

HOLMES: Health OnLine Model Ensemble Serving for Deep Learning Models in Intensive Care Units

Shenda Hong^{1,*}, Yanbo Xu^{1,*}, Alind Khare^{1,*}, Satria Priambada^{1,*}, Kevin Maher², Alaa Aljiffry², Jimeng Sun³, Alexey Tumanov¹

¹Georgia Institute of Technology, ²Childrens Healthcare of Atlanta, ³University of Illinois at Urbana-Champaign

ABSTRACT

Deep learning models have achieved expert-level performance in healthcare with an exclusive focus on training accurate models. However, in many clinical environments such as intensive care unit (ICU), real-time model serving is equally if not more important than accuracy, because in ICU patient care is simultaneously more urgent and more expensive. Clinical decisions and their timeliness, therefore, directly affect both the patient outcome and the cost of care. To make *timely* decisions, we argue the underlying serving system must be latency-aware. To compound the challenge, health analytic applications often require a combination of models instead of a single model, to better specialize individual models for different targets, multi-modal data, different prediction windows, and potentially personalized predictions. To address these challenges, we propose HOLMES—an online model ensemble serving framework for healthcare applications. HOLMES dynamically identifies the best performing set of models to ensemble for highest accuracy, while also satisfying sub-second latency constraints on end-to-end prediction. We demonstrate that HOLMES is able to navigate the accuracy/latency tradeoff efficiently, compose the ensemble, and serve the model ensemble pipeline, scaling to simultaneously streaming data from 100 patients, each producing waveform data at 250 Hz. HOLMES outperforms the conventional offline batch-processed inference for the same clinical task in terms of accuracy and latency (by order of magnitude). HOLMES is tested on risk prediction task on pediatric cardio ICU data with above 95% prediction accuracy and sub-second latency on 64-bed simulation.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks; Supervised learning by classification**; • **Applied computing** → **Health informatics**.

KEYWORDS

Healthcare; Health Informatics; Data Mining System; Software

* Authors contributed equally to this research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

© 2020 Association for Computing Machinery.

ACM ISBN 978-1-4503-7998-4/20/08...\$15.00

<https://doi.org/10.1145/3394486.3403212>

ACM Reference Format:

Shenda Hong^{1,*}, Yanbo Xu^{1,*}, Alind Khare^{1,*}, Satria Priambada^{1,*}, Kevin Maher², Alaa Aljiffry², Jimeng Sun³, Alexey Tumanov¹. 2020. HOLMES: Health OnLine Model Ensemble Serving for Deep Learning Models in Intensive Care Units. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20)*, August 23–27, 2020, Virtual Event, CA, USA. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3394486.3403212>

1 INTRODUCTION

Vast amount of real-time monitoring data such as electrocardiogram (ECG) and electroencephalogram (EEG) are being observed and collected especially in intensive care units (ICUs), which provides invaluable input for training deep learning (DL) models [21, 29, 37]. In fact deep learning models have achieved state-of-the-art performance in many healthcare and medical applications such as Radiology [30], Ophthalmology [12], and Cardiology [14, 16]. Almost all previous works focused on optimizing deep neural networks for prediction accuracy [35]. However, in a high stakes environment such as ICUs serving these trained machine learning models in *real time* is an equally important but hitherto largely overlooked requirement. In general, a computational graph of stateful and stateless components responsible for capturing and performing prediction on multi-modal sensory inputs must be provisioned to perform **high accuracy** predictions with **low latency** on a **dynamic** stream of multi-patient data.

Although recent work on prediction serving systems like TensorFlow Serving [27] and Clipper [9] have proposed general frameworks for serving deep models at the system level, they are unable to support the complex needs of health predictive models. First, these model serving platforms primarily target single model serving and can be thought of as the physical execution layer at the lower level of the stack. The complexity of health predictive pipelines involves capturing and pre-processing multi-modal sensory data, arriving at different frequencies, and performing periodic inference on this multi-rate, multi-modal stream of data under the constraint of soft real-time latency Service Level Objectives (SLOs). Second, in

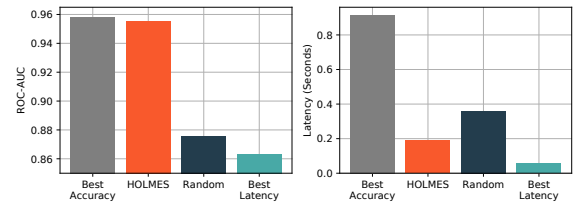


Figure 1: HOLMES finds a better balance between accuracy (ROC-AUC) and latency. HOLMES reaches competitive accuracy within the 200ms latency budget.

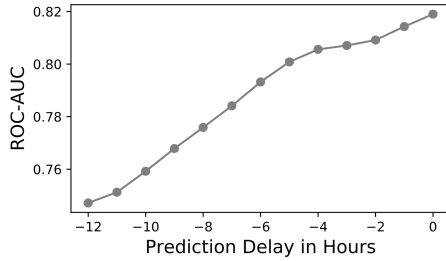


Figure 2: Accuracy decreases with prediction delay. Prediction accuracy plotted for CICU patient’s readiness for step-down transfer based on 3-lead continuous ECG waveforms.

a hospital setting often due to privacy/legal requirements, machine learning (ML) serving systems operate under the constraints of a fixed set of limited on-premises resources. This necessitates a careful exploration of the accuracy/latency tradeoff, whereby higher accuracy can be achieved at the expense of prohibitively high latency. Thus, the soft real-time serving platform needs to be able to explore this tradeoff and navigate it to find the right pipeline configuration for a specific clinical use case at hand. For example, in ICU environments, a large number of DL predictive models can be constructed based on different data modalities (e.g., ECG, heart rate, blood pressure), different prediction windows (e.g., 30 sec, 10 min, 1 hour, 1 day) and different patient data (e.g., models are retrained every week based on data from new patients).

Application 1: Length of Stay prediction (Accuracy first) ICU beds are limited and expensive resources to be managed effectively. It is estimated that there are 14.98 beds on average at each ICU ¹ [13], median daily cost of ICU ranges from \$6,318 (Medical Surgical ICU) to \$10,382 (Cardiac Surgical ICU) ² [11]. Accurate real-time prediction of length of stay and readiness for discharge of a patient is an important capability for ICU resource optimization. Model accuracy is essential as a wrong prediction will lead to inadequate resource allocation and sub-optimal care to critical patients.

Application 2: Mortality prediction (Latency first) ICU patients are vulnerable and unstable. Patient conditions may change quickly over time. Rapid response is the key to save a life. For example, accurate predicting cardiac arrest will happen even in the next 10 minutes can save lives. In this rapidly changing situation, the latency of the prediction is crucial. We want to be able to utilize most recently patient data to perform the prediction.

Deep learning models for ICU environments have drawn much more attention including risk prediction [37] and treatment recommendation [33]. However, it is difficult to support real-time model serving in ICU due to the following challenges:

- **Multiple data modalities:** Many different data modalities are captured in ICUs. Some modalities like Electrocardiogram (ECG) signals have high sampling frequency up to 1,000 Hz, while other modalities like medication only occurs once a few hours. It is unnecessary and nearly impossible to train and serve a unified model that serves all data modalities.
- **Noisy environment:** ICUs are highly dynamic environments. Data collections can be unreliable especially for sicker patients

as more interventions can happen to them which often affect the monitoring device (e.g., sensors fall off or are removed during interventions)

- **Heterogeneous conditions:** Diverse patients with very different conditions are present in ICU, which leads to different risk and monitoring needs. For example, complex patients after surgery may need to be monitored and assessed more frequently by multiple models, compared to stable patients who are close to being moved out of the ICU.
- **Limited computational resources:** Hospital is a typical resource constrained environment including computing. Most of the bedside computing or local clusters are limited in computing capability (e.g., the limited number of GPUs and memory). Real-time data capturing and model serving workload can be overwhelming for that hardware. While recent works can achieve predictive accuracy in ICU applications, those models can often be too heavy and inflexible to deploy. Since conditions of ICU patients often change quickly, a small latency for model serving is required especially for latency first tasks like cardiac arrest prediction.

