general queueing systems, such as Qplex[1] and SiMLQ.[2] Distinct from the previous literature, the present paper develops a neural network method for non-Markovian nonstationary queues;

[1] https://qplex.org/

[2] https://www.simlq.com/

we aim to characterize the transient performance in a stepwise lookahead fashion, drawing from the idea of recurrent neural networks.

Neural networks and RNNs: The research of RNN centers on their capacity to model sequential data by maintaining hidden states that adeptly capture temporal dependencies. Recent studies in this domain involve the exploration of sophisticated architectures, notably *long short-term memory* (LSTM) networks and *gated recurrent units* (GRUs), designed to mitigate the vanishing gradient problems where the impact of past observations diminishes with each recurrence, thereby enhancing the network's proficiency in learning long-term dependencies [12, 22]. The versatile applicability of RNNs is underscored across diverse domains, including but not limited to natural language processing, time series prediction, and generative tasks, affirming their efficacy in encapsulating intricate sequential patterns [13, 26]. Most recently, *transformer* [50] has provided a more modern and robust neural network architecture that eliminates the need for maintaining a hidden state, something that requires sequential processing of past information, allowing higher training parallelism and learning more complex spatial relationships of the data beyond their temporal (sequential) order. Distinct from the previous literature, the present paper develops an RNN applied to transient queueing performance analysis using only snapshot state information of the system.

3 Proposed methodology

We consider the $G_t/GI/n + GI$ queueing system having a nonstationary arrival process with rate $\lambda(t)$, *independent and identically distributed* (I.I.D.) service times following a general distribution G, n servers, and customer abandonment according to I.I.D. random variables following a general distribution F. Our goal is to compute the (transient) trajectory of queueing performance metrics such as the expected waiting time and server occupancy in a finite time. At each time step t, NeuraliNQ aims to predict the next-step queueing performance based on (i) the system state $\boldsymbol{S}_t$ at time t (the information needed to represent the "internal memory" of the system, which we will elaborate later), and (ii) the time-dependent external model input $\boldsymbol{\beta}_t$, such as the arrival rate. NeuraliNQ computes the trajectory for $\boldsymbol{S}_t$ at discrete time steps $t = 1, 2, 3, \ldots\ldots, T$, conditional on some initial $\boldsymbol{S}_0$ and future $\boldsymbol{\beta}_t$ parameters. Our key idea is a stepwise lookahead recursion that maps the present system state $\boldsymbol{S}_t$ and the model parameters $\boldsymbol{\beta}_t$ to the next-step system state $\boldsymbol{S}_{t+1}$, specifically, we write

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

where the function $\mathcal{F}$, is trained using neural networks, which will be introduced later. The training of $\mathcal{F}$ will be conducted parametrically. Specifically, we require the knowledge of the structure of the distributions (e.g., Poisson arrivals) for performing training data simulation, but not the precise values of their parameters (e.g., the exact arrival rate).

Let $\mathcal{B}$ be the spatial space of all nonstationary parameters. We assume $\mathcal{B}$ is finite and large enough to contain all possible values of these parameters. For the example of the arrival rate, $\mathcal{B} = [\underline{\lambda}, \bar{\lambda}]$ for some $\underline{\lambda} \geq 0$ and $\bar{\lambda} < \infty$.

The development of NeuraliNQ follows the steps below:

(i) **Parameter sampling.**
We generate training data for the steady-state and transient queueing performance, we draw sample values from the continuous spatial parameter space $\mathcal{B}$ following two steps. First, to learn the steady-state queue performance, we form a discrete basis $\mathcal{S} \equiv \{\widehat{\boldsymbol{\beta}}_1, \ldots, \widehat{\boldsymbol{\beta}}_k\}, \widehat{\boldsymbol{\beta}}_i \in \mathcal{B}, i = 1, \ldots, k$. We call $\mathcal{S}$ the stationary basis of $\mathcal{B}$. Next, to learn the transient queue performance, we consider a representative parametric structure and draw samples of its parameters. As we will show later, we experiment with sinusoidal parametric arrivals and sample of the amplitude and the frequency parameters of this structure within a selected problem domain of a maximum arrival volume limit and a maximum spectral (frequency) limit of the arrival domain. In this way, we form the basis of the parametric functions; We call $\mathcal{T}$ the space of transient basis.

(ii) **Training data generation.**

(a) Using each $\widehat{\boldsymbol{\beta}}_i$ in $\mathcal{S}$, we conduct simulations of a stationary queueing system under the (stationary) input parameter $\widehat{\boldsymbol{\beta}}_i$ to estimate the corresponding steady-state state values $\mathbf{S}_i^{\infty}$. This step provides training data for learning the *pointwise steady state (PSS)* of the queueing system (as if the system were a stationary model).

(b) Using each transient samples in $\mathcal{T}$, we simulate nonstationary trajectories of the system state process so we can generate training data for learning the *transient response function (TRF)* (to be specified later), which describes the transient behavior when the system's inputs (e.g., arrival rate) evolve over time.

(iii) **Building the neural networks.**
We use the training data generated in parts (a) and (b) of step (ii) to train the PSS and TRF neural networks, and then we integrate them to develop the function $\mathcal{F}$ as defined in (2).

See Sect. 2 for a schematic illustration of these steps. We next provide details for these steps. In this section, we restrict our attention to the prediction of the mean waiting time $\mathbb{E}[W(t)]$. In Sect. 4, we give results on other metrics such as the mean queue length, probability of abandonment, and tail probability of delay.

3.1 Representation of the system state

As explained in (2), NeuraliNQ aims to make stepwise predictions of the system's transient performance. In order to precisely characterize the system dynamics in a complex model, it is crucial to identify a proper system state representation which is informative enough to describe the present characteristics of the model. In a non-Markovian queue, this necessitates tracking a significant amount of information beyond the queue length (e.g., the elapsed wait times of all waiting customers and elapsed service times of those in service; see [55]). While increasing the input dimensionality of neural networks does not necessarily lead to disproportionate increases in model complexity [18], there are practical considerations regarding training efficiency and data require-

ments. This motivates us to identify the most parsimonious yet informative system state descriptor, $\boldsymbol{S}_t$, which strikes a balance between informativeness and simplicity.

In this paper, we choose $\boldsymbol{S}_t \equiv (\mathbb{E}[W(t)], \mathbb{E}[B(t)])$, which tracks the mean waiting time and server occupancy (i.e., fraction of busy servers) at t.[3] Such a state representation captures information from both sides of the queue: When the system is overloaded (underloaded), $\mathbb{E}[W(t)]$ ($\mathbb{E}[B(t)]$) is the predominant feature that describes the system's congestion level. While a one-sided descriptor might suffice in a large-scale model - where a positive waiting time implies occupancy close to 100% and a positive fraction of idle servers indicates 0 waiting time (see the right-hand panel of Fig. 2) - small-scale systems require both descriptors to jointly determine waiting time. As depicted in the left-hand panel of Fig. 3, the occupancy is not close to 100% even when the waiting time is strictly positive.

The simplicity of our state representation might seem to challenge the conventional view that tracking a system's dynamics requires more than just the mean, such as the second moment or the full distribution. However, our result will show that the mean waiting time and occupancy already capture the required information, so our neural networks do not explicitly require additional inputs. While our approach can provide accurate predictions for several key performance measures such as waiting time, queue length, and tail probabilities, it is important to acknowledge certain limitations. For performance measures that depend on complex distributional information beyond what can be captured by our state representation, additional model extensions would be required. For example, predicting the full waiting time distribution (rather than just its mean or specific quantiles) would require expanding our state space and modeling approach. Similarly, predicting customer-specific outcomes conditioned on their individual attributes would require a different modeling framework altogether. These limitations reflect the inherent tradeoff between model simplicity and the breadth of performance measures that can be accurately predicted.

3.2 Input sampling and simulation

In real service systems such as call centers, the most volatile model component is the demand process that often frequently evolves in time. Therefore, in order to predict the transient performance metrics, one should accurately capture the impact of the evolving arrival rate $\lambda(t)$ on the transient system dynamics. Comparing to the arrival rate $\lambda(t)$, other model inputs such as the service and abandonment distributions are relatively static and can be easily calibrated from historical customer data. For this reason, in the present paper we treat $\lambda(t)$ as the NeuraliNQ's external input $\boldsymbol{\beta}_t$, and assume that the service and abandonment distributions are given and remain unchanged in time. As mentioned in Sect. 1, in systems with customer abandonment (as in our $G_t/GI/n+GI$ model), the queue is always stable in the long run regardless of arrival rate, as customers who wait too long will abandon the system. This fundamental property allows us to consider arrival rates λ that exceed the total service capacity without concerns about instability in our steady-state simulations.

[3] An alternative state representation may be $\boldsymbol{S}_t \equiv (\mathbb{E}[Q(t)], \mathbb{E}[B(t)])$, which tracks the mean number of waiting customers and server occupancy at t.

In order for NeuraliNQ to make performance predictions for any $\lambda(t)$, we need to effectively train our model using training data under various arrival patterns. We do so by sampling potential values of the arrival rate within a sufficiently large space $\mathcal{B} = [\underline{\lambda}, \bar{\lambda}]$, where $0 \le \underline{\lambda} < \bar{\lambda} < \infty$. First, we sample from $\mathcal{B}$ to form a discrete basis $\mathcal{B}_k \equiv \{\widehat{\lambda}_1, \ldots, \widehat{\lambda}_k\}, \widehat{\lambda}_i \in \mathcal{B}, i = 1, \ldots, k$. We conduct long-run simulations under each $\widehat{\lambda}_i$ in order to estimate the steady-state system performance, called PSS. Next, to learn the transient queue performance, we consider cases of representative nonstationary arrival functions following parametric forms, such as sinusoidal and polynomial functions (because combinations of these functions can be used to approximate any arrival patterns). By properly sampling parameters of these parametric functions, we form a space of the bases of these parametric functions.

3.3 Pointwise steady state

The PSS at a fixed t, dubbed $\boldsymbol{S}_t^{\infty}$, is the steady state of a queueing system under a constant model parameter $\boldsymbol{\beta}_t$. Hence, at each time t, there is a t-indexed PSS. One way to interpret PSS is to imagine two separate time scales: in a nonstationary queueing system, the system's parameter $\boldsymbol{\beta}_t$ varies as t evolves on a slower scale; while at each t, there is a hypothetically faster PSS time scale on which the model instantaneously adapts the new $\boldsymbol{\beta}_t$ and converges to its steady state. At a glance, the PSS does not seem too relevant because the sequence of t-indexed PSS $\boldsymbol{S}_t^{\infty}$ is often significantly distinct from the transient state $\boldsymbol{S}_t$ at t, see Fig. 5 for an illustration. Indeed, the analysis of a queueing system's steady state and transient state are usually conducted separately in the queueing theory literature.[4] Nevertheless, as we will show later, we will build our stepwise prediction scheme by treating the PSS $\boldsymbol{S}_t^{\infty}$ as a supervisor. To generate representative training data of the steady-state waiting times, we conduct simulation experiments under the parameters sampled from space $\mathcal{B}$. A simple strategy is to sample each $\widehat{\lambda}_i$ uniformly within $[\underline{\lambda}, \bar{\lambda}]$. Alternatively, one can more frequently sample from the neighborhood of the arrival rates that may be more predominant in practice. We hereby use a uniform sampling on the logarithmic scale of the arrival rate. Under these arrival rate samples $\{\widehat{\lambda}_1, \ldots, \widehat{\lambda}_k\}$, we conduct simulations to estimate the PSS values (e.g., steady-state waiting times), denoted as $\{\widehat{\boldsymbol{S}}_1^{\infty}, \ldots, \widehat{\boldsymbol{S}}_k^{\infty}\}$, which is the set of i-indexed steady-state performance functions of the corresponding queueing system with $\widehat{\boldsymbol{S}}_i^{\infty} \equiv \mathcal{G}^P(\widehat{\boldsymbol{\beta}}_i)$, assuming each of the model parameters remains a constant $\widehat{\boldsymbol{\beta}}_i$, $i = 1, \ldots, k$. Here $\mathcal{G}^P(\cdot)$ denotes the PSS function.

