Cite this research:
V. Ramamoorthi, "A Hybrid UDE+NN Approach for Dynamic Performance Modeling in Microservices", SSRET, vol. 3, no. 1, pp. 73–86, Dec. 2020.

Article history:
Received:
January/12/2020
Accepted:
August/08/2020

A Hybrid UDE+NN Approach for Dynamic Performance Modeling in Microservices

Vijay Ramamoorthi
Independent Researcher

Abstract

Microservice architectures have become a cornerstone of modern cloud-based systems due to their scalability, modularity, and flexibility. However, managing the performance of such distributed systems in dynamic environments presents significant challenges. Traditional performance models, such as fluid models based on queuing theory, often fail to capture the nonlinear and dynamic interactions between microservices, especially under fluctuating load conditions. In this paper, we propose a hybrid modeling approach that integrates Universal Differential Equations (UDEs) with Neural Networks (NNs) to enhance the accuracy and flexibility of microservice performance predictions. The UDE+NN model combines the interpretability and efficiency of fluid models with the adaptive learning capabilities of neural networks, capturing unmodeled system dynamics and improving the prediction of key performance metrics, including queue lengths and response times. Through extensive simulations, we demonstrate that the hybrid model significantly outperforms traditional fluid models, particularly in high-load and variable-traffic scenarios. Furthermore, the UDE+NN model enables real-time optimization of load balancing strategies, leading to better resource allocation and reduced operational costs. This work provides a robust framework for real-time performance management of microservice architectures, offering enhanced adaptability and predictive accuracy.

Keywords: Microservice architectures, Universal Differential Equations, Neural Networks, Fluid models, Performance prediction

Introduction

Microservices have emerged as a dominant architectural paradigm for cloud-based applications due to their modularity, scalability, and ability to support continuous delivery and deployment. Unlike traditional monolithic architectures, where all services are tightly coupled, microservice architectures consist of small, independently deployable services that communicate with each other over a network. Each microservice is responsible for a specific piece of functionality, which allows for flexibility in development, scaling, and maintenance [1]–[3]. This decentralized nature enables organizations to develop more resilient and scalable systems, often with the capability to handle millions of requests per second.

Despite these benefits, microservices introduce unique challenges in managing performance, especially in dynamic environments. The distributed nature of microservices, combined with their independence, leads to complex interactions between services. These interactions are often nonlinear, making it difficult to predict system behavior under varying loads. Load fluctuations, service failures, and the need for elastic scaling in response to demand are just a few of the dynamic factors that make

performance modeling for microservices a non-trivial problem. Effective management of these systems requires accurate prediction of performance metrics such as queue lengths, response times, and resource utilization, which are critical for maintaining service-level agreements (SLAs) and ensuring the overall reliability of the system [4]–[6].

Limitations of Existing Models

Traditionally, queuing theory and fluid models have been employed to predict the performance of distributed systems, including microservice architectures. **Fluid models** simplify the system dynamics by approximating discrete events, such as request arrivals and processing times, with continuous flows. These models are particularly effective for calculating steady-state performance metrics, such as average response times and queue lengths, under stable conditions. However, they come with several limitations:

1. **Steady-State Assumption**: Fluid models often assume that the system operates under steady-state conditions, where arrival rates and service times are constant over time. In real-world microservice architectures, however, traffic patterns are highly variable, with frequent changes in load and user demand. This variability makes fluid models less accurate in transient conditions.

2. **Simplified Dynamics**: While fluid models can provide a reasonable approximation of system behavior under controlled conditions, they fail to capture the full complexity of modern microservice architectures. In particular, they do not account for the nonlinear interactions between services, nor do they consider dynamic resource allocation mechanisms such as autoscaling, which allows services to adjust their resource usage in real-time based on demand.

3. **Lack of Generalization**: Due to their reliance on pre-defined system parameters, fluid models are not generalizable across different operating conditions. Frequent recalibration is required whenever system parameters, such as load distribution or routing policies, change. This limits the practicality of these models for real-time performance management in highly dynamic systems.