We propose HOLMES—a novel real-time model ensemble composition and serving system for deep learning models with clinical applications in the ICU with the following contributions:

- **Model zoo:** Instead of the common approach to train one unified model serving all patients with the same set of data modalities, HOLMES trains a set of models (called *model zoo*). Each model can be specialized for one particular task using one data modality with a certain, independently chosen time window.
- **Ensemble composer:** HOLMES leverages sequential model-based Bayesian optimization with genetic exploration to select a subset of models from the model zoo for serving different patients under a restricted set of computational resources and within a specified latency budget.
- **Real-time model serving:** HOLMES extends the Ray [25] framework with a real-time model pipeline serving functionality.
- **Real-data experiment:** We extensively evaluate HOLMES on real ICU data in a realistic simulated streaming environment. Our experiments confirm HOLMES can simultaneously serve up to 100 beds of streaming data while achieving 95% accuracy for stepdown readiness prediction with sub-second latency.

In summary, HOLMES’ combination of the model ensemble search algorithm and a latency-aware, soft real-time model pipeline serving framework enables exploration of the accuracy/latency tradeoff space for a variety of relevant clinical use cases.

2 RELATED WORK

HOLMES draws its benefits from a combination of the automatic model ensemble composition algorithm and the underlying latency-sensitive model pipeline serving framework. In this section, we show how HOLMES builds on and compares to the state-of-the-art for both. The key takeaway in this comparison is two-fold. First, HOLMES proposes automatic model ensemble construction from specialized models that can individually be more narrowly trained to specialize in different data modalities and observation windows. This is a departure from a body of literature that depends on a single model. Second, HOLMES builds on a latency-aware system

¹Using data from American Hospital Association (AHA) statistics. Healthcare Cost Report Information System (HCRIS) data is unavailable.

²Data from Montefiore Medical Center in the Bronx, New York during 2013.

that is simultaneously capable of serving clinical use cases across the spectrum of prediction latency requirements. Different clinical applications can interact with different auto-constructed ensembles served by the same serving framework. We take a closer look at the state-of-the-art model development for the ICU application targeted in this paper, describe existing model serving platforms and optimization strategies.

Deep Models for ICUs. Deep models, a.k.a. deep neural networks, have achieved state-of-the-art performance in many application areas due to their ability to automatically learning effective features from large scale data [19]. ICU is one of the typical environment that generated large scale multi-modality data everyday [1, 8], so that design accurate deep models [15] in ICU has drawn a lot of attention from researchers in recent years.

For example, in [37], they proposed a unified recurrent attentive and intensive deep neural network for predicting decompensation and length of stay, by modeling high-frequency physiological signals, low-frequency vital signs, irregular lab measurements and discrete medication together. In [33], they built a recurrent neural network and learned with supervised reinforcement learning for treatment recommendation in ICU, by modeling static variables and low-frequency time-series variables. Other work applies temporal convolutions on the vital signs and lab sequences to predict decompensation [26]. The common characteristic of the above methods is that they aim to build unified models that serve all patients using the same data modalities. They are inflexible when encountering data heterogeneity. Besides, none of them move one step forward to serving in the real world.

Deep Models Serving. Training is only a fraction of the end-to-end Machine Learning lifecycle and has dominated the focus of much of the literature in model development for healthcare. While great results have been achieved in terms of accuracy, not as much work focused on the difficulties and importance of efficiently deploying trained healthcare models on a platform that provides unified support for a variety of clinical use cases.

Google’s TensorFlow Serving [27] is a well-recognized open source distributed framework for Machine Learning inference and is widely used to serve deep learning models. Despite its popularity and impact, the framework has some limitations for the purposes of our target application domain. First, TensorFlow Serving provides support for single model serving. While it is possible to write additional code and integrate inference to several models, it becomes a single unit of deployment and is configured as a black box. The clinical use cases that deal with multi-modal heterogeneous models depend on first-class support for multiple models composed in a pipeline at the framework level. This is one of the key contributions of HOLMES. Second, models trained in other frameworks, such as PyTorch [28] cannot easily be supported by TensorFlow Serving.

Ray [25] is a distributed framework that provides low-level support for a combination of stateful and stateless (side-effect free) computation, suitable by design for a variety of computational patterns that appear in ML workflows, including training and serving. We build on Ray as the underlying platform and provide ML pipeline serving functionality on top—a feature Ray does not provide out of the box. Ray’s tasteful choice of primitives and focus on ML and low system overhead is better suited for soft real-time serving than Apache Spark MLlib [23].

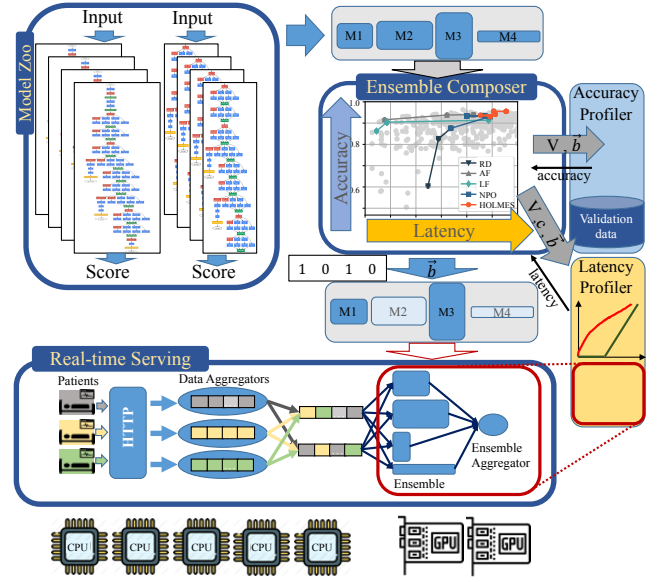


Figure 3: HOLMES system architecture: model zoo, ensemble composer and real-time serving system: the ensemble composer produces a model ensemble captured by \hat{b} ; the real-time serving serves the ensemble as a mix of stateful and stateless actors connected into a queueing pipeline.

Optimization Strategies. Finally, neural architecture search [7, 10, 22, 39] is an emerging area of research that targets efficient model composition at the operator level. While our work composes and serves optimal combinations of models as the fundamental building blocks. Besides, Bayesian optimization [31, 32] has been used to reduce experimental costs in hyper-parameter optimization to select the best single model [3, 4], or building a fixed-size ensemble learning [20]. It is a sequential design strategy for global optimization of black-box functions that doesn’t require derivatives [24]. In this paper, we solve model ensemble problem by extending Bayesian Optimization to Sequential Model-Based (Bayesian) Optimization (SMBO) [17] to further reduce the trial costs. Besides, we also introduce genetic algorithm to boost exploration.

3 HOLMES: HEALTH ONLINE MODEL ENSEMBLE SERVING

3.1 Overview

HOLMES consists of three main components illustrated in Figure 3: the model zoo, the ensemble composer, and the real-time serving system. The model zoo is populated with models trained with different model hyperparameters and different input data modalities.