See Fig. 4 for an example with 4 simulated trajectories of the mean waiting time (the solid line in the bottom panel). Each simulation should be kept long enough in order for the system to reach its near-stationary performance under stationary $\widehat{\boldsymbol{\beta}}_i$ (hereby arrival rate $\widehat{\lambda}_i$). As illustrated in Fig. 4, the mean waiting time trajectory eventually plateaus, reaching the stationary level (marked with intermittent lines). Next, we use the example in Sect. 1.4 to demonstrate the PSS. In Fig. 5, we compare the PSS curves

[4] Previous ideas of using stationary performance to study nonstationary queues arise from the optimal staffing problems, where the nonstationary staffing level n_t at time t may be approximately determined by optimizing the stationary state of a t-indexed queue; see [20] for original idea of pointwise stationary approximation, also see [32] for a review of the relevant staffing literature.

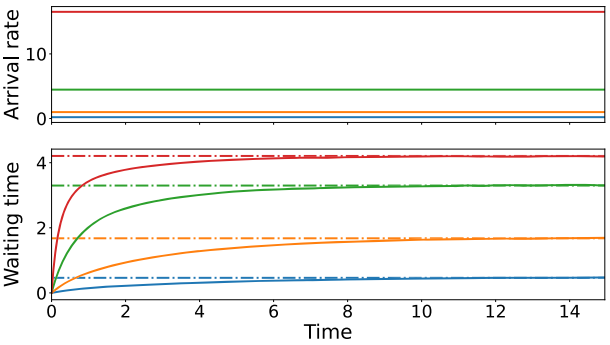


Fig. 4 Simulating the PSS values of waiting times under four arrival rate samples

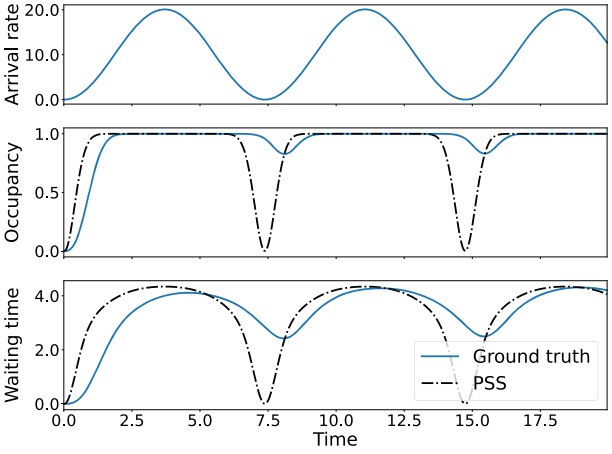


Fig. 5 Time-indexed PSS vs. transient trajectories: waiting time and occupancy

of the waiting time and occupancy to their corresponding transient paths as a function of time t. The PSS curves are significantly different from the associated transient values: PSS intends to immediately cope with the nonstationary arrival pattern, while the transient paths are less sensitive to the fluctuations in the system parameters; they react to the changes in the arrival rate after some time lag. In addition, the magnitude of the fluctuations of the actual waiting time trajectory is much less than that of its PSS.

In Fig. 5 we see how this specific system transitions between its underloaded (occupancy below 100%) and its overloaded mode, where it is operating at its full capacity (occupancy is at 100%). Both for mean occupancy and mean waiting time we observe that the PSS lines have two main characteristics. They are agnostic to the current queue length state that has accumulated over time in the system, something that forces the system to gradually drain from overloading when the arrival rate is close to zero. This creates a natural delay between PSS and the actual metrics. Secondly, we observe that both occupancy and waiting time PSS lines follow (almost) monotonically the arrival rate; when the arrival rate starts increasing, both PSS lines also increase. We show in

Fig. 9 an example of a large-scale system where this monotonicity occurs sequentially, first for occupancy and then for waiting time. This monotonic property of the vector $\boldsymbol{S}_t^{\infty} \equiv (\mathbb{E}[W(t)^{\infty}], \mathbb{E}[B(t)^{\infty}])$ uniquely characterizes every arrival rate into a future asymptotic state point where the system will tend to reach. The transition to this point is learned by our TRF neural network. We note here that for small scale systems, only one of these two features is enough to achieve this monotonic mapping to the arrival rate, since these systems typically have non-zero waiting times for very low arrival rates, and they achieve 100% occupancy at extremely high rates (these metrics never get saturated and do not lose their monotonicity).

The heart of NeuraliNQ builds on a recurrent stepwise prediction procedure, where the prediction at $t + 1$ draws from that at t. Therefore, the *bootstrapping* design of NeuraliNQ could potentially suffer from the accumulation of prediction errors as time increases (as it intends to learn a "guess" from another "guess"). Although PSS is mostly distinct from the true transient performance, it can be used as a benchmark at each time step, and it can help *supervise* the prediction process by reducing the cumulative prediction errors. For example, as showed in Fig. 4, the waiting time trajectory approaches its steady state at a faster (slower) speed if its present value is further from (closer to) the steady state. Drawing from this, PSS can alleviate accumulated errors by either accelerating or decelerating the waiting time trajectories using the distance between the transient value and PSS value: If the distance is higher than expected due to past errors, the predicted trajectories will move faster toward their PSS targets, correcting the past errors to some extent. In this way, NeuraliNQ's prediction is able to remain robust in longer predictive time frames. Using the sampled parameters and their corresponding simulated data, our PSS neural network is effectively trained to provide accurate steady-state values under all parameters in the parameter space. Under all arrival rates $\lambda \in \mathcal{B}$, we plot the PSS values of the mean waiting time (bottom panel) and server occupancy (top panel) in Fig. 6. Unsurprisingly, both curves are monotonically increasing in λ. In a large scale queue (e.g., $n = 50$), the waiting time (occupancy) becomes a more informative performance descriptor when the system is overloaded (underloaded), while the occupancy is 100% (waiting time is 0). However, in a smaller scale example (e.g., $n = 1$), they describe the system state jointly.

Remark 2 (Fitting the non-smooth PSS curves) Although we train a neural network to fit the PSS curves, more conventional methods such as more straightforward curve-fitting methods (e.g. B-spline interpolation or regression algorithms) can apply. The choice of using a neural network is for studying the relative complexity of different PSS curves and understanding which neural network architecture can best fit these data, while achieving smoothness and preserving the non-smooth properties of these curves (e.g. the transition point to overfitting for large scale systems). Our TRF network architecture was greatly based on the best performing PSS network architecture, an indication that the inherent complexity to learn these two datasets is similar (similar degrees of noise and a combination of smooth and non-smooth characteristics). Beyond this research opportunity, one can easily utilize modern automated machine learning (AutoML) methods for automating the model selection and performing the subsequent hyper-parameter optimization for PSS.

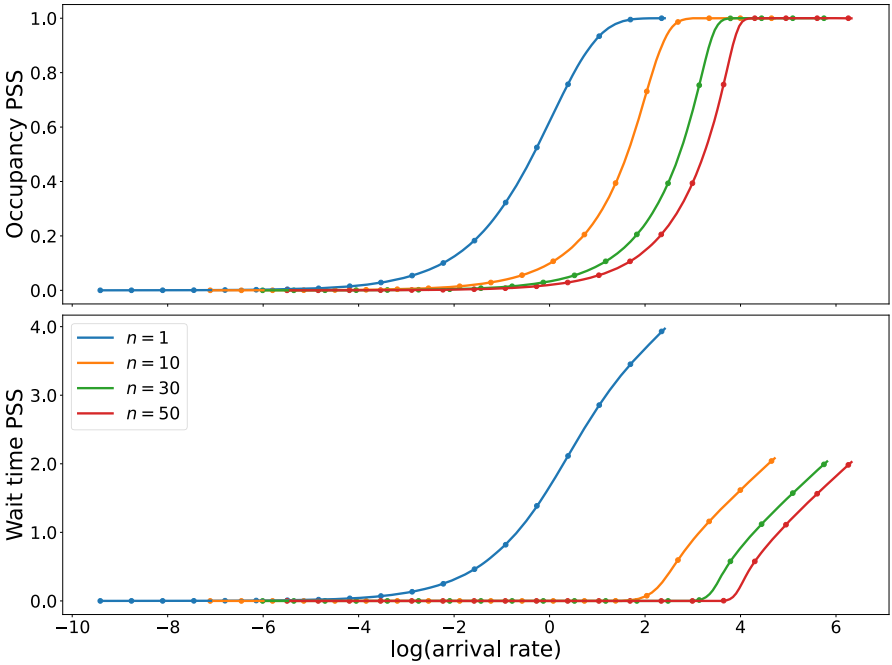


Fig. 6 The fitted PSS curves for server occupancy and waiting time at different system scale

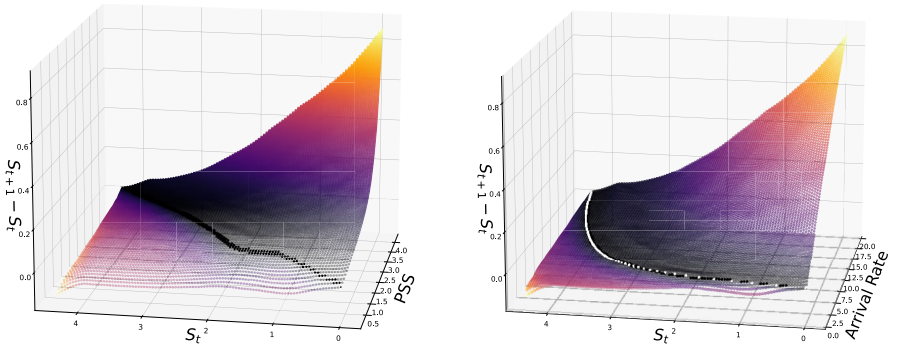
3.4 Transient response function

PSS learns the projected target for the performance functions under the assumption that the parameter $\boldsymbol{\beta}_{t+1}$ remains a constant after $t+1$. However, in practical (nonstationary) queueing systems, the system parameters (e.g., λ_t) often change rapidly, so there is not enough time for $\boldsymbol{S}_{t+1}$ to reach its PSS $\widehat{\boldsymbol{S}}_{t+1}^{\infty}$. In fact, at time $t+1$, the queueing trajectory may be on its way to reach $\widehat{\boldsymbol{S}}_{t+1}^{\infty}$, but it is not quite there yet. Hence, to predict $\boldsymbol{S}_{t+1}$, we need to quantify how far $\boldsymbol{S}_{t+1}$ is from its targeting PSS. This is handled by our *transient response function* (TRF), denoted by $\mathcal{G}^T$.

TRF is trained using the pair of data values at every two successive time steps $(t, t+1)$ on all simulated transient trajectories, such as the waiting time pairs at times (1, 2), (2, 3), (3, 4), etc. Taking inputs including $\boldsymbol{S}_t$ and PSS $\widehat{\boldsymbol{S}}_{t+1}$ (computed by the PSS neural network), TRF outputs $\boldsymbol{S}_{t+1}$.

Unlike PSS, TRF is a two-parameter function that maps the present state $\boldsymbol{S}_t$ and the PSS $\widehat{\boldsymbol{S}}_{t+1}^{\infty}$ to the predicted $\boldsymbol{S}_{t+1}$; see Fig. 7 for an illustration of a learned TRF curve of the waiting time.

The philosophy of TRF is to predict the process "derivative" $\Delta\boldsymbol{S}_{t+1} \equiv \boldsymbol{S}_{t+1} - \boldsymbol{S}_t$, the increment of the state from t to $t+1$ (z axis in Fig. 7), based on the present state $\boldsymbol{S}_t$ and PSS $\widehat{\boldsymbol{S}}_t^{\infty}$. When the present state $\boldsymbol{S}_t$ is close to $\widehat{\boldsymbol{S}}_t^{\infty}$, the system state remains unchanged from t to $t+1$ (this corresponds to the dotted curve in panel (a) that has 0 z values). When $\boldsymbol{S}_t < (>)\widehat{\boldsymbol{S}}_t^{\infty}$, a positive (negative) $\Delta\boldsymbol{S}_{t+1}$ is prescribed, which gives



(a) The trained TRF surface. (b) The trained prediction curve.

Fig. 7 Prediction of $\Delta \boldsymbol{S}_{t+1}$ as a function of the PSS (**a**) and the arrival rate (**b**) at t. The dotted lines correspond to the zero z values

the predicted $\boldsymbol{S}_{t+1} = \boldsymbol{S}_t + \Delta \boldsymbol{S}_{t+1}$. This explains why the dotted line corresponding to the dotted line in Fig. 7a is close to a 45-degree line.