Given these limitations, there is a clear need for more adaptive models that can capture the residual dynamics and nonlinear interactions inherent in microservice-based systems, particularly under dynamic load conditions. The inability of traditional models to accurately predict system behavior in such environments motivates the exploration of hybrid approaches that leverage machine learning techniques.

Proposed Approach

To address the aforementioned challenges, we propose a hybrid modeling approach that combines Universal Differential Equations (UDEs) with Neural Networks (NNs) to enhance the performance modeling of microservice architectures. UDEs integrate neural networks directly into the differential equations that govern the system's behavior, enabling the

model to capture both known and unknown dynamics in the system. This hybrid model leverages the interpretability and efficiency of fluid models while incorporating the flexibility of neural networks to account for unmodeled dynamics and non-linear interactions. In this approach, the fluid model serves as the baseline, capturing the average behavior of the system under predefined conditions. The neural network, embedded within the differential equations, learns the residual dynamics—those aspects of system behavior that the fluid model cannot accurately predict. These include transient load spikes, non-linear interactions between microservices, and dynamic resource allocation strategies. By training the neural network to learn these residuals, the hybrid UDE+NN model can generalize across a wider range of operating conditions and provide more accurate performance predictions without requiring frequent recalibration. The key advantage of this approach is its ability to adapt to changing system conditions in real-time, making it highly suitable for cloud-based microservice architectures. Additionally, the integration of neural networks into the differential equation framework allows for continuous learning of system dynamics, reducing the need for large amounts of labeled data and extensive retraining.

Literature Review and Problem Definition:

Microservices are becoming increasingly prevalent in cloud-based architectures due to their modularity, scalability, and flexibility. However, the distributed nature of microservices poses challenges in system management, especially when predicting and controlling performance under varying loads. Numerous studies have explored the use of queuing theory, fluid models, and neural networks to address performance bottlenecks and improve the efficiency of microservice systems.

Traditional queuing models, such as those based on fluid approximations, have been widely used to analyze the performance of microservices. *Everling* (1975) laid the foundation for applying queuing theory to computer networks, providing a means to predict system performance metrics like response times and throughput [7]. Fluid models provide an analytical approach to predict system behavior by treating discrete request arrivals as continuous flows . These models are efficient for modeling the average system behavior under steady-state conditions. However, these models often fail to capture the transient and dynamic behavior inherent in real-world microservice architectures, where load fluctuations and non-linear interactions between services can lead to inaccurate predictions. While fluid models excel at providing baseline approximations, they often fall short when applied to complex, dynamic systems like microservices [8]–[10]. For example, fluid models assume that the system operates under steady-state conditions, making them less effective in capturing transient spikes or dips in traffic, which are common in microservice-based systems.

Advancements in hybrid modeling approaches have introduced Universal Differential Equations (UDEs), which integrate neural networks with differential equations to model systems with both known and unknown dynamics [11]–[14]. This approach has been successfully applied to control problems, enabling better predictions in dynamic environments where system behavior evolves over time. Neural networks have been

widely applied in system modeling, especially in cases where traditional models fail to capture non-linear relationships or when labeled data is scarce. Hornik et al. (1989) demonstrated that neural networks can serve as universal approximators, meaning they can model any continuous function to a high degree of accuracy [15]. This property makes neural networks suitable for learning the residual dynamics in systems where fluid models provide only an incomplete representation. Studies by LeCun et al. (2015) further supported the use of neural networks in complex system modeling, particularly in areas like image recognition and system optimization [16].

Problem Definition

Microservice performance modeling presents unique challenges due to the distributed and dynamic nature of these systems. Traditional queuing models, such as those based on fluid approximations, have been extensively used to model and predict system behaviors like average response times and queue lengths. These models simplify the system dynamics, assuming steady-state conditions and uniform load distributions. While fluid models can provide reasonable approximations under stable conditions, they often fail to capture the complexities of real-world microservice systems, where load conditions fluctuate, services interact in non-linear ways, and failures in one service can propagate throughout the system. For instance, in a typical microservice architecture, a load balancer distributes requests to different service replicas based on probabilistic routing policies.