The ensemble composer optimizes for validation accuracy subject to latency constraints. It selects an optimal model set given the number of patients and the resource constraints. Then the serving pipeline is deployed with the chosen ensemble and the sensory data aggregators and is configured to operate on a large amount of streaming data and many ensemble queries in real-time.

3.2 Model Zoo

Model zoo is a repository of various prediction models that are already trained and ready for deployment.

ICU monitoring data are usually continuous and multimodal. They include but are not limited to dense signals like Electrocardiogram (ECG), sampled at frequencies ranging from 125 to 1,000 Hz, vitals signs like Blood Pressure sampled per second, and sequential discrete events like laboratory test or medication administration charted irregularly. Therefore, the deep learning models in model zoo are trained for each of the data modalities with short segmentation windows. A final prediction score will be generated by ensembling (i.e., aggregating predictions from multiple) individual scores across different data modalities as well as across different time segmentations within the observation window.

The model zoo approach makes it possible to specialize in the models to different data modalities. Adding new data modalities is then simplified. Instead of retraining the whole monolithic model from scratch and replacing the model running in production completely, a new model can be added to the model zoo and *automatically* chosen by the ensemble composer. The ensemble composer is triggered when any of the input conditions are changed: (a) changes in the model zoo, (b) the number of patients to serve, (c) resource constraints.

In addition, we also consider different sizes of deep models by varying the architecture hyperparameters. Instead of training one accurate but large-sized deep model, we train a set of less accurate but smaller sized models by borrowing the prediction strength from ensemble modeling [38]. By breaking models down to smaller sizes, our model zoo is more flexible and enables the feasibility that scores can serve still (even less accurate) given limited computational resources, while more accurate scores from ensemble models will be served given adequate resources.

Models in the model zoo are trained offline using previously collected ICU data, tuned and validated on an independent data from new patients. We store the pre-trained models along with their profiles. As shown in Table 3, one example model profile $\mathbf{v} \in \mathbb{R}^m$ (m as the number of fields specified in a profile) can contain model size details such as depth, width, number of multiply and accumulate operations and GPU memory usages, input data information such as data modality and length of data segmentation that models were trained on, and model performance on the validation set such as ROC-AUC scores, etc. Fields in model profiles can vary case by case. Table 3 in the Appendix lists those used in this paper.

3.3 Ensemble Composer

In this section we describe the second component of HOLMES—the ensemble composer. Model composition into ensembles has been known to improve performance, but focused primarily on accuracy [38]. Unconstrained, ensemble size can easily absorb all models in the model zoo (Sec. 3.1). The latency of deploying large ensembles on limited computational resources can be prohibitive, particularly for real-time serving scenarios.

To address this, we propose a latency-aware ensemble composition method. In Fig. 11, we show that the range of possible latency/accuracy outcomes for arbitrary ensembles drawn from the model zoo is large. The best performing method must achieve the

Table 1: Notations.

Notation	Definition
n	Number of models in model zoo
m	Number of fields for model description
d	Number of fields for system configuration
$\mathcal{M} = \{m_1, m_2, \dots, m_n\}$	Model zoo
$\mathbf{V} \in \mathbb{R}^{n \times m}$	Model description
$\mathbf{c} \in \mathbb{R}^d$	System configuration
$\mathcal{B} \triangleq \{0, 1\}^n$	Exploration space
\mathbf{B}	Profile set
$\mathbf{b} \in \mathcal{B}$	Model selector
$f_a(\mathbf{V}, \mathbf{b})$	Accuracy profiler
$f_l(\mathbf{V}, \mathbf{c}, \mathbf{b})$	Latency profiler

highest accuracy for **any** specified latency threshold. Below we formulate ensemble composition as an optimization problem and propose an efficient search algorithm within Bayesian optimization framework to find an optimal solution.

3.3.1 Problem setup. Suppose we have M_1, M_2, \dots , up to M_n models in the model zoo with each model having a profile $\mathbf{v}_i \in \mathbb{R}^m$, then we can represent the entire model zoo as $\mathbf{V} \in \mathbb{R}^{n \times m}$. We denote one system configuration as a vector $\mathbf{c} \in \mathbb{R}^d$, where d is the total number of items needed to be configured in a system. They include but are not limited to the number of GPUs, memory size, or number of clients in the system. Lastly a model ensemble is uniquely identified using a binary indicator vector $\mathbf{b} \in \mathcal{B} \triangleq \{0, 1\}^n$ such that $b_i = 1$ indicates model M_i is selected for ensembling and 0 otherwise.

Accuracy and latency trade-off. Our goal of ensemble composer is to find the \mathbf{b}^* such that

$$\begin{aligned} \mathbf{b}^* &\equiv \arg \max_{\mathbf{b} \in \{0, 1\}^n} \underbrace{f_a(\mathbf{V}, \mathbf{b})}_{\text{Accuracy profiler}} \\ \text{s.t.} \quad &\underbrace{f_l(\mathbf{V}, \mathbf{c}, \mathbf{b})}_{\text{Latency profiler}} \leq L, \end{aligned} \quad (1)$$

where $f_a(\mathbf{V}, \mathbf{b})$ is an *accuracy profiler* that produces prediction accuracy on the validation set given an model ensemble \mathbf{b} selected from the model zoo \mathbf{V} , $f_l(\mathbf{V}, \mathbf{c}, \mathbf{b})$ is a *latency profiler* that computes the latency of serving the ensemble \mathbf{b} under a certain system configuration \mathbf{c} , and L is the latency constraint required by a real-time serving system. We introduce a function δ and rewrite the above optimization problem in the following equation

$$\max_{\mathbf{b} \in \{0, 1\}^n} L_a(\mathbf{b}) = f_a(\mathbf{V}, \mathbf{b}) + \delta(L - f_l(\mathbf{V}, \mathbf{c}, \mathbf{b})), \quad (2)$$

where δ is an activation function. If δ is a linear function, then the above Eq. (2) converges to the common Lagrange multiplier setting that supports soft constraint. If δ is a step function specified below, we reach a hard constraint on latency instead:

$$\delta(x) = \begin{cases} -\inf, & \text{if } x < 0 \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

Alternatively, we can switch the objective function and constraint in Eq. (1) and minimize the latency subject to a minimum accuracy requirement given any accuracy sensitive task. It's beyond the scope of this paper, but its formulation is similar (listed

in the supplementary material) and can be solved using the same searching algorithm we introduce below.

3.3.2 Ensemble Composer exploration. There exist two main challenges for solving the Eq. (2) optimization problem: 1) unknown accuracy profiler $f_a(\mathbf{V}, \mathbf{b})$ and latency profiler $f_l(\mathbf{V}, \mathbf{c}, \mathbf{b})$ for all ensemble choices when model zoo size n is large ($|\mathbf{b}| = 2^n$!); and 2) high dimensional binary searching space as opposed to continuous.

To tackle the first challenge, we adapt the widely known black-box searching algorithm – Bayesian optimization – into our searching algorithm, in which surrogate probability models [18] are used for approximating the accuracy and latency profilers. For the second challenge, as typical Bayesian exploration functions (such as Gaussian process regression) usually searches parameters in continuous space, we propose to use Genetic search algorithm [34] for exploration so that our method can support efficient search in high-dimensional binary space.