Unlike PSS which responds to the changes in the arrival rate instantaneously, TRF properly balances the new arrival rate (via its PSS) and the previous system state, resulting in a more gradual and smooth transition from t to $t+1$. As depicted in Fig. 5, the fluctuations in the nonstationary trajectories of the waiting time and server occupancy are much less drastic than those in the PSS curves. With the PSS providing directional guidance, TRF is able to quantify the increment of the state process from t to $t+1$.

3.5 Neural networks

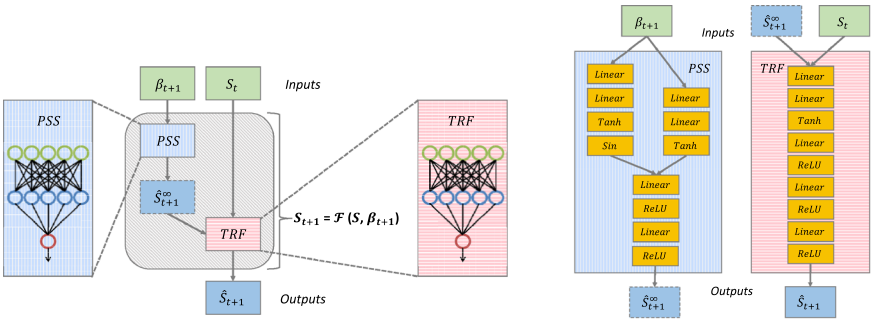
Using PSS and TRF, we obtain our predicted value of the next-step state as below:

$$\widehat{\boldsymbol{S}}^{\infty}_{t+1} \leftarrow \mathcal{G}^P\left(\boldsymbol{\beta}_{t+1}\right), \tag{3}$$

$$\widehat{\boldsymbol{S}}_{t+1} \leftarrow \mathcal{G}^T\left(\boldsymbol{S}_t, \widehat{\boldsymbol{S}}^{\infty}_{t+1}\right). \tag{4}$$

From the training data, we learn the two functions (3) and (4) using two feedforward neural networks; see panel (a) in Fig. 8 for the complete architecture of our stepwise lookahead prediction model. These two neural networks have similar architecture, consisting of alternating linear and nonlinear layers.

The motivation behind using two separate networks for PSS $\mathcal{G}^P$ and TRF $\mathcal{G}^T$ is twofold. On the one hand, separating these predictions offers better explainability on how the system learns the transient and stationary behavior. On the other hand, this helps to keep the cumulative prediction error under control in long prediction horizons. Because NeuraliNQ predicts the next-step system value based on the present value (which itself is also a prediction using past values), this recursion might potentially suffer from accumulated errors as time evolves. Nevertheless, the next-step PSS is independent of the past prediction error (see Equation (3)), so it can apply neces-



(a) Input-output diagram. (b) Neural network architecture.

Fig. 8 The neural network architecture

sary corrective adjustments to the prediction process to prevent the predicted system function from diverging from its true trajectory. See our later examples (e.g., Fig. 17) where we show that PSS plays a critical supervisor role.

3.6 Inference process

A key aspect of NeuraliNQ that requires clarification is how it should be used for inference after training. The inference process is outlined in Algorithm 1. During inference, NeuraliNQ requires the current system state (mean waiting time and server occupancy at time t) and the arrival rate at time $t+1$ as inputs. It then produces predictions for the system state at time $t+1$. Despite its recurrent nature, NeuraliNQ is designed to compute the mean transient performance over a finite time interval rather than giving real-time prediction along a particular sample path. For example, it may be used to calculate the mean waiting time for the course of tomorrow; but it is not designed to give a real-time prediction of the waiting time at $t+1$ using the length of the queue observed or the waiting time at t.

Algorithm 1 NeuraliNQ Inference Process.

Require: Initial system state $\boldsymbol{S}_0 = (\mathbb{E}[W(0)], \mathbb{E}[B(0)])$
Require: Arrival rate sequence $\lambda_1, \lambda_1, ..., \lambda_T$
Require: Trained PSS network $\mathcal{G}^P$ and TRF network $\mathcal{G}^T$
$t \leftarrow 0$
while $t < T$ **do**
  $\widehat{\boldsymbol{S}}^{\infty}_{t+1} \leftarrow \mathcal{G}^P(\lambda_{t+1})$ ▷ Compute PSS for next time step
  $\boldsymbol{S}_{t+1} \leftarrow \mathcal{G}^T(\boldsymbol{S}_t, \widehat{\boldsymbol{S}}^{\infty}_{t+1})$ ▷ Predict next system state
  $t \leftarrow t+1$
end while
return $\boldsymbol{S}_1, \boldsymbol{S}_2, ..., \boldsymbol{S}_T$

For empirical arrival models, we follow standard procedures [6, 8] to estimate the arrival rate sequence. To initiate the recurrent prediction depicted in Algorithm 1, NeuraliNQ requires the initial system state $\mathbf{S}_0$. We consider the following two cases:

- **Empty system.** Our primary use case is the prediction of the mean waiting time over a "day" that begins in an empty state, where the initial mean waiting time and occupancy are zero. This setup is particularly relevant for service systems that do not operate continuously over 24 h, such as call centers which typically start each day with no customers in the system.
- **Non-empty system.** The situation becomes more complex when the queueing system is not initially empty, although it remains possible to compute or estimate the initial state. For instance, consider a Markovian queueing system with m customers waiting in queue at time 0. In this case, the waiting time of a newly arriving customer follows a phase-type distribution: Suppose the service rate is μ, the abandonment rate is θ, and there are n servers. The new customer must wait for $m + 1$ departures (either through service completion or abandonment). Consequently, the customer's expected waiting time is the sum of the means of exponential random variables with rates $m\theta + n\mu$, $(m-1)\theta + n\mu, \ldots, n\mu$. For systems with non-exponential abandonment distributions, additional information is required to compute the initial state such as the ages (i.e., elapsed waiting times) of customers currently in the queue [2]. Nevertheless, we may resort to effective approximation methods such as [53] where the waiting time is approximately computed by a birth-death process with the death rate constructed using the abandonment hazard rate. Alternatively, MC simulation can always be used to estimate the mean waiting time at time 0; this simulation is expected to be quick and simple since only a small initial horizon needs to be simulated for generating the waiting time samples at time 0, given the starting condition of the system (i.e. simulate the time until all initial waiting customer either enter service or abandon).

4 Numerical experiments

In this section, we evaluate the effectiveness and robustness of NeuraliNQ. By effectiveness, we refer to the accuracy of NeuraliNQ in predicting queueing performance metrics compared to ground truth simulation results. By robustness, we refer to the method's ability to maintain this accuracy across various system configurations, including different scales, arrival patterns, and distributional characteristics. Given the data-driven approach and its inherent black-box nature of our method, it is difficult to provide theoretical guarantees of its performance. For this reason, we conduct a comprehensive set of numerical studies to evaluate the performance of NeuraliNQ. In Sect. 4.1, we describe the detailed settings of our numerical experiments. In Sect. 4.2, we apply NeuraliNQ to compute the mean waiting times in an $M_t/GI/n + GI$ base example; we test the robustness of NeuraliNQ under various model settings, including different system scales, arrival-rate patterns, low and high variability of service and abandonment times, and non-Poisson arrivals. In all these examples, we use the fluid approximation as a performance benchmark. In Sect. 4.3, we consider other predominant service-level metrics, including the mean queue length, probability of abandonment, and tail probability of delay. In Appendix 2 we offer additional analysis where we evaluate the generalization capabilities and limits of NeuraliNQ with vari-

Table 1 Experimental parameter space

Model components	Settings
Arrival processes	M_t (non-homogeneous Poisson)
	G_t (non-Poisson with hyperexponential interarrivals)
Service distributions	M (Exponential, SCV = 1)
	H_2 (Hyperexponential, SCV = 4)
	E_2 (Erlang-2, SCV = 0.5)
Abandonment distributions	M (Exponential), E_2 (Erlang-2)
Number of servers/system scale	$n = 1, 10, 30, 50$
Arrival rate range	$[0, 20n]$ where n is the system scale

ous nonstationary arrival patterns, both synthetic such as square and triangular waves, and one realistic arrival example taken from a contact center.

4.1 Experiment settings

4.1.1 Experimental parameter space

To clearly define the scope of our experiments, Table 1 summarizes the specific parameter ranges and distributions used in our study. In addition, we evaluate our models in different arrival structures, such as sinusoidal, square, and triangular waves. For the combinations of Table 1 parameters that we present in this section, we follow the entire method sequence of simulating, training, and testing separate neural networks. We train with pairs of $[t, t + 1]$, while we test recurrently, using past predictions as inputs for the future.

Throughout the paper, we hold the mean service time $1/\mu = 1$ and treat it as the time unit; all other temporal measures are expressed in units of the mean service time. We focus on phase-type distributions (exponential, Erlang, hyperexponential) which have finite moments and can approximate a wide range of distribution shapes. Our current implementation is specifically trained and validated for these distribution families, though the methodological approach could be extended to other distributions by generating the corresponding simulated training data.

4.1.2 Simulation and input sampling

Our model's training data are obtained via simulations and are split into two main categories, one for the PSS neural network and the other for the TRF network. In each category, we sample parameters of the arrival process within pre-specified bounds of interest (i.e., arrival volume bounds).

For every presented queueing system below, we sample the arrival process parameter $\widehat{\lambda}_1, \ldots, \widehat{\lambda}_k$ by drawing $k = 200$ samples in $[\underline{\lambda}, \bar{\lambda}] = \left[0, e^3\right]$. We simulate the steady-state waiting time and service occupancy under each arrival rate $\widehat{\lambda}_i$. To do so, we run the simulation until a large time T_p (here we set $T_p = 50$ time units). We note

here that because we run simulations in a high-compute system, we tune T_p empirically, and we confirm the system reached a steady state by detecting an unchanged moving average for $k = 20$ consecutive intervals.

In order to generate transient training data, we consider sinusoidal arrival functions in the form of $\lambda(t) = (a + b \cdot \sin(ct))$ where we sample the parameters a, b, c in proper local spaces. We select this arrival-rate pattern due to its following properties: a) It provides an easy way to span the predefined arrival process bounds, in this case $[0, a + b]$ for the volume bounds and $(0, c/2\pi]$ for the frequency bounds. These two bounds can be easily estimated for any local space by taking the maximum of all possible arrival volumes; also, any arrival space can be transformed into its spectral space via a Fourier transformation by taking the maximum harmonic. b) It provides a convenient way to generate arrivals that drives the system to transition between its underloaded and overloaded states. However, we note here that this arrival generation basis is limited to continuous and smooth arrivals only, without rapid changes between high and low volumes. We discuss the impact of this limitation and how to overcome it in Appendix 2 where we explore the generalization ability of our method beyond the proposed training arrival structures. We hereby consider $a \in (0, e^3/2]$, and $b, c \in (0, e^3]$, and uniformly draw 160 samples. Under each arrival rate sample, we simulate trajectories of waiting time and occupancy in the time window $[0, T]$ (we set $T = 20$ time units).

For each simulation scenario, we begin with an initially empty system and obtain the Monte Carlo estimated mean waiting time trajectory by averaging N_s independent simulation paths. We set $N_s = 50{,}000$ for small and median scale systems and $N_s = 10{,}000$ for higher scale systems (in all cases we control the standard error within 0.01 for every learned metric). We operate in a finite time interval $[0, T]$ and discretize time with step size $\Delta t \equiv 0.05$ (see Appendix 1 for a detailed analysis of this decision).

4.1.3 Structuring and training the PSS and TRF neural networks

We train the two networks separately: We first train PSS, and then we use the trained PSS neural network as an upstream module during the TRF training. Separating PSS and TRF as two individual neural networks can help improve the overall model explainability; this allows us to easily examine and measure the accuracy and limitations of each network, and to fine-tune their hyperparameters for improved performance.