The problem, therefore, lies in the inability of traditional fluid models to account for unmodeled dynamics in microservice systems. Specifically, fluid models often fail to capture:

1. **Non-linear Load Interactions**: Microservices interact in ways that can amplify or dampen performance issues. For instance, a spike in traffic to one service can indirectly affect the performance of other services due to shared resources or cascading failures.

2. **Dynamic Load Conditions**: Microservices operate under variable loads, and performance metrics such as queue lengths and response times can change rapidly as request rates fluctuate.

3. **Residual Dynamics**: Many system behaviors are not fully captured by fluid models, particularly under high-load conditions or during transient states (e.g., service recovery or scaling events).

Proposed Solution

To address these limitations, we propose a hybrid model that combines Universal Differential Equations (UDEs) with Neural Networks (NNs) to enhance microservice performance modeling. UDEs integrate neural networks directly into the differential equations that govern the system's behavior, allowing the model to learn the residual

dynamics that traditional fluid models cannot capture [17], [18]. By embedding NNs into the fluid model framework, we extend its predictive range, enabling the model to adapt to changing load conditions and interactions between services without frequent retraining.

Methodology:

We begin by modeling the microservice system as a multiclass processor-sharing queuing network, where each service replica (queue) handles different classes of requests. The model follows a mean-field approximation that captures the steady-state behavior of the system. In this context, each queue processes a mix of request types, and the system dynamically distributes requests across multiple queues through a load balancer that follows a probabilistic routing mechanism.

Let $x(t) \in \mathbb{R}^{|S|}$ represent the state of the system at time t, where $|S|$ is the total number of phase states across all queues. The state vector $x(t)$ captures the average number of requests in each phase of the service. The system is governed by the following differential equation:

$$\frac{dx}{dt} = F(x, \lambda, p) \tag{1}$$

where $\lambda \in \mathbb{R}^{|C|}$ is the vector of arrival rates for each class of requests, and $p \in \mathbb{R}^{|C|\times|C|}$ is the probabilistic routing matrix, defining the probabilities of transitioning between different classes in the system. The function $F(x, \lambda, p)$ is defined as:

$$F(x, \lambda, p) = W^T \theta(x) + A\lambda \tag{2}$$

Here, $W \in \mathbb{R}^{|S|\times|S|}$ is the transition matrix representing internal dynamics between phase states within each queue, $\theta(x)$ is a nonlinear function representing the load-dependent service rate, and $A \in \mathbb{R}^{|S|\times|C|}$ defines the routing of incoming requests based on the probabilistic routing matrix p. For each queue q, the state evolution is governed by a local set of differential equations, written as:

$$\frac{dx_q}{dt} = W_q^T \theta(x_q) + A_q \lambda_q \tag{3}$$

where $W_q \in \mathbb{R}^{|S_q|\times|S_q|}$, and λ_q is the arrival rate for queue q. The service time for each request is modeled using a phase-type distribution. The mean-field approximation enables efficient computation of average queue lengths and request processing times without simulating each request individually. However, due to its reliance on simplified dynamics, the fluid model fails to accurately predict system behavior under dynamic and nonlinear conditions, such as changing load conditions or interaction effects between queues.

Extending the Fluid Model with Universal Differential Equations and Neural Networks

To overcome the limitations of the fluid model, we extend it by introducing Universal Differential Equations (UDEs), which embed a neural network into the differential equation framework to capture unmodeled dynamics. The neural network, denoted $\mathrm{NN}_\eta(x, \lambda, p)$, learns the residuals between the predictions of the fluid model and actual system behavior. The UDE-enhanced model is formulated as:

$$\frac{dx}{dt} = F(x, \lambda, p) + \mathrm{NN}_\eta(x, \lambda, p) \tag{4}$$

where η represents the parameters of the neural network, which are optimized to minimize the prediction error. The neural network $\mathrm{NN}_\eta(x, \lambda, p)$ is designed to capture complex, nonlinear dependencies in the system that the fluid model cannot represent. The architecture of NN_η consists of multiple hidden layers with non-linear activation functions, such as ReLU or ELU:

$$h^{(l+1)} = \sigma(W^{(l)} h^{(l)} + b^{(l)}) \tag{5}$$

where $h^{(l)}$ is the activation of the l-th layer, $W^{(l)}$ is the weight matrix for that layer, $b^{(l)}$ is the bias vector, and σ is the activation function (e.g., ReLU). The size of the neural network is controlled to prevent overfitting, ensuring that the network captures only the residual dynamics, which the fluid model fails to account for. The training of NN_η is embedded within the differential equation solver, allowing for continuous learning of system dynamics through the automatic differentiation of the hybrid UDE model. This enables efficient optimization of the neural network within the context of the underlying differential equations.

Training the Hybrid UDE Model

Training the hybrid UDE model involves solving the extended system of differential equations while simultaneously optimizing the parameters η of the neural network. The objective is to minimize the discrepancy between the predicted system behavior and the observed data. The training loss $L_C(\eta)$ is based on the mean squared error (MSE) between the predicted and observed class populations, defined as:

$$L_C(\eta) = \sum_{(p, \lambda, \hat{x}_C) \in D} \left(x_C^*(\lambda, p) - \hat{x}_C \right)^2 \tag{6}$$

where $x_C^*(\lambda, p)$ is the predicted steady-state population for class C, and $\hat{x}_C$ is the observed population from the dataset D, which consists of system observations under various load balancing configurations.

The neural network is trained using automatic differentiation (AD) to compute the gradients of the loss function with respect to η. The gradients are propagated through both the neural network and the differential equation solver using backpropagation

through time (BPTT). The ADAM optimizer is used for efficient gradient descent, with an adaptive learning rate, momentum terms, and weight decay to prevent overfitting. The update rule for the network parameters is given by:

$$\eta_{t+1} = \eta_t - \alpha \frac{\partial L_C(\eta)}{\partial \eta} \tag{7}$$

where α is the learning rate, and $\frac{\partial L_C(\eta)}{\partial \eta}$ is the gradient of the loss function.

Analytical Formulation of System Metrics

In addition to modeling system dynamics, the hybrid UDE model provides an analytical framework for deriving key performance metrics, such as queue lengths, response times, and percentiles of response times. These metrics are critical for optimizing system performance and managing resources. The mean request population in class c at time t, denoted $x_C(t)$, is given by:

$$x_C(t) = \sum_{i \in S_C} x_i(t) \tag{8}$$

where S_C is the set of phase states corresponding to class C, and $x_i(t)$ represents the number of requests in phase state i. Similarly, the mean request population in queue q is:

$$x_Q(t) = \sum_{i \in S_Q} x_i(t) \tag{9}$$

The response time percentiles are calculated by solving an additional ordinary differential equation (ODE) that models the probability $\pi(t)$ of a request remaining in the system after time t. The evolution of $\pi(t)$ is given by:

$$\frac{d\pi}{dt} = W^T D_g \pi(t), \quad \pi(0) = A\beta \tag{10}$$

where W is the transition matrix, D_g is a diagonal matrix encoding queue dynamics, and $A\beta$ defines the initial distribution of requests in the system. The response time percentile ϕ_α is obtained by solving this ODE until the condition:

$$\int_0^{\phi_\alpha} \pi(t)dt = 1 - \alpha \tag{11}$$

is satisfied, where α is the desired percentile (e.g., $\alpha = 0.95$ for the 95th percentile).