Under the Bayesian Optimization framework, our goal is to iteratively enrich a true valued set $\mathbf{B} \in \mathcal{B}$, in which accuracy and latency are truly profiled by $f_a(\mathbf{V}, \mathbf{b})$ and $f_l(\mathbf{V}, \mathbf{b})$, and then update the objective $L_a(\mathbf{b})$ in Eq. (2) given all the available \mathbf{b} 's in the set. When it reaches the budget of N profiler calls, a \mathbf{b}^* that maximizes $L_a(\mathbf{b})$ over the current set \mathbf{B} is returned as the optimal solution. The crucial step in the framework is how to smartly explore the searching space at each iteration, and pick the right vector into set \mathbf{B} so that we don't waste the budget of calling the profilers for evaluating the true accuracy and latency.

a) Genetic algorithm for exploring binary parameter space. The key idea of exploration is to search new \mathbf{b} 's towards the optimal \mathbf{b}^* and efficiently enlarge the set \mathbf{B} . Usually, when searching domain is continuous [2], random exploration would be better than grid search. However, when searching domain is binary as in our case, the benefit of randomness from dimensionality combination is low – there are only two values for each dimension of \mathbf{b} . Thus, we borrow the ideas from Genetic Algorithm (GA) [34], in which each \mathbf{b} in the explore space \mathcal{B} can be naturally represented as a genotype. Hence, genetic operators such as recombination and mutation can be utilized to explore our binary parameters. In detail, we define

$$\begin{aligned} \text{Recombination}(\mathbf{b}_1, \mathbf{b}_2): \quad \mathbf{b} &\triangleq \text{concat}(\mathbf{b}_1[1:i], \mathbf{b}_2[i+1:n]), \\ \text{Mutation}(\mathbf{b}_3, 1): \quad \mathbf{b} &\triangleq \mathbf{b}_3[i] = \begin{cases} 1, & \text{if } \mathbf{b}_3[i] = 0 \\ 0, & \text{if } \mathbf{b}_3[i] = 1, \end{cases} \end{aligned} \quad (4)$$

where i is a random number from $\{1, 2, \dots, n\}$. We perform S times of mutations (called mutation degree) on \mathbf{b}_3 , i.e., randomly sample a new vector from the neighborhood of \mathbf{b}_3 within a Manhattan distance of S , and denote the newly formed candidate set as \mathbf{B}' .

b) Surrogate models for approximating accuracy and latency profilers. For selecting the right points into \mathbf{B} , we aim to build less expensive surrogate probability models for approximating $f_a(\mathbf{V}, \mathbf{b})$ and $f_l(\mathbf{V}, \mathbf{c}, \mathbf{b})$. That is, based on the current true value set \mathbf{B} , we fit separated latency surrogate probability model \hat{f}_l using $\{(\mathbf{b}, f_l(\mathbf{V}, \mathbf{c}, \mathbf{b})) : \text{for all } \mathbf{b} \in \mathbf{B}\}$, and a accuracy surrogate probability model \hat{f}_a using $\{(\mathbf{b}, f_a(\mathbf{V}, \mathbf{b})) : \text{for all } \mathbf{b} \in \mathbf{B}\}$. Given a new candidate $\mathbf{b}' \in \mathbf{B}'$, we can evaluate their approximated latency as $\hat{f}_l(\mathbf{V}, \mathbf{b}')$ and approximated accuracy as $\hat{f}_a(\mathbf{V}, \mathbf{b}')$ using the surrogate models. Then we pick K candidates whose approximated

Algorithm 1 Ensemble Composer exploration in HOLMES

```

1: Input:  $\mathbf{V}$  model zoo,  $\mathbf{c}$  system constraint,  $L$  latency constraint
2: Parameters:  $\lambda$ ,  $N$  number of search iterations,  $N_0$  number of
   warm start samples,  $M$  number of explore samples,  $K$  number
   of newly added samples for profiling,  $S$  degree of mutation,  $p$ 
   probability of genetic explore,  $q$  probability of mutation.
3: Output:  $\mathbf{b}^*$ 
4: Initialize surrogate models  $\hat{f}_a$  and  $\hat{f}_l$ 
5: /* Warm start to get some seed solutions */
6: Warm start to get an initial  $\tilde{\mathbf{B}} \in \{0, 1\}^{N \times |\mathbf{V}|}$ 
7:  $\mathbf{B} = \emptyset \cup \tilde{\mathbf{B}}, Y_a = \emptyset, Y_l = \emptyset$ 
8: for  $i=1:N$  do
9:   /* Profile accuracy and latency in  $\mathbf{B}$  */
10:  Profile  $\tilde{Y}_a = \{f_a(\mathbf{V}, \mathbf{b}), \forall \mathbf{b} \in \mathbf{B}\}, \tilde{Y}_l = \{f_l(\mathbf{V}, \mathbf{c}, \mathbf{b}), \forall \mathbf{b} \in \mathbf{B}\}$ 
11:   $Y_a = Y_a \cup \tilde{Y}_a, Y_l = Y_l \cup \tilde{Y}_l$ 
12:  /* Fit surrogate models on profiled results */
13:  Fit  $\hat{f}_a$  using  $\mathbf{B}$  and  $Y_a$ , and fit  $\hat{f}_l$  using  $\mathbf{B}$  and  $Y_l$ 
14:  /* Genetic exploration, details in Algo.2 */
15:   $\mathbf{B}' = \text{Explore}(\mathbf{B}, M, S, p, q)$ 
16:  /* Approx. accuracy and latency in  $\mathbf{B}'$  */
17:  Approx.  $\hat{L}_a(\mathbf{B}') = \{\hat{f}_a(\mathbf{V}, \mathbf{b}) + \lambda(L - \hat{f}_l(\mathbf{V}, \mathbf{c}, \mathbf{b})), \forall \mathbf{b} \in \mathbf{B}'\}$ 
18:  /* Pick top-K highest valued vectors to get  $\tilde{\mathbf{B}}$  */
19:   $\tilde{\mathbf{B}} = \text{argsort}_K(\hat{L}_a(\mathbf{B}'))$ 
20:  /* Add seed solutions to profile set */
21:   $\mathbf{B} = \mathbf{B} \cup \tilde{\mathbf{B}}$ 
22: end for
23: /* Get the optimal solution from  $\mathbf{B}$  */
24:  $\mathbf{b}^* = \arg \max_{\mathbf{b} \in \mathbf{B}} L_a(\mathbf{b})$ 

```

objective values computed by Eq. (2) have ranked top K among the set \mathbf{B}' , add them into the current set \mathbf{B} and evaluate their true values using the accuracy and latency profilers.

We summarize our algorithm in Algorithm 1 (pseudo code of function *Explore* is given in supplementary material Algorithm 2).

3.3.3 Prediction ensemble. Given the optimal solution \mathbf{b}^* , we make our final prediction using bagging ensemble [5]:

$$\mathbb{E}[Y|\mathbf{x}] = \frac{1}{n} \sum_{i=1}^n \mathbf{b}_i^* \mathbb{E}_{m_i}[Y|\mathbf{x}], \quad (5)$$

where Y is the outcome measure of interest for a given prediction task, and \mathbf{x} is an instance of ICU data input.

3.4 Real-time Serving

Overview. The third component of HOLMES is the real-time serving system used to serve the model ensemble as part of the end-to-end ensemble pipeline. It serves two purposes: (1) ensemble candidate latency profiling (Fig. 3) and (2) real-time ensemble pipeline serving.