The PSS neural network. Our PSS neural network consists of 11 layers in total and has a width of 500 neurons. We select convenient activation functions that are suitable for the steady-state waiting times curves, which are locally smooth and likely contain non-smooth neighborhoods around the point of critical loading (where the system has low waiting time and high service occupancy); see Fig. 6. We conduct extensive experiments to identify the neural network architecture that can successfully capture smooth and non-smooth sections of the PSS curve while avoiding overfitting the noisy data. These idiomatic characteristics of the data lead us to compose a combination of activation functions that strikes the right balance in distinguishing this data noise and the non-smooth curve properties, something that we discover is hard for a single type of activation function. For this reason, we separate the network into two branches, each branch designed to address each aspect by leveraging the properties of different

activation functions. As shown in panel (b) of Fig. 8, PSS consists of several common activation functions including linear, Tanh, sinusoidal, and ReLU. The common Tanh layer alone can capture smooth and nonlinear functions, but it fails to account for non-smooth characteristics. Therefore, inspired by [49], we include a periodic activation function (e.g., sinusoidal) on a parallel branch of the original Tanh sequential network. The output of these two parallel sequential networks is concatenated together and then passed through alternating affine and ReLU layers, with a final ReLU to ensure non-negative results.

The TRF neural network. Our TRF network has 9 layers and a width of 2000 neurons. (We increase the scale here to account for the higher complexity in TRF than PSS.) As shown in Fig. 8(b), we use alternating linear layers (with bias) and layers with smooth non-linear activation functions, including Tanh and ReLU. See Fig. 7 for an illustration of a learned TRF surface. Because the TRF surface is more smooth than the PSS curve, we omit the periodic activation function.

The training of the two networks can be performed sequentially for the ease of implementation and parameter fine-tuning. We present the sequential training settings below with the training hyperparameters tuned specifically for each network. The sizes of the training samples are 200 for PSS and 63840 for TRF. We use a training batch size of 50 samples, 5000 epochs, and a slow learning rate of 2×10^{-7} for training the PSS model. We use a batch size of 500, 2000 epochs for TRF, and a learning rate of 10^{-7}. For both networks, we use the Adam stochastic optimization [27] and an L1 loss function (i.e., the mean absolute error). The total epochs for PSS are 30,000, and for TRF, are 3000.

The motivation behind the above training volume choices is primarily from the domain knowledge of the data characteristics and their properties. Specifically, the PSS training data need to be dense to accurately characterize the transition point between the underloaded and overloaded modes, a point that is not necessarily known during the problem space parametric sampling. We follow the same motivation when we sample the transient space using a similar order of magnitude of parametric samples. Finally, we use the empirical insights from [25] on the minimum number of network parameters conditional on the size of the training data for avoiding overfitting, while we use very large numbers of epochs (inversely proportional to the training data volume) combined with very small learning rates to avoid data regularization and early stopping (we consider a slower network learning a simpler method to avoid overfitting in this type of data).

4.1.4 Simulation details

In our simulations, we consider the system to have reached steady state when the moving average of the waiting time changes by less than 0.1% over 20 consecutive time units. Systems with customer abandonment are guaranteed to reach their steady states because abandonment prevents unlimited queue buildup.

We conducted extensive convergence experiments to determine appropriate simulation times. For the systems we present here, we found that 50 time units are sufficient to reach steady state, even for low arrival rates. This is because the abandonment mechanism prevents queue buildup and facilitates faster convergence. Even for arrival rates

below 20 (with a mean service time of 1), our convergence criteria were met within this timeframe. For our transient simulations with 20 time units, we verified that this horizon is sufficient to capture the essential system dynamics where the system transitions from a cold start to a dynamic (periodic) steady state. In most cases, systems exhibited complete transitions between underloaded and overloaded states multiple times within this window, providing rich training data for our neural networks.

4.1.5 Computational resources

To generate the training data[5], we run Monte Carlo simulations on a host with 96 CPU cores. Our total simulation effort represents approximately 9000 CPU hours distributed across our computing cluster with 96 Intel Xeon E5-2686 v4 @ 2.30GHz CPU cores, resulting in a wall-clock time of approximately four days. This substantial computational investment was necessary to generate sufficient high-quality training data across the parameter space. It's important to acknowledge that the total computational cost of NeuraliNQ includes both the significant upfront investment in simulation data generation and model training, as well as the minimal inference cost. The data generation process required approximately 9,000 CPU hours in our experiments, and the model training took an additional 15 min on a single GPU. Once the training is completed, NeuraliNQ's prediction time is approximately 350 milliseconds on a single Intel Xeon E5-2686 v4 @ 2.30GHz core (200 milliseconds on a Tesla V100 SXM2 16GB GPU), reducing computational cost by several orders of magnitude.

This computational tradeoff makes NeuraliNQ particularly valuable for scenarios where: (i) multiple analyses of the same queueing system with different arrival patterns are required, (ii) the service and abandonment distributions remain fixed while arrival patterns change, (iii) fast predictions are needed for real-time decision support, and (iv) the system needs to be analyzed repeatedly over time. In such cases, the significant upfront computational investment is amortized over many subsequent fast predictions.

4.2 Performance evaluation

We evaluate the solution accuracy of NeuraliNQ by benchmarking with the Monte-Carlo simulated results, which we refer to as the *ground truths*. We start below with the analysis of the intermediate PSS network, and we then expand to the end-to-end performance of our method.

4.2.1 PSS network performance analysis

The PSS network serves as a foundation for NeuraliNQ's overall performance prediction capability. Before examining the end-to-end performance of NeuraliNQ, it is important to evaluate the accuracy of the PSS component independently, as it plays a critical supervisory role in the prediction process.

[5] To foster more research in the queueing community on this type of work, we have open sourced the training data of the $M_t/H_2/n + E_2$ system at this Hugging Face repository.

Table 2 PSS accuracy across different system configurations

Experiment in Sect. 4	Test MAE
Table 3, scale $n = 1$	5.5e−4
Table 3, scale $n = 10$	3.9e−4
Table 3, scale $n = 30$	6.7e−4
Table 3, scale $n = 50$	1.1e−4
Table 7, SCV $c_S^2 = 0.5$	5.2e−4
Table 7, SCV $c_S^2 = 1$	5.5e−4
Table 7, SCV $c_S^2 = 4$	5.4e−4

We evaluate the performance of all trained PSS networks against a 10% test (held-out) subset of the training data ($m = 20$ samples). To properly evaluate the fitted curve across different operational regions (both underloaded and overloaded), we evenly distribute the test samples by selecting every tenth sample from our $\{\widehat{\lambda}_1, \ldots, \widehat{\lambda}_k\}$ basis (these test samples are shown in Fig. 6). We use mean absolute error (MAE) as our evaluation metric:

$$\mathcal{E}^{\text{PSS}} = \frac{\sum_{i=1}^{m} |y_i - x_i|}{m} = \frac{\sum_{i=1}^{m} |e_i|}{m}.$$

While we are generally interested in the end-to-end performance of our method, for completeness, Table 2 presents the consolidated (partial) PSS accuracy results for all the queueing systems reported in our end-to-end performance analysis experiments of the following Sections. The consistently small residual errors are primarily due to the inherent noise in estimating stationary points from simulation. Increasing the number of simulated paths would further reduce this error.

These results demonstrate that our PSS network accurately captures the steady-state behavior across various system configurations, providing a reliable supervisory signal for the TRF component. The PSS network could also be used independently for steady-state analysis of queueing systems, representing a valuable contribution in its own right.

4.2.2 End-to-end performance analysis

We extend the $M_t/H_2/n + E_2$ example defined in Sect. 1.4 having nonhomogeneous Poisson arrivals, I.I.D. H_2 service times, and I.I.D. E_2 abandonment times. To quantify the solution fidelity, we compute the *time-averaged mean squared error* (TAMSE). Specifically, we define the TAMSEs of NeuraliNQ $\mathcal{E}^D$ and the fluid model $\mathcal{E}^F$ as:

$$\mathcal{E}^D \equiv \frac{1}{T}\sum_{t=1}^{T}\left(e_t^D\right)^2 \quad \text{and} \quad \mathcal{E}^F \equiv \frac{1}{T}\sum_{t=1}^{T}\left(e_t^F\right)^2, \tag{5}$$

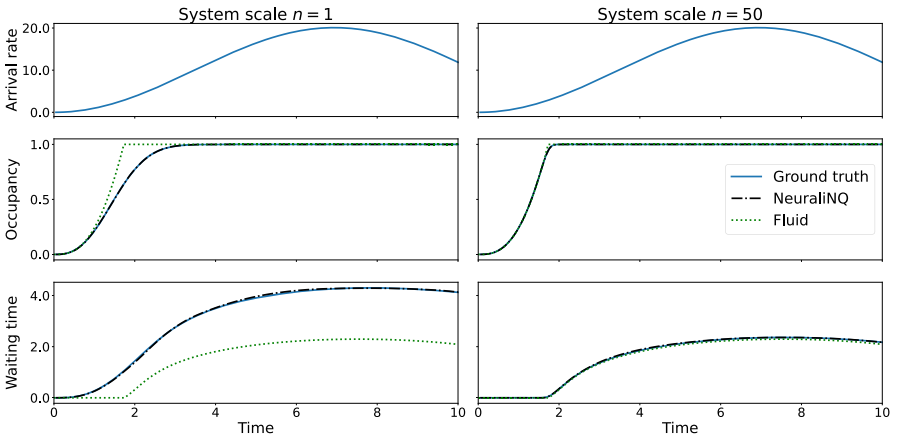


Fig. 9 NeuraliNQ vs. fluid with a sinusoidal arrival with $c = 0.022$ for various scales

where e_t^D and e_t^F are the prediction errors (i.e., the simple sum of the squared difference of the predicted value and the ground truth metrics) of NeuraliNQ and the fluid model at time t.

Unlike the fluid model approximation, the performance of NeuraliNQ should be robust to the system scale. To validate this point, we scale our base $M_t/H_2/n + E_2$ model introduced in Sect. 1.4 by multiplying the arrival rate and the staffing level with a factor n (referred to as the *system scale*). Specifically, we consider the arrival rate

$$\lambda_n(t) = n(1 + r \cdot \sin(ct)). \tag{6}$$

Here, besides the scale n which controls the system's size, the relative amplitude r and the frequency parameter c account for how the arrival process fluctuates in space and time. We also examine other arrival rate functions later.

4.2.3 On the system scale *n*

In Table 3, we report the recurrent average TAMSEs for NeuraliNQ and the fluid model with the system scale $n = 1, 10, 30, 50$ in the interval $[0, T]$, $T = 10$, for all different combinations of $r = 0.05, 0.3, 0.5, 0.7, 0.9$ and $c = \{0.144, 0.372, 0.961, 2.479\}\pi$. Unlike the fluid approximation, where the accuracy degrades as n decreases, the TAMSE of NeuraliNQ is robust to n. In particular, when the system has a smaller scale, NeuraliNQ outperforms the fluid approximation by yielding a smaller TAMSE (e.g., $n = 1, 10$). When the system scale is relatively larger (e.g., $n = 50$), both methods are effective, showcasing similar TAMSE values. Also see Fig. 9 for the performance curves at two system scales.

4.2.4 On the frequency *c* and relative amplitude *r*

We next test the robustness of NeuraliNQ to the frequency parameter c. In Table 4, we report the TAMSEs for $c = \{0.144, 0.372, 0.961, 2.479\}\pi$ (each time we calculate

Table 3 Fluid model vs. NeuraliNQ: TAMSE under different system scale n

System scale n	1	10	30	50
Fluid	2.73e+0	9.84e−3	3.47e−3	2.73e−3
NeuraliNQ	**2.23e–3**	**2.93e–4**	**3.82e–4**	**1.09e–3**

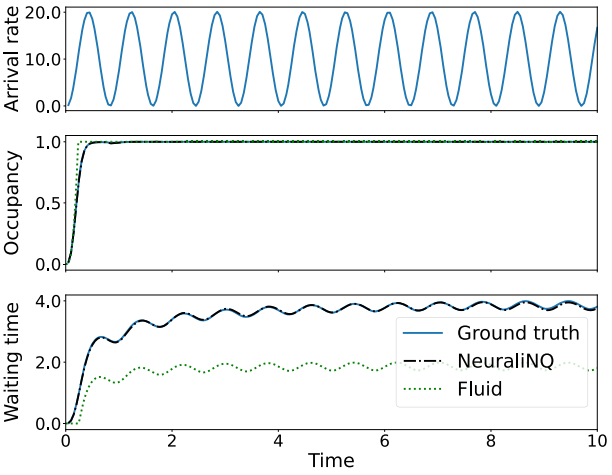


Fig. 10 NeuraliNQ vs. fluid: A sinusoidal arrival rate with $c = 2.479\pi$

Table 4 Fluid model vs. NeuraliNQ: TAMSE under different frequencies

Frequency c/π	0.144	0.372	0.961	2.479
Fluid	2.73e+0	3.03e+0	3.17e+0	3.21e+0
NeuraliNQ	**2.23e–3**	**3.49e–3**	**6.90e–3**	**3.05e–3**

the average for every different amplitude $r = 0.05, 0.3, 0.5, 0.7, 0.9$ at scale $n = 1$). Table 4 shows that NeuraliNQ remains effective and robust in c. Also see Fig. 10 for the performance of NeuraliNQ under a frequently varying arrival rate.