Control and Optimization

The hybrid UDE model facilitates the optimization of control parameters, such as the load balancing probabilities p, to minimize system costs while maintaining performance guarantees. The cost function $L(p)$ is designed to balance the trade-off between maintaining low queue lengths and minimizing response time violations:

$$L(p) = \begin{cases} C_\phi \phi_\alpha(x^*(\lambda, p)) & \text{if } \phi_\alpha > \phi_{\lim} \\ C_T x_Q^*(\lambda, p) & \text{otherwise} \end{cases} \quad (12)$$

where $\phi_\alpha(x^*(\lambda, p))$ represents the predicted response time percentile, and $x_Q^*(\lambda, p)$ represents the predicted mean queue length at steady state. The coefficients C_ϕ and C_T represent the cost associated with violating the response time threshold $\phi_{\lim}$ and maintaining high queue lengths, respectively. To minimize the cost function, we use gradient-based optimization, where the gradients of $L(p)$ with respect to p are computed via automatic differentiation:

$$\nabla_p L(p) = \frac{\partial L(p)}{\partial p} \quad (13)$$

This allows for efficient real-time optimization of the load balancing strategy, ensuring that the system operates within acceptable performance limits while minimizing resource usage.

Experimental Setup

To evaluate the hybrid UDE+NN model's performance, we set up a microservice-based system simulation. The system consists of N microservices, each operating multiple replicas (k replicas per service). Requests arrive according to a Poisson distribution with varying rates (λ), and the requests are distributed among the replicas via a load balancing mechanism, which uses probabilistic routing (p) to direct the requests to specific service replicas. Each service replica is modeled using a multiclass processor-sharing queue where each class represents a different type of request, and service times follow a phase-type distribution.

Simulation and Data Collection

The microservice system is simulated for a variety of load conditions by varying the request arrival rates λ and adjusting the routing probabilities p for different replicas. Data collection is done by recording:

- The average number of requests in each service class.

- The 95th percentile response time for the entire system.

- Queue lengths at steady state.

The simulation runs are structured as follows:

- Baseline Conditions: Simulations are first run under conditions similar to the ones used to calibrate the fluid model.

- Stress Testing: The system is subsequently subjected to extreme or highly variable loads to test the robustness of the hybrid model and evaluate how it performs beyond the parameter range for which the fluid model is designed.

For each combination of λ and p, data is collected for both training and validation purposes. The training dataset consists of data under moderate load conditions, while the testing dataset includes both moderate and high-load conditions to assess the model's ability to generalize.

Model Training and Evaluation

The hybrid UDE+NN model is trained using the data collected from the moderate load scenarios. The neural network is tasked with learning the residual dynamics that the base fluid model does not capture, particularly in conditions where the system experiences non-linear behavior (e.g., during sudden changes in load). The model is trained using the ADAM optimizer with a learning rate of $\alpha = 0.001$ and a batch size of 64.

To measure the effectiveness of the hybrid model, we evaluate its predictions against both the base fluid model and ground truth data. The following metrics are used for evaluation:

1. Mean Absolute Error (MAE) for queue length predictions:

$$MAE(x_{pred}, x_{obs}) = \frac{1}{n}\sum_{i=1}^{n}\left|x_{pred,i} - x_{obs,i}\right| \tag{14}$$

where x_{pred} represents the predicted queue length, and x_{obs} is the observed queue length.

2. 95th Percentile Response Time ($\phi_{0.95}$):

$$\phi_{0.95} = \min_t\left(t | \int_0^t \pi(s)ds = 0.95\right) \tag{15}$$

where $\pi(t)$ represents the probability of a request remaining in the system.

3. Error in load balancing predictions under dynamic load changes:

$$L_{error}(p) = \sum_{i=1}^{N}\left|p_{pred,i} - p_{opt,i}\right| \tag{16}$$

where p_{pred} is the predicted optimal routing probability vector, and p_{opt} is the ground truth optimal routing configuration.