The served pipeline consists of four main components: the source of streaming data (typically bed-side patient monitoring data), the HTTP server that simplifies data ingest into the serving system, patient data accumulators that buffer the data, and the ensemble itself. The HOLMES pipeline is implemented by deploying the data aggregators and ensemble models as actors on top of Ray [25]. The ensemble queries are routed through multi-modal queues, with each queue corresponding to the appropriate data modality.

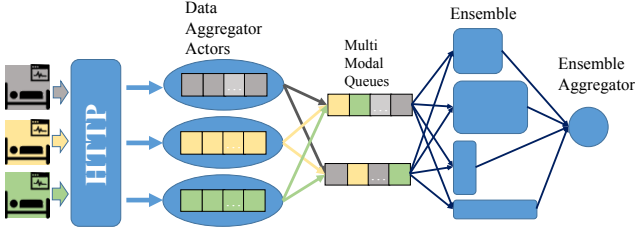


Figure 4: HOLMES Ensemble Serving System: requires a combination of stateful and stateless components, buffers multi-modal and multi-frequency sensory data in data aggregators, conditionally queries the ensemble, and returns the aggregated ensemble result to the patient monitoring system.

Support for stateful compute. The aggregator actors make it possible to deliver synchronized, coordinated buffers of streaming data to the ensemble. While the data stream may come at different frequencies and modalities, the input supplied to the ensemble must correspond to the same observation window across all sensors (to capture sensory correlations). E.g., ECG produces waveforms at 250 samples per second (250qps), while BP monitor outputs 1 sample per second (1qps). The data aggregator actors buffer this data for the *same* interval of time ΔT before the ensemble is queried. This requires a platform that’s capable of supporting stateful compute.

Support for stateless compute. Ensemble models are deployed as actors with a queue. Inference performed on the ensemble models implies that the models themselves can be stateless. This simplifies model lifecycle management and deployment and makes it possible to horizontally scale individual ensemble models as the load on the system increases. It is, therefore, imperative that the serving platform natively supports a combination of both *stateful* and *stateless* compute—a property HOLMES borrows from Ray [25] in its implementation of the ensemble serving platform.

Latency profiling. The latency profiling component is exposed to the ensemble composer (Sec. 3.3) through the API $f_l(V, c, \vec{b}_i)$, where \vec{b}_i corresponds to the model zoo subset choice considered at iteration i . Given the frequency of interaction with the latency profiler, it must be highly performing. Ensemble serving latency reported by the latency profiler consists of two constituent components: the queueing delay T_q and the serving latency of the ensemble T_s . Thus, $\hat{T} = T_q + T_s$, where \hat{T} is the estimate of end-to-end response time, T_q denotes queueing delays in the system, and T_s denotes serving delay of the ensemble. This breakdown is fundamental as T_q depends on the characteristics of the client load (i.e., ingest rate, number of clients, inter-arrival process characteristics), while T_s depends on the characteristics of the ensemble and the resources used. As such, the methodology for estimating T_q and T_s is completely different.

Estimating T_s and T_q . To estimate ensemble latency T_s , we measure its throughput capacity μ (qps) by directly performing inference on the ensemble in a closed-loop fashion and averaging over a statistically significant number of queries. T_s is then estimated by configuring the clients to generate queries at the ingest rate $\lambda \leq \mu$ and taking the 95th %-ile latency. This provides enough information to estimate T_q .

T_q depends on query inter-arrival distribution and, thus, indirectly depends on the number of patients generating query load on the ensemble. To estimate T_q we rely on network calculus. As profiling queries are generated, we construct two curves: an arrival and service curve. The arrival curve captures the max number of queries that have thus far been observed within any time interval of length Δt . The service curve captures the number of queries that can be serviced within an interval of time Δt -long. The maximum horizontal distance between these two curves (Fig. 5) is a known tight upper bound on the queueing latency for such a system.

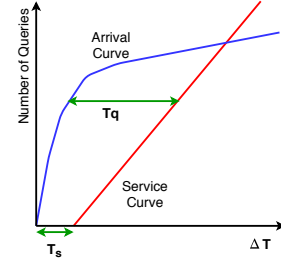


Figure 5: HOLMES latency estimator: latency estimation leverages network calculus. The arrival curve is constructed from query data observed at runtime during profiling. The service curve is constructed analytically. The maximum distance between the curves is known as a tight upper bound on queueing latency T_q .

4 EXPERIMENTAL EVALUATION

4.1 Experimental Setup

4.1.1 Model zoo training. Given a specific prediction task, model zoo needs to be populated with pre-trained models for subsequent ensemble composition. Here we describe what prediction task is targeted in our experiments, what ICU data is used, and what/how candidate models are trained.

Task. We focus on a binary classification task that predicts if a post-surgical (specifically the Norwood surgery) patient is getting *stable* or stay still *critical* in the Cardiac Intensive Care Units (CICUs).

Data. We have collected multimodal ICU data from 57 children who had undergone the Norwood procedure between 2016/10 and 2019/09 in the Cardiac Intensive Care Unit (CICU) at Children Healthcare of Atlanta (CHOA). These patients stayed at the CICU after the operation from 4 days up to 30 days, with 45 (78.9%) being successfully discharged to the general care cardiology floor, 6 (10.5%) deceased and 6 (10.5%) transferred for other operations. We extract data from the first two days of post-op CICU stays from all the 57 patients, and label them as 0 (*critical*); we extract data from the last day prior to floor transfer from the 45 successfully discharged patients, and label them as 1 (*stable*).

Modalities in this data include 3-lead (I, II, and III) ECG waveforms sampled at 250 Hz, 7 vital signs (mean blood pressure, SpO2, etc) sampled per second and 8 discrete lab results (pH, lactic acid, etc) measured when needed. We segment the continuous signals (waveforms and vital signs) into 30 second clips, and result at vectors of length 7,500 per single lead waveform and length 30 per single vital sign. There are in total of 328,320 data points per each signal labelled as 0 and 129,600 per signal labelled as 1.

Training details. We first split the cohort by putting 47 earlier patients into training set and 10 recent patients into test set. We train a state-of-art convolutional neural network, by modifying the kernel in the convolutional layer in ResNeXt [36] from 2-D patch to 1D stripe, individually for each single lead ECG clips. By filtering out missing signals, we obtain 164,972 training samples and 71,342 validation samples for lead-I ECG, 230,046 training and 71,364 validation samples for lead-II, and 130,564 training and 60,785 validation samples for lead-III.

We train differently sized networks per each ECG lead, particularly varying the number of filters in the first convolutional layer of the network between $\{8, 16, 32, 64, 128\}$ and number of residual blocks between $\{2, 4, 8, 16\}$. As a result, we reach at a deep model zoo of size $|V| = 3 \text{ (ECG leads)} \times 5 \text{ (filters)} \times 4 \text{ (blocks)} = 60$. As there is no need to train deep models on low resolution data like vital signs or lab measurements, we simply train a random forest for each vital sign, and a Logistic regression for labs. Since inference from these models using CPUs is negligible compared to inference from deep models using GPUs, we do not include them into the model zoo and don't take their inference time into account for system latency. But prediction accuracy ensembles the optimal deep models selected from the model zoo with these ML models.

4.1.2 System setup details. We specify system configuration $c \in \mathbb{R}^2$ to include number of GPUs and number of patients. We use 2 NVIDIA Tesla V100s with 32GB RAM as ensemble serving node in our experimental setup. In order to simulate the client request, we build a data generator to simulate the data flow in ICU which generate requests at a frequency of 250qps from a client node. We connect the serving node and client node via HTTP and RPC. The serving node can start RPC call to client node to start a simulation to generate client requests. The data generated will then be sent by the client node and captured by the HTTP server. The data captured is passed on and stored inside accumulators running on the serving node (Figure 3). When the observation window is reached the data will then be further passed on for inference to the ensemble queue. The ensemble models dequeue queries and perform prediction on query data. The end-to-end latency that we profile is the latency of serving system from the moment the data is captured by HTTP server until the prediction results are finished by the ensemble ³.