We also evaluate the performance NeuraliNQ under different relative amplitude r of the arrival process (6). In Table 5, we report the TAMSEs for $r = 0.05, 0.3, 0.5, 0.7, 0.9$ (each time we calculate the average for every different frequency $c = \{0.144, 0.372, 0.961, 2.479\}\pi$ at scale $n = 1$). Table 5 shows that NeuraliNQ remains effective and robust in r, consistently yielding more accurate results with TAMSEs three orders of magnitude smaller than those of the fluid approximation.

4.2.5 Transitions between underloaded and overloaded intervals

Nonstationary queueing systems are already complex and difficult to treat, and the analysis becomes even more involved when the system constantly alternates between *underloaded* and *overloaded* intervals, where in the former scenario the system has

Table 5 Fluid model vs. NeuralinQ: TAMSEs under different relative amplitudes

Rel. amplitude r	0.05	0.3	0.5	0.7	0.9
Fluid	1.62e+0	1.86e+0	2.08e+0	2.30e+0	2.58e+0
NeuralinQ	**5.36e–3**	**6.65e–3**	**6.22e–3**	**4.96e–3**	**3.03e–3**

Table 6 Fluid model vs. NeuralinQ: During alternating overloaded and underloaded intervals

System scale n	1	50
Fluid	1.93e+0	1.30e−3
NeuralinQ	**7.08e–4**	**6.52e–4**

smaller waiting times and some idle servers, while in the latter case the system exhibits longer waiting times and no idle servers. In the extant queueing theory literature, these two cases are often analyzed separately. For example, when the system scale is large, an underloaded system is operating in the so-called *quality-driven* (QD) regime and is asymptotically equivalent to the corresponding infinite-server queue; an overloaded system is operating in the *efficiency-driven* (ED) regime which can be reformulated as another age-truncated infinite-server queue having "service" times corresponding to customers' abandonment times [39]. A useful way to treat a queueing system that constantly switches between underloaded and overloaded intervals is via the following steps: (i) identify the successive overloaded and underloaded intervals; (ii) characterize the model dynamics within each interval; and (iii) piece these interval-specific results together alternately until the entire time horizon is covered [37].

This strategy often is effective for the interior of an underloaded (overloaded) interval, but the solution fidelity usually degrades around the transition time points where the system is going through the *critically loaded* state to move from underloading (overloading) to overloading (underloading). The main issue is that these transition times, treated as deterministic times in the above method, are in fact random variables, so that the system evolution is usually quite smooth between two intervals, while the above-mentioned approximation often gives non-smooth performance curves around these time points; see numerical results in [37]. NeuralinQ, on the other hand, is insensitive to the system's workload, so it remains effective regardless of the system's congestion level (overloaded, underloaded, or critically loaded). According to Fig. 3, NeuralinQ's prediction is able to smoothly cope with the performance functions even when the system constantly switches between underloaded and overloaded intervals. Also see Table 6 for consistently good performance of NeuralinQ under both system scales (with NeuralinQ yielding a lower TAMSE).

4.2.6 Stochastic variabilities in service times

We investigate the impact of the service distribution on the effectiveness of NeuralinQ. To partially describe the service-time distribution beyond the mean, we use the *squared coefficient of variation* SCV $c_S^2 \equiv \text{Var}(S)/(\mathbb{E}[S]^2)$ of the service time S. We consider three $M_t/GI/n + E_2$ queueing systems that are distinct in only the service-time

Table 7 Fluid model vs. NeuralinQ: TAMSEs under different service CSVs

SCV c_S^2	0.5	1	4
Fluid	1.61e−1	3.36e−1	2.73e+0
NeuralinQ	**2.09e–4**	**2.39e–4**	**2.23e–3**

distribution (having the same mean service time $1/\mu = 1$). Specifically, we consider (i) H_2 service with $c_s^2 = 4$, (ii) M service with $c_s^2 = 1$, and (iii) E_2 service with $c_s^2 = 1/2$. For each system, we follow our method as described in Sect. 4.1.2 for generating the training data, and we train the corresponding RNNs using the same hyperparameters of Sect. 4.1.3.

Table 7 confirms NeuralinQ's effectiveness; it exhibits TAMSEs that are three orders of magnitude smaller than that of the fluid model. In this table, we report the average TAMSAE for every combination of the parameters $c = \{0.144, 0.372, 0.961, 2.479\}\pi$, $r = 0.05, 0.3, 0.5, 0.7, 0.9$ and $n = 1$.

4.2.7 Non-Poisson arrivals

We next consider the fully non-Markovian setting featuring a non-Poisson arrival process G_t. We follow the combined inversion-and-thinning algorithm (CIATA) [31] for modeling a nonstationary non-Poisson process (NNPP). Specifically, we consider the $G_t/H_2/n + E_2$ model, which is our base example with its M_t arrival replaced by a G_t arrival constructed using H_2 distributed interarrival seeds. CIATA follows three steps: First, we build a piecewise constant majorizing arrival rates; second, we generate successive equilibrium renewal processes having the majorizing rate and desired H_2 variability parameter; and finally, we conduct an acceptance-rejection process to determine the actual arrival count using the acceptance probability defined as the real arrival rate divided by its majorizing value. See [31] for details. Distinct from an M_t arrival process where the *dispersion ratio* (i.e., $D(t) \equiv \text{Var}(N(t))/\mathbb{E}[N(t)]$) is 1, this G_t arrival process is *overly dispersed*, exhibiting a nonstationary dispersion ratio exceeding 1 (Panel (b) in Fig. 11). See [31] and references therein for empirical evidence of overdispersion in arrival data and other practical motivations of NNPP.

According to Fig. 11, NeuralinQ maintains its effectiveness for G_t arrivals, consistently outperforming the fluid approach. We note here that $D(t)$ is by nature more stochastic and even with 50.000 simulation Monte Carlo paths appears very noisy. In contrast, the solution accuracy of the fluid approximation degrades in the G_t case because it is unable to capture the distributional information beyond the mean values (i.e., arrival rate) of the arrival process.

4.3 Other performance metrics

So far, our numerical experiments have been focusing on the computation of the mean waiting times. However, several other service-level metrics are of practical relevance in modern service systems including: (i) the *queue length*, (ii) the *probability of abandonment*, which is the probability a customer abandons from the waiting queue (i.e.,

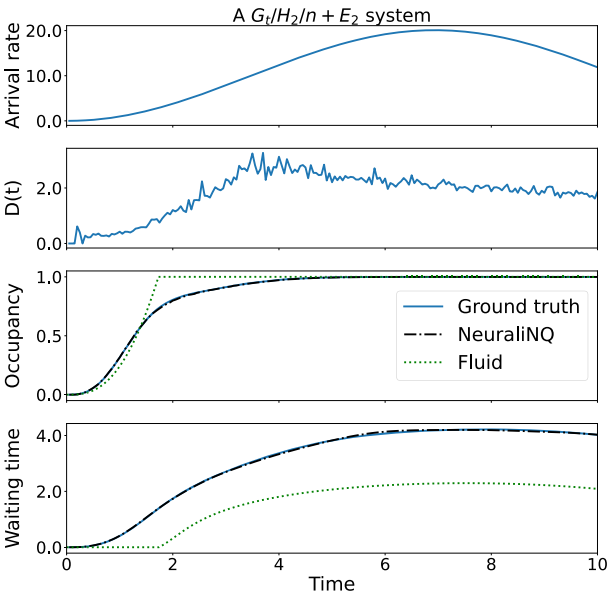


Fig. 11 NeuraliNQ vs. fluid approximation: The $G_t/H_2/n + E_2$ model having NNPP arrivals

$\mathbb{P}(W(t) > A)$ with A being a generic customer abandonment time), (iii) the *probability of delay*, which is the probability a customer experiences a positive waiting time (i.e., $\mathbb{P}(W(t) > 0)$), and (iv) the *TPoD*, which is the probability a customer's waiting time exceeds some target $\tau > 0$ (i.e., $\mathbb{P}(W(t) > \tau)$). See [32] for a review of these service-level metrics and their applications in service systems. In the extant literature, heavy-traffic methods are developed to treat these metrics. However, the methodologies for analyzing and controlling a queueing system with the goal of achieving desired service-level targets differ in the choice of the above-mentioned metrics, and because the mathematical properties of the asymptotic heavy-traffic limits are different. For example, achieving a desired probability of delay target asymptotically puts the system in the *quality-and-efficiency* (QED) regime, whereas controlling the TPoD sets the model in the *efficiency-driven* (ED) regime.

In contrast, NeuraliNQ treats these above-mentioned service-level metrics in nearly identical ways. Our approach is to slightly modify the structure of NeuraliNQ by including the desired service-level metric in its state. For example, to compute the TPoD $\mathbb{P}(W_t > \tau)$ for a given delay target $\tau > 0$, we augment the state to $\boldsymbol{S}_t = (\mathbb{E}[W_t], \mathbb{E}[B_t], \mathbb{P}(W_t > \tau))$. The remaining settings of the neural networks remain the same as the base model.

We consider our base $M_t/H_2/n + E_2$ example. In Fig. 12, besides the mean waiting time and occupancy, we report NeuraliNQ's predictions for the mean queue length, probability of abandonment, and TPoD. Here the waiting time target is $\tau = 1$ for a high-scale system (right-hand panels) and is $\tau = 2$ for a low-scale system (left-hand panels). Consistent with earlier results, we see high predictive accuracy in both examples. These examples show that the effectiveness of NeuraliNQ is robust to the

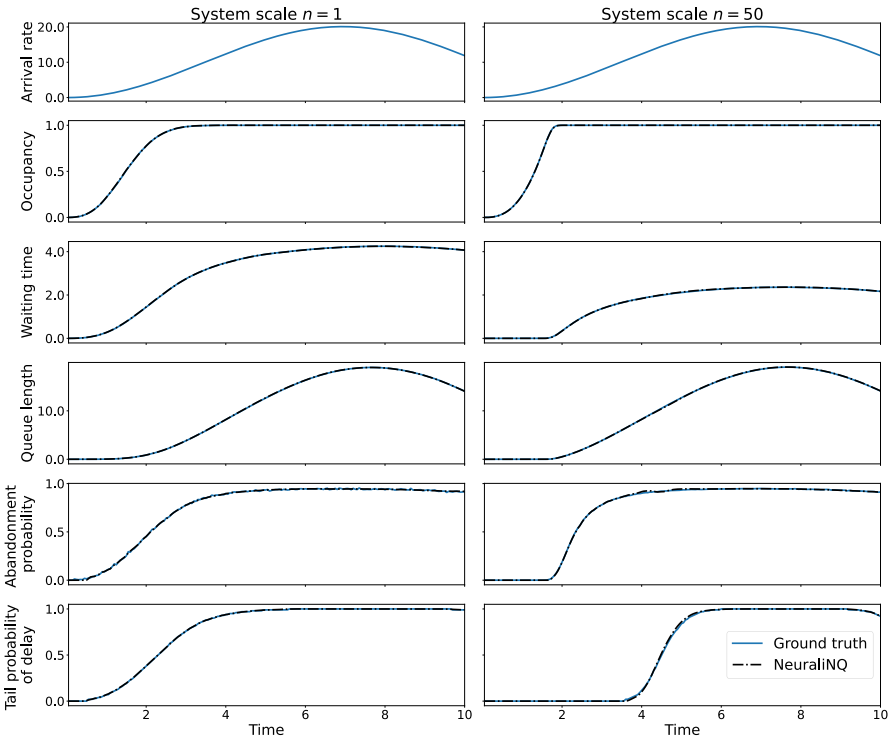


Fig. 12 NeuraliNQ vs. simulations: predictions of other performance metrics including the mean queue length, probability of abandonment, and TPoD with $\tau = 2$ for the small-scale system and $\tau = 1$ for the large-scale system

choices of the service-level metrics. In addition, NeuraliNQ is able to treat the more complex metric, such as TPoD, that draws from the distributional information above the mean waiting time, metrics that the fluid approximation cannot produce. Also, the treatment of TPoD is conceptually more straightforward in NeuraliNQ, while the heavy-traffic analysis on TPoD is quite involved and only available in certain cases (for example, TPoD is intractable in queues having G service times). We note here that the inclusion of additional metrics would eventually require additional hyper-parameter tuning of the neural networks, as they will need to learn more information (e.g., additional PSS and TRF). Although our preliminary studies show promising results, we have not carefully investigated how a more comprehensive neural network can be built to efficiently cover all desired performance metrics under minimum training efforts. This will be one of our future directions. See Appendix 2 for additional evaluation analysis on various nonstationary arrival patterns, both synthetic such as square and triangular waves, and one realistic arrival example taken from a contact center.