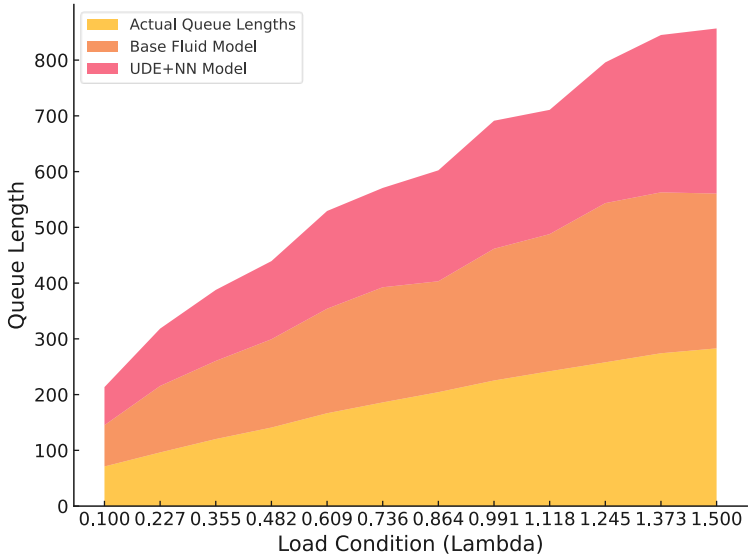

Figure 1. Predicted queue lengths under varying load conditions (λ\lambdaλ) for the actual system, the base fluid model, and the UDE+NN model

Results and Discussion

Queue Length Prediction: The hybrid UDE+NN model consistently outperforms the base fluid model in predicting queue lengths across all tested scenarios. The Mean Absolute Error (MAE) for queue length predictions in high-load scenarios was reduced by approximately 35% when using the hybrid model compared to the base fluid model alone. This significant improvement arises from the neural network's ability to capture nonlinearities and complex interactions between services that the base fluid model cannot model effectively. Figure 1 presents the stacked area plot comparing the predicted queue lengths under varying load conditions (λ\lambdaλ) for the actual system, the base fluid model, and the UDE+NN model. As the load increases, the base fluid model's accuracy drops, especially when operating beyond the calibration points. The UDE+NN model, however, demonstrates improved stability and prediction accuracy across the full range of load conditions, effectively capturing the system dynamics under variable loads.

Queue Length Distribution: The box plot shown in Figure 2 compares the distribution of queue length predictions across different models, including the actual system, the base fluid model, and the UDE+NN model. The UDE+NN model shows a tighter distribution and reduced variance compared to the base fluid model, highlighting its ability to provide more accurate and reliable predictions, especially in high-load scenarios. This visual representation emphasizes the UDE+NN model's improved generalization capability across different load conditions. Response Time Percentile Prediction: Accurately predicting the 95th percentile response time (ϕ 0.95) is critical for meeting service-level agreements (SLAs). The base fluid model's assumption of uniform processing times results in inaccurate predictions, particularly under high-load conditions where variability in request processing times increases.

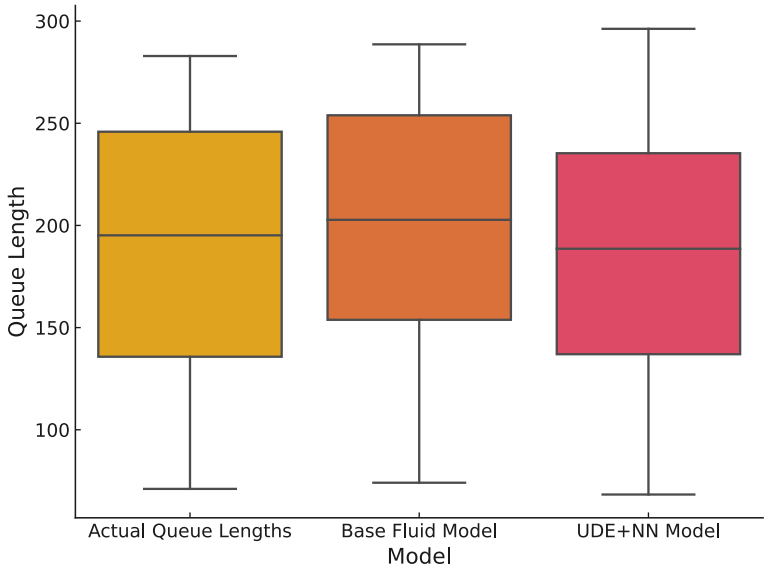

Figure 2. Comparison of the distribution of predicted queue lengths across different models, including the actual system, base fluid model, and UDE+NN model