4.2 End-to-end HOLMES Performance

In this section we evaluate the performance of HOLMES' ensemble composition component with respect to the prediction accuracy and latency. We report prediction accuracy in terms of area under receiver operating characteristic curve (ROC-AUC), area under precision recall curve (PR-AUC), F1 score, and accuracy. We report system efficiency in terms of latency (second). We compare HOLMES with the following baselines:

- **Random (RD)** iteratively chooses one random single model from model zoo without replacement, and adds it to the current model set, till the ensemble model exceeds latency constraint.
- **Accuracy First (AF)** iteratively chooses the next most accurate single model from model zoo, and adds it to the current model set, till the ensemble model exceeds latency constraint.

³Our code can be found at <https://github.com/hsd1503/HOLMES>

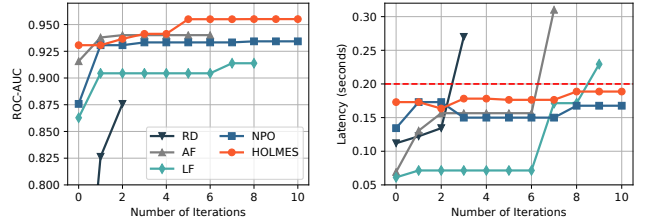


Figure 6: Search trajectory: accuracy (left) and latency (right) as a function of iteration.

- **Latency First (LF)** iteratively chooses the next lowest latency single model from model zoo, and adds it to the current model set, till the ensemble model exceeds latency constraint.
- **Non-Parametric Optimization (NPO)** (modified based on [32]) iteratively chooses a random subset B (size bounded by the number of models selected by LF) from model zoo, and merges them to the current model set, till the number of profiler calls exceeds the budget N ; returns the b^* that maximizes the objective function in Eq. (2) over the final explored model set.

For both NPO and HOLMES, we also add solutions from RD, AF and LF as their initial profiling. The budget N to profiler calls is the same for NPO and HOLMES. In HOLMES, we build two random forest [6] as the surrogate models for accuracy and latency.

4.2.1 Overall performance compared with baselines. Table 2 summarizes the prediction performance from all methods under a 200ms latency constraint. HOLMES achieves the best performance for all measurements within the same latency constraint.

Table 2: Comparison results.

Method	ROC-AUC	PR-AUC	F1	Accuracy
RD	0.8758 \pm 0.1334	0.8198 \pm 0.2404	0.6887 \pm 0.2246	0.7760 \pm 0.1311
AF	0.9307 \pm 0.0862	0.9025 \pm 0.0791	0.7426 \pm 0.2920	0.8526 \pm 0.1113
LF	0.9135 \pm 0.1020	0.8755 \pm 0.1093	0.8302 \pm 0.1387	0.8695 \pm 0.1083
NPO	0.9343 \pm 0.0741	0.9078 \pm 0.1418	0.8237 \pm 0.1828	0.8756 \pm 0.0941
HOLMES	0.9551 \pm 0.0521	0.9349 \pm 0.0834	0.8501 \pm 0.1054	0.8837 \pm 0.0815

4.2.2 Tradeoff space exploration efficacy. First, we track the search trajectory through the accuracy/latency tradeoff space in Figure 6, tracking incremental changes in accuracy (Left) and latency (Right). HOLMES is able to quickly reach the 200ms latency constraint, but continuing model selection, packing a more accurate model ensemble within the latency budget. RD, AF, and LF stop after exceeding the latency budget (higher than the 200ms horizontal line) and don't reach optimality w.r.t. AUC-ROC scores. NPO search trajectory stays under the latency budget, but doesn't reach optimal ROC-AUC, due to inefficient random exploration.

Second, we show that HOLMES finds ensembles with accuracy/latency on the Pareto frontier of the tradeoff space. Namely, for a range of fixed latency budgets, HOLMES consistently composes ensembles that outperform NPO (the highest performing baseline) w.r.t. ROC-AUC (Figure 7). Furthermore, NPO ROC-AUC variance is higher due to the unstable random search/exploration. In contrast HOLMES produces narrower ROC-AUC distribution as the ensemble models it explores have smaller ROC-AUC variance.

Third, we validate the efficacy of our exploration algorithm in HOLMES by showing the two surrogate models improve when the

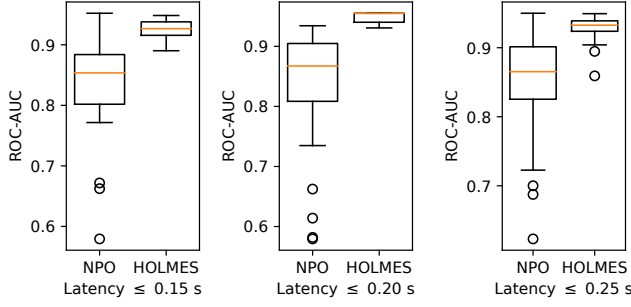


Figure 7: HOLMES overall finds more accurate ensemble models than NPO under different latency constraints.

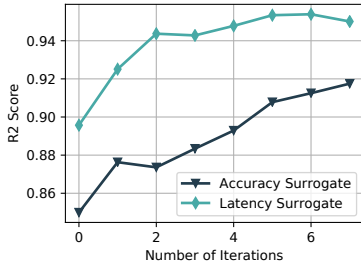


Figure 8: Surrogate models' performances increasing with the number of profiler interactions.

number of exploration steps increases. In Figure 8, we plot R_2 score at each iteration step. R_2 measures the differences between the predicted accuracy/latency by the surrogate model and the true accuracy/latency profiled by the profiler on an independent validation set that has not been explored by the algorithm. We can tell both models' predictions get improved as their R_2 scores increase. This result explains why HOLMES finds more accurate ensemble models than NPO, even if they have the same number of iterations – HOLMES can explore and predict more promising ensemble models using surrogates before actually profiling them.

4.3 Ensemble Serving System Performance

4.3.1 Online vs. offline inference latency. We now evaluate the serving system performance. First, we demonstrate the benefit of using an online serving system relative to the conventionally adopted approach of periodically re-evaluating patient condition offline. In Figure 9, we plot the ensemble query latency over time for HOLMES and the conventional batching method commonly adopted in hospital ICUs for patient discharge evaluation.

The experiment is carried out for a single patient over the period of 60 minutes. There's a single periodic spike at the end of the 60min time window corresponding to the latency of evaluating patient's condition on accumulated stale data. This approach affects both the accuracy (see Figure 2) and latency. It can be seen that the batching approach incurs inference latency that's an order of magnitude higher than HOLMES (log y-axis).

HOLMES, on the other hand performs evaluation every 30s, corresponding to the periodic spikes up to an order of magnitude of 10^{-1} s. The smaller latency in between the spikes correspond to just the sensory data collection part of the pipeline (Figure 4). As

the system is well-provisioned, we expect the online sensory data collection and aggregation latency not to diverge and remain approximately at the same level, in this case, at the order of magnitude of 1-10ms. For the purposes of this evaluation, the highest accuracy model was chosen as the prediction model.

Thus, we observe that by reducing data staleness, we not only gain in terms of accuracy but also achieve an order of magnitude faster inference latency for patient condition re-evaluation.