5 Discussions

In this section, we give in-depth discussions to generate additional insights. We first explain why the snapshot information appears to be sufficient in the stepwise look-ahead prediction process. Next, we explore NeuralinQ's distinct advantages when compared to the other two approaches prevalent in the existing literature: simulation and heavy-traffic approximation. In Appendix 3, we offer valuable discussions on the impact of including earlier system state information.

5.1 NeuralinQ vs. computer simulation

Simulation is frequently the preferred method when dealing with complex queueing systems due to its ability to produce performance analysis results with limited to zero analytical efforts. With full knowledge of the system parameters and sufficient computational resources, simulation is guaranteed to yield very accurate results. Nevertheless, some advantages are associated with our proposed approach compared to simulation.

First, NeuralinQ exhibits a significantly superior advantage in runtime when compared to simulation. As the previous section shows, Monte-Carlo simulation for the mean waiting time requires 3–4 orders of magnitude more time than NeuralinQ to produce high-quality performance results. Although the training of NeuralinQ builds on simulated queueing data, NeuralinQ is able to "remember" all training data under different model settings (e.g., arrival patterns). On the other hand, simulation is in general a "brute force" approach which does not recycle results from other simulation runs. In this sense, NeuralinQ is a more efficient approach.

Next, NeuralinQ is conceptually simpler and exhibits greater ease of implementation when compared to simulation. To estimate the mean waiting times, Monte-Carlo simulation necessitates the generation of many independent sample paths, each requiring the execution of discrete event simulation which operates at the level of all detailed elements within a queueing system, such as the exact moment of the next arrival time, and the remaining abandonment and service times, etc. In addition, to advance the simulation clock on each sample path, the system needs to memorize all of the above information, necessitating stringent requirements on data structure and storage. In contrast, NeuralinQ operates directly on the abstraction level of the end outcome (e.g., mean waiting time), something that allows us to produce accurate predictions without this level of state details, but rather only one time-frame's history. This, combined with the low prediction latency, makes our method applicable for near real-time predictions upon receiving the most recent state observations.

5.2 NeuralinQ vs. heavy-traffic methods

First, fluid model yields high fidelity results when the system scale is high, but its approximation begins to degrade as the system scale decreases. This is common to all heavy-traffic approximations because they arise from the limiting results derived as

the scale approaches infinity. On the contrary, NeuraliNQ, a scale-free method, is able to produce accurate predictions for systems at both small and large scales.

Next, heavy-traffic methods rely on model inputs spanning the entire time horizon (e.g., the global arrival-rate function over the entire $[0, T]$ interval), whereas the stepwise lookahead nature of NeuraliNQ requires only local model information at each prediction time (e.g., arrival rate at the next time step). The latter exhibits two advantages: First, the stepwise structure can largely reduce the dimensionality of the input variables of the neural network, thus reducing the scale of the network and lowering the training efforts (at each time t, NeuraliNQ relies only on the local arrival rate at t). Second, as the future arrival forecast tends to become less accurate as the future prediction horizon increases, it becomes unnecessary to immediately require an accurate arrival forecast at further future steps. In real-time operations, more accurate demand forecast for time t usually becomes available as we approach t, so that NeuraliNQ is able to take into account such timely updates in order to produce more accurate results.

Last, analytic approaches (exact or heavy-traffic analysis) are developed under specific model settings. For example, the investigations of Markovian models and non-Markovian models draw from completely different methodologies, and controlling different service-level metrics give rise to distinct asymptotic regimes (for example, setting a nontrivial target for the probability of delay requires the system to operate in the heavy-traffic QED regime, while a positive waiting time target puts the system in the ED regime in presence of customer abandonment). Hence, changing the model setting or performance target often requires the development of new analytic procedures and methodologies. In this sense, NeuraliNQ is a much more robust approach. As illustrated earlier, NeuraliNQ is less sensitive to various model settings and gives a unified treatment to several common service-level metrics.

5.3 Limitations

The present version of NeuraliNQ treats the $G_t/GI/n + GI$ queueing system having a constant staffing level. This setting is reasonable for service systems having inflexible staffing with fixed staffing intervals. For example, in hospitals, the staffing level remains unchanged for several hours. In call centers, the staffing level varies once every 30 min or 1 h; this staffing interval may still be considered sufficiently long compared to the much shorter service times (in minutes). Nevertheless, we are interested in developing a more general version of NeuraliNQ to capture the nonstationary staffing function. Next, NeuraliNQ focuses on handling general nonstationary arrival patterns while assuming fixed service and abandonment distributions. In future work, we aim to extend NeuraliNQ to accommodate more flexible service and abandonment distributions. One approach is to incorporate parametric phase-type distributions and effectively explore and sample from this family of distributions [4]. We are also interested in exploring a more efficient training data generation via computer simulation that can allow incorporating additional system input parameters while keeping the simulation computational effort low. We are motivated by the good accuracy of our

method and we are interested in researching how to balance the amount of training data and the achieved accuracy.

6 Conclusions

In this research, we propose NeuraliNQ, a neural network approach for studying the transient performance of a non-Markovian nonstationary queueing system. At the heart of NeuraliNQ is a stepwise lookahead recursion that determines the system state at time $t + 1$ based on the model input at t and the previous state at t. NeuraliNQ is comprised of two important modules, each implemented via a neural network: (i) the pointwise steady state and (ii) the transient response function. The former serves as a supervisor, which points the prediction in the right direction and helps reduce cumulative prediction errors, while the latter makes quantitative adjustments on the transition from t to $t + 1$. Numerical experiments confirm the effectiveness of NeuraliNQ and exhibit its advantage over the commonly used heavy-traffic approximation; they also show that NeuraliNQ is robust to several important factors, such as the system scale, arrival pattern, and service-time variability.

There are several venues for future research. While the present iteration of NeuraliNQ focuses on predicting the transient trajectory of the mean waiting time, this new paradigm can be used to study other practical and more complex service-level metrics (e.g., the TPoD which draws from the distribution of the waiting times beyond the mean). Preliminary studies in Sect. 4.3 reveal promising results in this direction. We plan to develop a more comprehensive version of NeuraliNQ capable of effectively addressing all wait-time service-level metrics. Second, while the current framework relies on training data generated under fixed service and abandonment time distributions, we aim to extend NeuraliNQ to accommodate more general settings. Specifically, we plan to train the neural network on data generated from a broader range of arrival-service-abandonment combinations, thereby enhancing its flexibility and applicability. Third, to address queueing systems without customer abandonment where the PSS is no longer well-defined, we plan to develop a new model architecture that does not rely on PSS as a supervisory signal. Next, the observed similarity between the stationary and transient characteristic dynamics under different system primitives motivates us to investigate *transfer learning* methods [56]. This involves adapting models trained for specific queueing systems to other systems. The concept is to rapidly construct a neural network for a new system (e.g., a non-Markovian system) using only a small set of its training data to fine-tune a base neural network of a similar system (e.g., a Markovian model). The former captures the *unique* features specific to the new model, while the latter retains the *common* characteristics shared between the two systems. Another important direction is to evolve NeuraliNQ from a performance prediction tool to a decision support tool by developing learning-based real-time staffing adjustment rules in order to achieve stable and satisfactory service-level targets. One possibility is to integrate NeuraliNQ with *generative adversarial networks* (GANs) [19] to invert a trained neural network model into an optimal staffing model.

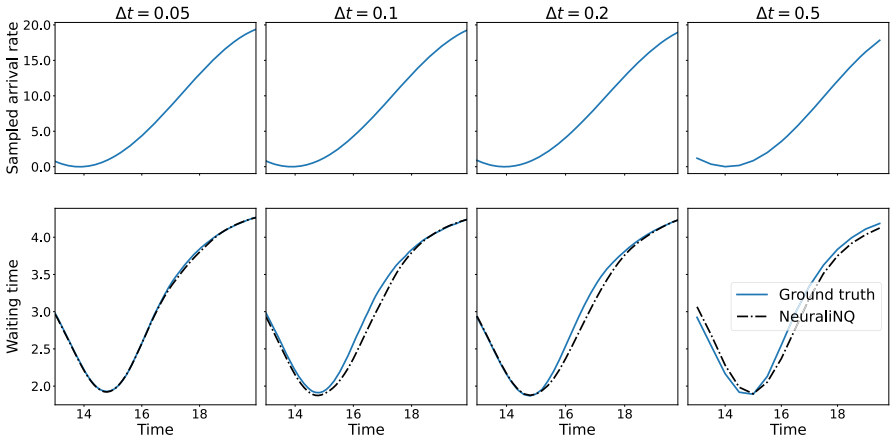


Fig. 13 NeuralNQ performance for the $M_t/H_2/n + E_2$ system with $c = 0.45$, under different time discretization values $\Delta t = 0.05, 0.1, 0.2, 0.5$

Acknowledgements We thank the editor Harsha Honnappa for providing many constructive comments! This work is part of the I+D+i project titled BLOSSOMS, grant PID2024-158530OB-I00, funded by MICIU/AEI/10.13039/501100011033/ and by ERDF/EU.

Appendix

A On the time discretization for NeuralNQ

Although we use t and $t + 1$ to denote the two time steps, they are not the actual time points. The difference between the two steps is Δt. In this section, we investigate how the value of Δt impacts NeuralNQ's performance. The time discretization step Δt should be chosen based on a trade-off between the solution accuracy and computation needed to simulate, train, and use our neural networks. First, the choice of Δt affects the sample size of the Monte Carlo simulation: We divide a simulation horizon of $[0, T]$ by Δt, which leads to $T/\Delta t$ pairs of t and $t+1$ state samples, and as Δt increases, the total training data decrease, requiring additional Monte Carlo simulations to generate the equivalent training data volume. Next, bigger Δt values can lead to loss of information in the arrival patterns, especially when the arrival-rate function is fast-varying. In Figures 13 and 14, we show the performance of four different NeuralNQ's, each under a specific Δt. We can see that when the true arrival rate is fast-varying and the discretization step Δt is big (e.g., $\Delta t = 0.5$), the sampled arrival rate can be quite rough (see top panels in Figure 14).

As a result, the prediction quality degrades (see bottom panels in Figure 14). On the other hand, such an effect is less evident if the true arrival rate is slowly varying (as illustrated in Figure 13).

Besides the loss of information from this under-sampling, an additional reason for the reduction of the accuracy is due to the inaccurate representation of the input latest state $\boldsymbol{S}_t$, as it is calculated by the average of longer intervals and can deviate from the

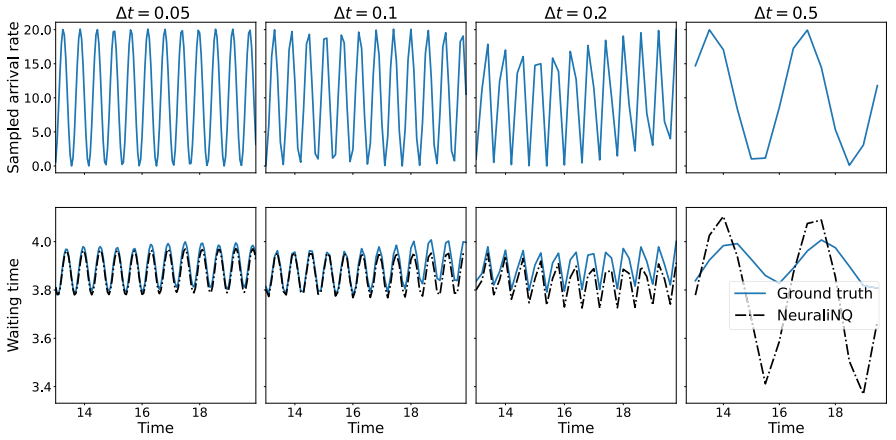


Fig. 14 NeuraliNQ performance for the $M_t/H_2/n + E_2$ system with $c = 10.68$, under different time discretization values $\Delta t = 0.05, 0.1, 0.2, 0.5$

true latest state more. On the other hand, an extremely short Δt can lead to compute limitations, as higher memory and computations are required in simulation and model training. We experiment with various Δt values and we find that a sampling rate of approximately three to four times the maximum frequency ($\Delta t = 0.05$ time units) strikes a good balance between accuracy and computation complexity in the presented results and the selected problem space (the selected maximum domain frequency is $c = e^3 \approx 20$).