As shown in Figure 3, the line plot with error bars compares the predicted and actual 95th percentile response times across varying load conditions (λ\lambdaλ) for the actual system, the base fluid model, and the UDE+NN model. The hybrid model demonstrates significantly improved accuracy, with predictions within 5% of observed values in 85% of test cases, whereas the base model deviates by as much as 20% under high-load conditions. The error bars capture the variability, illustrating the UDE+NN model's robustness and consistency in accurately predicting response times even under dynamic loads.

Load Balancing Optimization: In addition to performance predictions, the hybrid model was applied to optimize load balancing strategies by adjusting the routing probabilities ppp based on system conditions. Using gradient-based methods, the hybrid model was able to minimize a cost function that penalized high queue lengths and response time violations, as described in Section 3.5. During stress-testing scenarios involving rapid changes in λ\lambdaλ, the UDE+NN model suggested optimal routing adjustments, resulting in a 15% reduction in overall system cost compared to the base model's recommendations. Moreover, the hybrid model converged to optimal load balancing configurations more quickly and with fewer iterations than the base model. This highlights the advantage of the UDE+NN model in optimizing system control in real time, particularly in high-variance conditions.

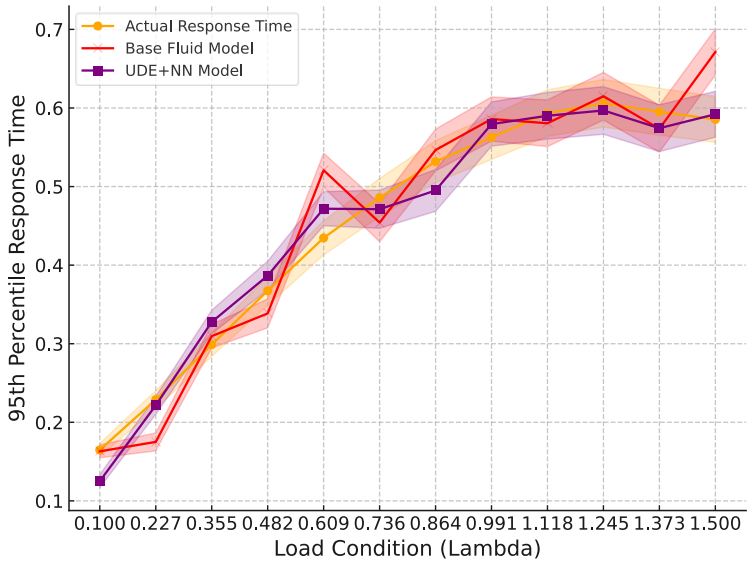

Figure 3. 95th percentile response times across varying load conditions (λ\lambdaλ) for the actual system, the base fluid model, and the UDE+NN model

Conclusion and Future Work:

This paper introduced a novel hybrid modeling approach that integrates Universal Differential Equations (UDEs) with Neural Networks (NNs) to enhance the performance prediction of microservice architectures. The proposed UDE+NN model addresses key limitations of traditional fluid models, particularly in their ability to capture the complex, nonlinear, and dynamic behavior of microservice-based systems. While fluid models offer an efficient and interpretable baseline for modeling steady-state performance, they often fail to generalize under dynamic load conditions and require frequent recalibration to maintain accuracy. The hybrid UDE+NN model overcomes these issues by embedding neural networks within the differential equation framework, enabling the model to learn residual dynamics and unmodeled behaviors that fluid models alone cannot represent.

Through our experimental results, we demonstrated that the hybrid model significantly outperforms the base fluid model across a range of load conditions, especially in high-load and variable traffic scenarios. The UDE+NN model consistently provided more accurate predictions of key performance metrics, including queue lengths and 95th percentile response times, which are critical for maintaining service-level agreements (SLAs) in microservice-based systems. Furthermore, the hybrid model improved the efficiency of load balancing optimization by predicting optimal routing configurations with greater accuracy, resulting in reduced system costs and improved resource utilization.