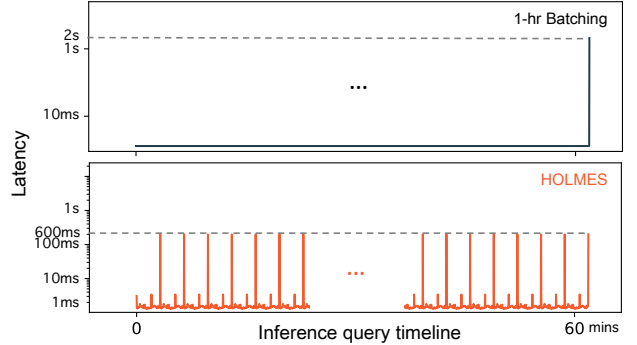


Figure 9: Comparison of an end-to-end timeline from the batching solution (every 1 hour) and HOLMES.

4.3.2 Varying resource constraints. We use the ensemble models selected by HOLMES to perform the scalability experiments.

In Figure 10 (left) we vary throughput by changing the number of patients and keeping the number of GPUs fixed (2 NVIDIA V100 GPUs). We generate a throughput of 250 qps from each patient in an open loop arrival process (i.e., not blocking on the results of prior queries). Throughput is then linear in the number of patients. Throughput increase contributes to (a) higher queueing delays and (b) more contention for the GPUs, increasing the end-to-end system response T plotted on the y-axis.

As the number of patients increases, the throughput gets higher. Higher throughput leads to higher queueing delays which increases latency. HOLMES serving system is able to perform inference of a 10 model ensemble within 1.15 seconds (95th percentile) even when the ingest throughput is high (64 patients * 250 qps = 16000 queries per second). In Figure 10 (right), we vary number of GPUs, keeping throughput fixed (16000 qps) to gauge its effect on latency. Due to less resource contention, ensemble served on two GPUs provides a lower latency compared to one GPU.

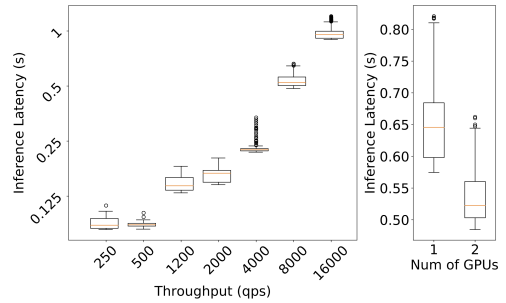


Figure 10: Latency scalability. (Left): latency scales linearly with ingest rate from an increasing number of patients. (Right): latency improves with more GPUs.

5 CONCLUSION

ICU environment necessitates both timely and accurate decisions to manage patients' standard of care and cost of care. Clinical tasks, such as timely patient discharge, require expert medical staff. Deep learning models have achieved high accuracy of prediction on these tasks, but neglected latency implications. In this paper, we propose HOLMES—an online model ensemble serving system for DL models in ICU. HOLMES navigates the accuracy/latency tradeoff space, achieving the highest accuracy within specified sub-second latency targets. HOLMES's Ensemble Composer automatically constructs a model ensemble from a set of models trained for different sensory data modalities and, potentially, observation windows. HOLMES provides a real-time serving platform built on top of an open-source Ray [25] framework, with support for serving pipelines that consist of a combination of stateful (e.g., data aggregation) and stateless (e.g. model serving) components to achieve sub-second, latency SLO-aware performance. HOLMES is shown to outperform the conventional offline batched inference both on clinical prediction accuracy and latency (by order of magnitude).

6 ACKNOWLEDGEMENTS

This work was in part supported by the National Science Foundation award IIS-1418511, CCF-1533768 and IIS-1838042, the National Institute of Health award NIH R01 1R01NS107291-01 and R56HL138415.

REFERENCES

- [1] Sébastien Bailly, Geert Meyfroidt, and Jean-François Timsit. 2018. What's new in ICU in 2020: big data and machine learning. *Intensive care medicine* 44, 9 (2018), 1524–1527.
- [2] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of machine learning research* 13, Feb (2012), 281–305.
- [3] James Bergstra, Daniel Yamins, and David Daniel Cox. 2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *JMLR* (2013).
- [4] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*. 2546–2554.
- [5] Leo Breiman. 1996. Bagging Predictors. *Mach. Learn.* 24, 2 (Aug. 1996), 123–140. <https://doi.org/10.1023/A:1018054314350>
- [6] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.
- [7] Han Cai, Ligeng Zhu, and Song Han. 2018. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv:1812.00332* (2018).
- [8] Leo Anthony Celi, Roger G Mark, David J Stone, and Robert A Montgomery. 2013. A Big data in the intensive care unit. Closing the data loop. *American journal of respiratory and critical care medicine* 187, 11 (2013), 1157.
- [9] Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2016. Clipper: A Low-Latency Online Prediction Serving System. *CoRR abs/1612.03079* (2016). [arXiv:1612.03079](http://arxiv.org/abs/1612.03079) <http://arxiv.org/abs/1612.03079>
- [10] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2018. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377* (2018).
- [11] Hayley B Gershengorn, Allan Garland, and Michelle N Gong. 2015. Patterns of daily costs differ for medical and surgical intensive care unit patients. *Annals of the American Thoracic Society* 12, 12 (2015), 1831–1836.
- [12] Varun Gulshan, Lily Peng, Marc Coram, Martin C. Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, Ramasamy Kim, Rajiv Raman, Philip C. Nelson, Jessica L. Mega, and Dale R. Webster. 2016. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. *JAMA* 316, 22 (12 2016), 2402–2410. <https://doi.org/10.1001/jama.2016.17216> [arXiv:https://jamanetwork.com/journals/jama/articlepdf/2588763/joi160132.pdf](https://jamanetwork.com/journals/jama/articlepdf/2588763/joi160132.pdf)
- [13] Neil A Halpern and Stephen M Pastores. 2015. Critical care medicine beds, use, occupancy and costs in the United States: a methodological review. *Critical care medicine* 43, 11 (2015), 2452.
- [14] Awni Y Hannun, Pranav Rajpurkar, Masoumeh Haghpanahi, Geoffrey H Tison, Codie Bourn, Mintu P Turakhia, and Andrew Y Ng. 2019. Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. *Nature medicine* 25, 1 (2019), 65.
- [15] Hrayr Harutyunyan, Hrant Khachatrian, David C Kale, Greg Ver Steeg, and Aram Galstyan. 2017. Multitask learning and benchmarking with clinical time series data. *arXiv preprint arXiv:1703.07771* (2017).
- [16] Shenda Hong, Yuxi Zhou, Junyuan Shang, Cao Xiao, and Jimeng Sun. 2020. Opportunities and challenges of deep learning methods for electrocardiogram data: A systematic review. *Computers in Biology and Medicine* (2020), 103801.
- [17] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*. Springer, 507–523.
- [18] Sławomir Koziel and Leifur Leifsson. 2013. *Surrogate-based modeling and optimization*. Springer.
- [19] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.
- [20] Julien-Charles Lévesque, Christian Gagné, and Robert Sabourin. 2016. Bayesian hyperparameter optimization for ensemble learning. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*. 437–446.
- [21] Zachary C Lipton, David C Kale, Charles Elkan, and Randall Wetzel. 2016. Learning to diagnose with LSTM recurrent neural networks. *ICLR*.
- [22] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive neural architecture search. In *ECCV*. 19–34.
- [23] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. 2015. MLlib: Machine Learning in Apache Spark. [arXiv:1505.06807](https://arxiv.org/abs/1505.06807) [cs.LG]
- [24] Jonas Mockus. 2012. *Bayesian approach to global optimization: theory and applications*. Vol. 37. Springer Science & Business Media.
- [25] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. 2018. Ray: A distributed framework for emerging {AI} applications. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 561–577.
- [26] Phuoc Nguyen, Truyen Tran, and Svetha Venkatesh. 2017. Deep learning to attend to risk in ICU. *arXiv preprint arXiv:1707.05010* (2017).
- [27] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. 2017. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv:1712.06139* (2017).
- [28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*. 8024–8035.
- [29] Alvin Rajkomar, Eyal Oren, Kai Chen, Andrew M Dai, Nissim Hajaj, Michaela Hardt, Peter J Liu, Xiaobing Liu, Jake Marcus, Mimi Sun, et al. 2018. Scalable and accurate deep learning with electronic health records. *NPJ Digital Medicine* 1, 1 (2018), 18.
- [30] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Ding, Aarti Bagul, Curtis Langlotz, Katie Shpanskaya, et al. 2017. CheXnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. *arXiv preprint arXiv:1711.05225* (2017).
- [31] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE* 104, 1 (2015), 148–175.
- [32] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*. 2951–2959.
- [33] Lu Wang, Wei Zhang, Xiaofeng He, and Hongyuan Zha. 2018. Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2447–2456.
- [34] Darrell Whitley. 1994. A genetic algorithm tutorial. *Statistics and computing* 4, 2 (1994), 65–85.
- [35] Cao Xiao, Edward Choi, and Jimeng Sun. 2018. Opportunities and challenges in developing deep learning models using electronic health records data: a systematic review. *JAMIA* 25, 10 (2018), 1419–1428.
- [36] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1492–1500.
- [37] Yanbo Xu, Siddharth Biswal, Shripasad R Deshpande, Kevin O Maher, and Jimeng Sun. 2018. Raim: Recurrent attentive and intensive model of multimodal patient monitoring data. In *KDD*. ACM, 2565–2573.
- [38] Zhi-Hua Zhou. 2012. *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC.
- [39] Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).