B Analysis on Other Arrival Patterns

In the main body of our analysis, most performance evaluations of NeuraliNQ were conducted under sinusoidal arrivals, with the model trained using data generated from sinusoidal arrival rates. This might raise the question of NeuraliNQ's ability to generalize to other arrival patterns. In this subsection, we apply the NeuraliNQ model—trained on sinusoidal data—to predict waiting times for models with non-sinusoidal (out-of-sample) arrival patterns. In Appendix 1, we consider an arrival function arising from realistic call-center data. In Appendix 2, we evaluate NeuraliNQ's performance on various non-sinusoidal, non-smooth, and discontinuous patterns, including trapezoidal, triangular, square, and sawtooth. Finally, in Appendix 3, we discuss potential strategies for fine-tuning the model to handle these new patterns when needed.

B.1 More realistic arrival rate

Next, we evaluate the performance of NeuraliNQ under a more realistic arrival pattern. Specifically, we consider our base example having the arrival rate estimated from the arrival data in an Israeli call center [6]; see the top panel of Figure 15. We resample the arrival rate for hourly time units while we assume scale $n = 50$ to better emphasize

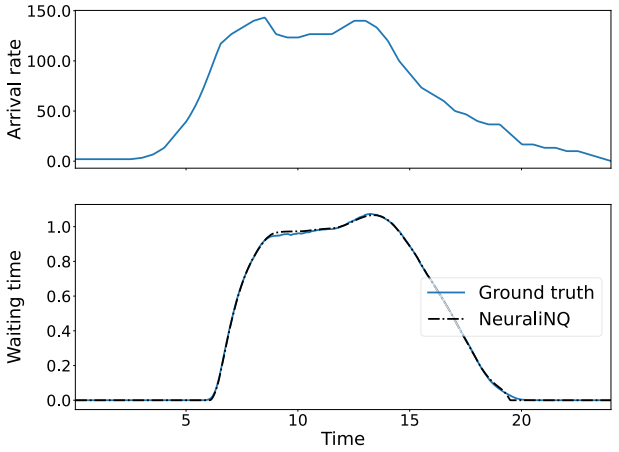


Fig. 15 Fluid model vs. NeuraliNQ: A realistic call-center arrival rate

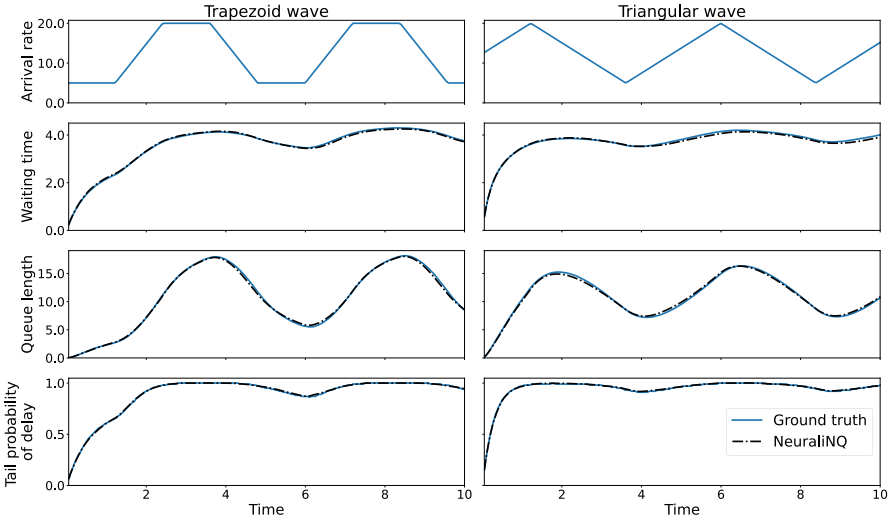


Fig. 16 NeuraliNQ vs. simulations for trapezoid and triangular arrival rates

the transition between overloaded and underloaded periods. Finally, to achieve high degrees of resolution, we smooth the original piecewise stationary arrival rates by performing a simple linear interpolation.

As seen in Figure 15, NeuraliNQ maintains its effectiveness by achieving highly accurate waiting-time predictions. The TAMSE of this example is 4.9×10^{-5}. In this example, we used the $M_t/H_2/n+E_2$ model with scale $n = 10$, trained as described in Section 4.1, but didn't rescale the plot to match the original arrival rates per interval. The time axis of Figure 15 represents the hour of the day.

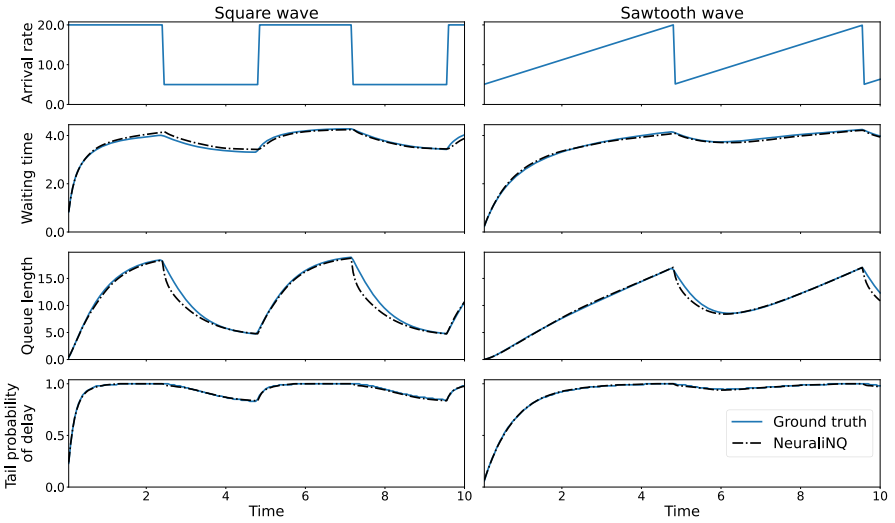


Fig. 17 NeuraliNQ vs. simulations for square and sawtooth arrival rates

B.2 Other nonstationary arrival patterns

In this section, we assess the generalization capabilities of our method for different arrival patterns. Specifically, we apply our model—previously trained on sinusoidal arrival rates as in (6)—to predict the transient queueing dynamics with non-sinusoidal and non-smooth arrival rates, including (i) trapezoidal, (ii) triangular, (iii) square, and (iv) sawtooth patterns. Our numerical examples show good performance similar to that in Section 4.2 (with TAMSE of 2.73e-3 and 3.01e-3 for the trapezoid and triangular waveforms, respectively). In Figure 16, we present each waveform, the model predictions for waiting time, queue length, and TPoD $\mathbb{P}(W_t > \tau)$ with $\tau = 2.0$, and the corresponding ground truth from simulation. The strong performance in the trapezoidal and triangular cases is not surprising, as these are continuous functions that are relatively similar to sinusoidal patterns. In contrast, the square and sawtooth arrival patterns, being discontinuous and featuring abrupt jumps, present a greater challenge. We expect NeuraliNQ, trained on sinusoidal data, to be less effective in these scenarios. This issue is most pronounced in regions where $\widehat{\boldsymbol{S}}^{\infty}_{t+1}$ significantly deviates from $\boldsymbol{S}_t$, particularly when $\widehat{\boldsymbol{S}}^{\infty}_{t+1}$ is much larger or smaller than $\boldsymbol{S}_t$. According to (3), this corresponds to high $\boldsymbol{\beta}_{t+1}$ and low $\boldsymbol{S}_t$, or vice versa. Indeed, Figure 17 shows that NeuraliNQ tends to underestimate both waiting times and queue lengths when the arrival rate drops sharply to a much lower value. This occurs because the sinusoidal training data fails to adequately capture the queueing dynamics that arise under such drastic changes in arrival rates. In the next subsection, we investigate how NeuraliNQ can be further fine-tuned in order to better represent these discontinuous arrival patterns.

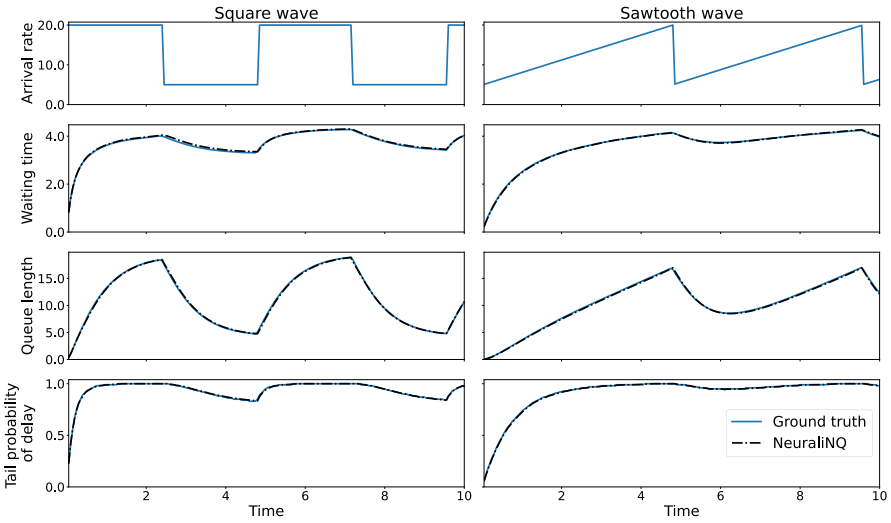


Fig. 18 NeuraliNQ vs. simulations for square and sawtooth arrival rates using a fine-tuned TRF network

B.3 Fine-tuning the TRF network for discontinuous arrival patterns.

One of the benefits of NeuraliNQ compared to analytical methods is the convenience of improving its accuracy using additional training data. In this section, we demonstrate this by improving the accuracy of a model trained on sinusoidal arrivals (as proposed in Section 4.2), for the square and sawtooth arrival trajectories presented in the previous Section, where our method reaches its limits (see Figure 17). As already discussed, our method is less accurate for the under-represented training data combination of low arrival rates in $\boldsymbol{\beta}_{t+1}$ and highly congested current state $\boldsymbol{S}_t$. To mitigate this data gap, we include additional training data for training our TRF model (our PSS model is not affected by transient patterns since its only function is to predict the asymptotic equilibrium state) that contain such combinations of $\boldsymbol{\beta}_{t+1}$ and $\boldsymbol{S}_t$.

These under-represented combinations can be simulated using piecewise constant arrival patterns. For example, we keep a high arrival rate c_1 for time t_1 and switch to a low arrival rate c_2 so we can capture the "draining" process of the queue. We repeat our method by including 40 additional training trajectories with randomly sampled parameters t_1, c_1 and c_2. We note here that this under-represented problem space is very narrow compared to the rest of the learned TRF surface (the top side of panel (b) in Figure 7). For this reason, the additional data needed are fairly limited compared to the 160 sinusoidal arrivals. Keeping the same training hyperparameters as before, we retrain the TRF network and reevaluate NeuraliNQ for the same square and sawtooth arrival patterns. This resulted in a significant improvement in accuracy, reducing errors from 4.45e−2 in the initially trained TRF to 3.65e−3 after retraining with the new data. Figure 18 illustrates the performance of the improved NeuraliNQ, showing a marked reduction in errors during sudden drops in arrival rates, where the original network had previously performed suboptimally.

We summarize our findings: to enable NeuraliNQ to handle a wide range of arrival patterns, the training process should incorporate data generated under both continuous and discrete arrival rates. The discrete case may be particularly important in practice, as demand forecasting in real-world service systems often assumes a constant arrival rate over successive intervals, such as 30 min or 1 h. In practice, NeuraliNQ performs at its best when guided by a user with a deep understanding of the problem structure. In such cases, the improvement process is conceptually straightforward, as these new steps are a natural extension of the existing training pipeline. This further highlights the flexibility and ease of use that NeuraliNQ offers.