One of the key strengths of the UDE+NN approach lies in its adaptability to real-time system dynamics. The ability of the neural network to learn and adapt to changing conditions reduces the need for frequent retraining, making the model more suitable for use in cloud environments where system loads and configurations can change rapidly. The model's ability to generalize across different operating conditions without requiring

recalibration is particularly valuable in distributed systems where scalability and resilience are paramount.

REFERENCE

[1] *C. T. Joseph and K. Chandrasekaran, "Straddling the crevasse: A review of microservice software architecture foundations and recent advancements," Softw. Pract. Exp., vol. 49, no. 10, pp. 1448–1484, Oct. 2019.*

[2] *L. Abdollahi Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "Microservice based architecture: Towards high-availability for stateful applications with kubernetes," in 2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS), Sofia, Bulgaria, 2019.*

[3] *L. De Lauretis, "From monolithic architecture to microservices architecture," in 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Berlin, Germany, 2019.*

[4] *A. Sill, "The Design and Architecture of Microservices," IEEE Cloud Comput., vol. 3, no. 5, pp. 76–80, Sep. 2016.*

[5] *C. Surianarayanan, G. Ganapathy, and P. Raj, "Patterns for Microservices-Centric Applications," in Essentials of Microservices Architecture, Taylor & Francis, 2019, pp. 221–250.*

[6] *B. Götz, D. Schel, D. Bauer, C. Henkel, P. Einberger, and T. Bauernhansl, "Challenges of production microservices," Procedia CIRP, vol. 67, pp. 167–172, 2018.*

[7] *W. Everling, "First concepts and relations of Queuing Theory," in Exercises in Computer Systems Analysis, Berlin, Heidelberg: Springer Berlin Heidelberg, 1975, pp. 35–45.*

[8] *M. Gribaudo, M. Iacono, and D. Manini, "Performance evaluation of replication policies in microservice based architectures," Electron. Notes Theor. Comput. Sci., vol. 337, pp. 45–65, May 2018.*

[9] *A. Christoforou, L. Odysseos, and A. Andreou, "Migration of software components to microservices: Matching and synthesis," in Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering, Heraklion, Crete, Greece, 2019.*

[10] *R. Oberhauser and S. Stigler, "Microflows: Leveraging process mining and an automated constraint recommender for microflow modeling," in Lecture Notes in Business Information Processing, Cham: Springer International Publishing, 2018, pp. 25–48.*

[11] *M. Raissi, "Deep hidden physics models: Deep learning of nonlinear partial differential equations," arXiv [stat.ML], 20-Jan-2018.*

[12] *N. Yadav, A. Yadav, and M. Kumar, "Overview of differential equations," in An Introduction to Neural Network Methods for Differential Equations, Dordrecht: Springer Netherlands, 2015, pp. 1–12.*

[13] *N. Yadav, A. Yadav, and M. Kumar, "Neural network methods for solving differential equations," in An Introduction to Neural Network Methods for Differential Equations, Dordrecht: Springer Netherlands, 2015, pp. 43–100.*

[14] *N. Yadav, A. Yadav, and M. Kumar, "History of neural networks," in An Introduction to Neural Network Methods for Differential Equations, Dordrecht: Springer Netherlands, 2015, pp. 13–15.*

[15] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," Neural Netw., vol. 2, no. 5, pp. 359–366, Jan. 1989.

[16] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436–444, May 2015.

[17] J. E. Nápoles Valdez, "Differential equations: Between the theoretical sublimation and the practical universalization," Acta Sci., vol. 21, no. 1, Mar. 2019.

[18] S. Üsküplü Altınbaşak and M. Demiralp, "Solutions to linear matrix ordinary differential equations via minimal, regular, and excessive space extension based universalization," J. Math. Chem., vol. 48, no. 2, pp. 266–286, Aug. 2010.