SUPPLEMENTARY MATERIALS

A.1 Exploring in HOLMES using genetic search

Algorithm 2 Explore using Genetic algorithm

```

1: Input:  $B$ , number of samples  $N_1$ , degree of mutation  $S$ , probability of genetic explore  $p$ , probability of mutation  $p_1$ .
2: Output:  $B'$ 
3: while  $i < N_1$  do
4:   Uniformly random pick two numbers  $rnd, rnd_1$  from  $[0,1]$ 
5:   Random pick  $b_1, b_2, b_3$  from  $B$ 
6:   if  $rnd > p$  then
7:     /* Random explore */
8:      $b = \text{Random}(B)$ 
9:   else
10:    if  $rnd_1 > p_1$  then
11:      /* Recombination explore */
12:       $b = \text{Recombination}(b_1, b_2)$ 
13:    else
14:      /* Mutation explore */
15:       $b = \text{Mutation}(b_3, S)$ 
16:    end if
17:  end if
18:  /* Not add duplicates */
19:  if  $b \in B$  or  $b \in B'$  then
20:    continue
21:  else
22:     $i = i + 1$ 
23:    /* Add it to candidates */
24:     $B' = B' \cup b$ 
25:  end if
26: end while

```

A.2 Evolution of ROC-AUC and latency when exploring the searching space using different algorithms

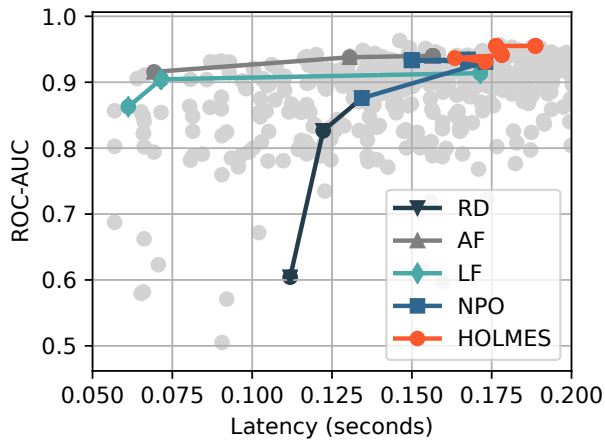


Figure 11: ROC-AUC vs. Latency vary by iterations in different Explore algorithms

A.3 Comparison of accuracy and latency from the optimal ensemble selected by different algorithms

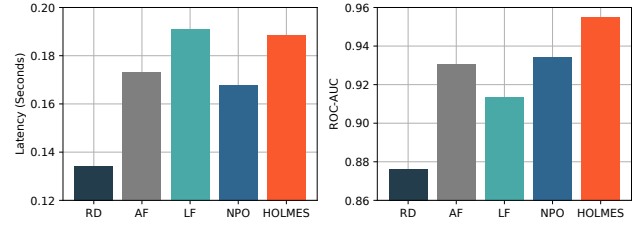


Figure 12: Comparing under 0.2 second latency constraint. (Left): HOLMES has comparable utility of latency with LF (Latency First) method. (Right): HOLMES selects higher accurate ensemble model under that latency constraint.

A.4 Change of accuracy and latency by varying the observation window

In Figure 13, we do comparison of execution in latency profile to show how does increase in observation window causes a small increase in latency while leads to high latency profile. Our right graph shows multiple execution of a model inside HOLMES serving system. Timeit (Time in PyTorch) legend shows the execution of plain PyTorch model in a GPU. TS (serving delay) legend shows the execution of the model inside the serving system. TQ (queuing delay) legend shows the worst analysis of a request waiting in a queue inside the serving system. TQ + TS legend shows end-to-end latency for a query inside the serving system.

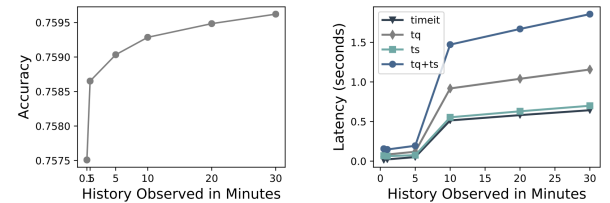


Figure 13: Effects of different history aggregation.

A.5 Deep model description in the Model Zoo.

Table 3: Deep model description in the Model Zoo.

Field	Description
Depth	Number of stacked layers
Width	Number of convolutional filters
MACS	Multiply-accumulate operations
Memory size	GPU memory usage
Input data modality	ECG Lead number or vital sign names
Input data length	Length of each input signal segmentation
Accuracy	ROC-AUC on validation set

A.6 An Alternative Formulation of Accuracy Sensitive Constraint.

In the main paper, as desired for a real-time serving system, we have formulated our problem as a latency sensitive task that aims to maximize accuracy subject to a latency upper bound L . Alternatively, for other accuracy sensitive tasks, we can switch the objective function and constraint in Eq. (1) and reach at a new

Lagrange function:

$$\min_{\mathbf{b} \in \{0,1\}^n} L_l(\mathbf{b}) = f_l(\mathbf{V}, \mathbf{c}, \mathbf{b}) - \delta(f_a(\mathbf{V}, \mathbf{b}) - A),$$

where A is the accuracy lower bound that a model ensemble needs to achieve at least. Although this setup is beyond the scope of this paper, it can be equivalently solved by following the same searching algorithm we propose in the following subsection.