C Does earlier system information help better predict future state?

As illustrated in Section 4, it is evident that our NeuraliNQ framework is able to effectively predict the future queueing performance (e.g., mean waiting time at $t+1$) based on the "snapshot" of the system (i.e., mean waiting time at t). Notably, it does so in a non-Markovian queue with non-Poisson arrivals, and nonexponential service and abandonment times. This may seem counterintuitive, as traditionally, only in a Markovian model is the system's dynamics fully characterized by the present state. The general perception of non-Markovian systems is that the present state provides only partial descriptions for the system's evolutionary state. Hence, given the state at t, the prediction of the future can often be improved by providing additional information of past states (before t), because this can help add supplemental information about time t.

Upon initial inspection, the exceptional performance of NeuraliNQ appears to defy common expectations for non-Markovian systems. However, this perception can be misleading; we elaborate on this below. A Markov model (e.g., Markov chain) is often defined using the so-called Markov property in the form of a conditional probability. Specifically, we require that $\mathbb{P}(Y_{t+1} = j|Y_t, Y_{t-1}, \ldots, Y_0) = \mathbb{P}(Y_{t+1} = j|Y_t)$, in order for the process $\{Y_t, t = 0, 1, 2, \ldots\}$ to be a Markov chain. On the other hand, if the above equality is violated, Y_t is no longer a Markov chain. A queueing system featuring non-Poisson arrivals, and nonexponential service and abandonment times is evidently not a Markov chain. In fact, NeuraliNQ is by no means assuming so. Specifically, NeuraliNQ's stepwise lookahead procedure utilizes the *sample-averaged* system state (e.g., mean waiting time) rather than the *realized* system state (e.g., the observed waiting time) to predict future sample-averaged system state. The distinction is that the former describes the system's aggregated state accounting for all possible scenarios, while the latter is based on one particular scenario. Taking the queue length as an example: when abandonment times are nonexponential, knowing that there are 10 customers waiting in the queue at t is not sufficient to predict the next event, unless we know the precise ages of these 10 customers; in case of deterministic abandonment times, if all ages are small (large), then less (more) customers are likely to abandon in the next moment. In contrast, the information that the *expected* queue length is 10 has already taken into account all possible scenarios (including cases of larger and smaller ages).

Table 8 TAMSEs under additional earlier states

Number of earlier states j	1	2	3	4	5
TAMSE	2.24e−3	2.22e−3	2.27e−3	2.18e−3	2.22e−3

To give quantitative verification, we extend the snapshot state space in NeuraliNQ by including additional earlier states. In particular, we extend the prediction framework from (2) to

$$\boldsymbol{S}_{t+1} = \mathcal{F}((\boldsymbol{S}_t ... \boldsymbol{S}_{t-j}), \boldsymbol{\beta}_t), \quad t = j, j+1, j+2, \ldots \tag{7}$$

where we predict $\boldsymbol{S}_{t+1}$ using a sequence of previous states $(\boldsymbol{S}_t ... \boldsymbol{S}_{t-j})$ of length $j \geq 1$. In Table 8, we report the TAMSEs of NeuraliNQ under $j = 1, \ldots, 5$. Our results reveal similar values of TAMSEs and confirm that the inclusion of additional previous states does not improve the prediction accuracy. This experiment shows that the snapshot information is sufficient for NeuraliNQ.

References

1. Aras, A.K., Chen, X., Liu, Y.: Many-server Gaussian limits for non-Markovian queues with customer abandonment. Queueing Syst **89**(1), 81–125 (2018)
2. Aras, A.K., Liu, Y., Whitt, W.: Heavy-traffic limit for initial content process. Stoch. Syst. **7**(1), 95–142 (2017)
3. Ata, B., Harrison, M., Si, N.: Drift control of high-dimensional RBM: a computational method based on neural networks (2023)
4. Baron, O., Krass, D., Senderovich, A., Sherzer, E.: Supervised ML for solving the $GI/GI/1$ queue. INFORMS J. Comput. **36**(3), 766–786 (2024)
5. Brown, L., Gans, N., Mandelbaum, A., Sakov, A., Shen, H., Zeltyn, S., Zhao, L.: Statistical analysis of a telephone call center: a queueing science perspective. J. Am. Stat. Assoc. **100**, 36–50 (2005)
6. Brown, L., Gans, N., Mandelbaum, A., Sakov, A., Shen, H., Zeltyn, S., Zhao, L.: Statistical analysis of a telephone call center: a queueing-science perspective. J. Am. Stat. Assoc. **100**(469), 36–50 (2005)
7. Che, E., Dong, J., Namkoong, H.: Differentiable discrete event simulation for queuing network control, 2024. [Online]. Available: https://arxiv.org/abs/2409.03740
8. Chen, N., Guilek, R., Lee, D., Shen, H.: Can customer arrival rates be modelled by sine waves? Serv. Sci. **16**(2), 70–84 (2023)
9. Chen, X., Hong, G., Liu, Y.: Online learning and optimization for queues with unknown demand curve and service distribution. Working paper (2023)
10. Chen, X., Liu, Y., Hong, G.: An online learning approach to dynamic pricing and capacity sizing in service systems. Oper. Res. **72**(6), 2263–2775 (2024)
11. Chen, Y., Casale, G.: Deep learning models for automated identification of scheduling policies. In: 2021 29th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 1–8 (2021)
12. Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation (2014)
13. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Gated feedback recurrent neural networks.(2015)
14. Dai, J.G., Gluzman, M.: Queueing network controls via deep reinforcement learning. Stoch. Syst. **12**(1), 30–67 (2021)
15. Deng, L., Yu, D.: Deep learning: methods and applications. Found. Trends Signal Process. **7**(3–4), 197–387 (2014). https://doi.org/10.1561/2000000039

16. Garbi, G., Incerto, E., Tribastone, M.: Learning queuing networks by recurrent neural networks. CoRR arXiv:2002.10788 (2020)
17. Garnett, O.O., Mandelbaum, A., Reiman, M.: Designing a call center with impatient customers. Manuf. Serv. Oper. Manag. **4**(3), 208–227 (2002)
18. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. The MIT Press, Cambridge (2017)
19. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks. Commun. ACM **63**(11), 139–144 (2020). https://doi.org/10.1145/3422622
20. Green, L., Kolesar, P., Whitt, W.: Coping with time-varying demand when setting staffing requirements for a service system. Prod. Oper. Manag. **16**, 13–39 (2007)
21. He, S.: Diffusion approximation for efficiency-driven queues when customers are patient. Oper. Res. **68**(4), 1265–1284 (2020)
22. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
23. Honnappa, H., Jain, R., Ward, A.R.: The $\Delta(i)/GI/1$ queueing model, and its fluid and diffusion approximations. Queueing Syst. **80**(1), 71–103 (2015)
24. Jia, H., Shi, C., Shen, S.: Online learning and pricing for service systems with reusable resources. *Working paper*, 2021. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3755902
25. Kaplan, J., McCandlish, S., Henighan, T., Brown, T.B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., Amodei, D.: Scaling laws for neural language models 2020. [Online]. Available: https://arxiv.org/abs/2001.08361
26. Karpathy, A., Johnson, J., Fei-Fei, L.: Visualizing and understanding recurrent networks (2015)
27. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2017)
28. Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., Anandkumar, A.: Neural operator: learning maps between function spaces with applications to pdes. J. Mach. Learn. Res. (2023)
29. Krishnasamy, S., Sen, R., Johari, R., Shakkottai, S.: Learning unknown service rates in queues: a multiarmed bandit approach. Oper. Res. **69**(1), 315–330 (2021)
30. Liu, B., Xie, Q., Modiano, E.: Reinforcement learning for optimal control of queueing systems. In: 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton), pp. 663–670 (2019)
31. Liu, R., Kuhl, M., Liu, Y., Wilson, J.: Modeling and simulation of nonstationary non-Poisson arrival processes. INFORMS J. Comput. **31**, 347–366 (2019)
32. Liu, Y.: Staffing to stabilize the tail probability of delay in service systems with time-varying demand. Oper. Res. **66**, 514–534 (2018)
33. Liu, Y., Sun, X., Hovey, K.: Scheduling to differentiate service in a multiclass service system. Oper. Res. **70**(1), 527–544 (2022)
34. Liu, Y., Whitt, W.: Large-time asymptotics for the $G_t/M_t/s_t + GI_t$ many-server fluid queues with abandonment. Queueing Syst. **67**, 145–182 (2011)
35. Liu, Y., Whitt, W.: A network of time-varying many-server fluid queues with customer abandonment. Oper. Res. **59**(4), 835–846 (2011)
36. Liu, Y., Whitt, W.: A many-server fluid limit for the $G_t/GI/s_t + GI$ queueing model experiencing periods of overloading. Oper. Res. Lett. **40**(5), 307–312 (2012)
37. Liu, Y., Whitt, W.: The $G_t/GI/s_t + GI$ many-server fluid queue. Queueing Syst. **71**(4), 405–444 (2012)
38. Liu, Y., Whitt, W.: Algorithms for time-varying networks of many-server fluid queues. Informs J. Comput. **26**(1), 59–73 (2014)
39. Liu, Y., Whitt, W.: Many-server heavy-traffic limits for queues with time-varying parameters. Ann. Appl. Probab. **24**, 378–421 (2014)
40. Liu, Y., Whitt, W., Yu, Y.: Approximations for heavily-loaded $G/GI/n+GI$ queues. Nav. Res. Logist. **63**, 187–217 (2016)
41. Mandelbaum, A., Massey, W.A., Reiman, M.I.: Strong approximations for Markovian service networks. Queueing Syst. **30**(1), 149–201 (1998)
42. Ojeda, C., Cvejosky, K., Sánchez, R.J., Schuecker, J., Georgiev, B., Bauckhage, C.: Recurrent adversarial service times 2019. [Online]. Available: https://arxiv.org/abs/1906.09808
43. Raeis, M., Tizghadam, A., Leon-Garcia, A.: Predicting distributions of waiting times in customer service systems using mixture density networks. In: 2019 15th International Conference on Network and Service Management (CNSM). IEEE, Oct. (2019)

44. Raissi, M., Perdikaris, P., Karniadakis, G.: Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys. **378**, 686–707 (2019)
45. Shah, D., Xie, Q., Xu, Z.: Stable reinforcement learning with unbounded state space. Working Paper, 2020. [Online]. Available: https://arxiv.org/pdf/2006.04353.pdf
46. Sherzer, E.: Computing the steady-state probabilities of the number of customers in the system of a tandem queueing system, a machine learning approach. Eur. J. Oper. Res. **326**(1), 141–156 (2025)
47. Sherzer, E., Baron, O., Krass, D., Resheff, Y.: Approximating $G(t)/GI/1$ queues with deep learning. Eur. J. Oper. Res. **322**(3), 889–907 (2025)
48. Shi, P., Chou, M., Dai, J.G., Ding, D., Sim, J.: Models and insights for hospital inpatient operations: time-dependent ed boarding time. Manag. Sci. **62**(1), 1–28 (2014)
49. Sitzmann, V., Martel, J.N.P., Bergman, A.W., Lindell, D.B., Wetzstein, G.: Implicit neural representations with periodic activation functions (2020)
50. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need (2023)
51. Wang, R., Honnappa, H.: Calibrating infinite server queueing models driven by cox processes. In: Winter simulation conference (WSC) 2021, 1–12 (2021)
52. Wang, R., Jaiswal, P., Honnappa, H.: Estimating stochastic Poisson intensities using deep latent models. In: Winter Simulation Conference (WSC), 2020, pp. 596–607 (2020)
53. Whitt, W.: Engineering solution of a basic call-center model. Manag. Sci. **51**(2), 221–235 (2005)
54. Whitt, W.: A multi-class fluid model for a contact center with skill-based routing. AEU-Int. J. Electron. Commun. **60**, 95–102 (2006)
55. Whitt, W.: Fluid models for multiserver queues with abandonments. Oper. Res. **54**(1), 37–54 (2016)
56. Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., He, Q.: A comprehensive survey on transfer learning. Proc. IEEE **109**(1), 43–76 (2021)