Oikonomos-II+: A Reinforcement-Learning, Cloud-Resource Recommender for HPC & AI Workloads

River Betting, Qilin Chen, Chris De Zeeuw, and Christos Strydis, *Senior Member, IEEE*

***Abstract*—Oikonomos-II+ is a hybrid, reinforcement-learning system for recommending optimal cloud-instance types for High-Performance Computing (HPC) and Artificial-Intelligence (AI) applications. Unlike existing approaches that require historical data or repeated job executions, Oikonomos-II+ learns online using user-submitted jobs. It combines a modified Neural-LinUCB algorithm with Gaussian-Process regression to model the relationship between job parameters, instance types, and execution time. This allows it to balance exploration and exploitation efficiently, even in the absence of prior data. We evaluated six configurations of Oikonomos-II+ on a diverse set of HPC and AI workloads, optimizing for cost and speed. Results show that the complete system converges to optimal resource choices, outperforming purely predictive or search-based approaches. By treating deployed applications as a black box and by eliminating the need for preexisting training data or auxiliary runs, Oikonomos-II+ provides a general-purpose, low-overhead solution for dynamic resource selection in heterogeneous cloud environments.**

***Index Terms*—High-performance computing (HPC), artificial intelligence (AI), machine learning, resource recommendation, cloud computing, prediction, middleware, deep learning, Gaussian processes.**

I. Introduction

MODERN cloud platforms, such as Amazon EC2 and Microsoft Azure, now offer a vast array of on-demand, pay-per-use computing resources, often without queuing delays. For example, Amazon EC2 currently offers over 800 instance types [1]. Despite their categorization into families and transparent pricing, selecting the optimal configuration for a given HPC and AI workload remains a challenge. Execution time depends on both input parameters and hardware characteristics in complex, often non-obvious ways [2].

Existing recommender systems attempt to aid this process but suffer from key limitations: search-based systems require costly, repeated-job executions [10], [12], while prediction-based systems rely on extensive historical data [7], [8], which may not exist for new applications or workloads.

We introduce Oikonomos-II+, a *general-purpose, reinforcement-learning (RL) recommender system* for selecting optimal instance types in heterogeneous cloud environments. Oikonomos-II+ treats instance selection as a contextual multi-armed bandit problem and learns online from real user-submitted jobs. It combines a modified Neural-LinUCB algorithm [5] with Gaussian-Process (GP) regression [29] to model the relationship between job parameters, hardware configurations, and performance. Unlike prior methods, Oikonomos-II+ requires *no* prior training data and *no* auxiliary executions, making it cost-efficient and broadly applicable.[1]

Building on our previous work, Oikonomos [4], which used a purely predictive multilayer perceptron (MLP), Oikonomos-II+ introduces several algorithmic and architectural innovations to enhance robustness, scalability, and learning efficiency. The main contributions of this work are:

- A novel, hybrid recommender system (Oikonomos-II+) that combines deep contextual bandits and Gaussian-Process (GP) regression to dynamically recommend cloud resources for HPC workloads.
- Three enhancements to the Neural-LinUCB algorithm [5] that improve compatibility with deep networks: soft updates, prioritized training using RHO-LOSS [30], and latent-space regularization via a Variational Autoencoder (VAE) [31].
- A unified exploration-exploitation mechanism, where inverse-variance weighting balances GP and neural predictions, enabling efficient convergence without historical data or repeated job executions.

Received 4 August 2025; revised 17 February 2026; accepted 30 March 2026. Date of publication 9 April 2026; date of current version 9 June 2026. This work was supported in part by the European Union's Horizon Europe Research and Innovation Programme through Project SEPTON under Grant 101094901 and Project SECURED under Grant 101095717, in part by the Dutch Research Council's Gravitation Programme through Project DBI^2 under Grant 024.005.022, and in part by SURF Cooperative under Grant EINF-13316. Recommended for acceptance by T. Bicer. *(Corresponding author: Christos Strydis.)*

River Betting and Qilin Chen are with the Department of Neuroscience, Erasmus Medical Center, 3015 GD Rotterdam, The Netherlands (e-mail: j.betting@erasmusmc.nl; q.chen@erasmusmc.nl).

Chris De Zeeuw is with the Department of Neuroscience, Erasmus Medical Center, 3015 GD Rotterdam, The Netherlands, and also with the Netherlands Institute for Neuroscience, 1105 BA Amsterdam, The Netherlands (e-mail: c.dezeeuw@erasmusmc.nl).

Christos Strydis is with the Department of Neuroscience, Erasmus Medical Center, 3015 GD Rotterdam, The Netherlands, and also with the Quantum & Computer Engineering Department, Delft University of Technology, 2628 CD Delft, The Netherlands (e-mail: c.strydis@erasmusmc.nl).

Digital Object Identifier 10.1109/TCC.2026.3682234

[1]A preliminary version of this paper appeared as 'Oikonomos-II: A Reinforcement-Learning, Resource-Recommendation System for Cloud HPC' [3].

- A detailed evaluation across six diverse applications, covering both HPC and AI training workloads, and analyzing performance under time and cost optimization goals.
- An open-source dataset and simulation framework that enables reproducible benchmarking of cloud-HPC resource recommendation systems.[2]

The remainder of this paper is structured as follows: Section II reviews related work and positions Oikonomos-II+ within the landscape of cloud-HPC recommendation systems. Section III details the architecture and learning algorithms used in Oikonomos-II+. Section IV describes the implementation and experimental setup. Section V presents and analyzes evaluation results across different configurations and workloads. Section VI discusses the system's implications and potential extensions. Finally, Section VII concludes the paper.

II. Related Work

HPC and AI workloads differ from other workloads in a variety of ways. First of all, HPC and AI workloads often require large amounts of computational resources to run efficiently. These workloads tend to benefit from accelerators such as Graphics Processing Units (GPUs), Tensor Processing Units (TPUs), and Field-Programmable Gate Arrays (FPGAs), and be highly parallelized. Furthermore, the same application is run many times with a variety of workloads. For instance, a neural simulator such as SimHH is able to run simulations of a single neuron, as well as simulations of large networks of thousands of neurons. The optimal hardware configuration is often highly dependent on the parameters and input of the workload. This makes it challenging to tailor the computational resources to individual workloads.

A quantitative comparison of existing work in the field of cloud-HPC resource recommendation is challenging, as different publications use their own metrics and HPC applications to validate their work. Publications often rely on datasets generated in their own lab environment, which is not accessible to others. Furthermore, other works sometimes use different settings or make different assumptions about the workloads and the problem at hand. Therefore, our comparison with other works will be qualitative. However, we will compare the influence of various subcomponents of our algorithm quantitatively.

Work in the field of cloud-HPC resource recommendation generally falls into one of two categories: search-based algorithms and prediction-based algorithms. Search-based algorithms evaluate different hardware combinations in succession to find the optimal choice. These algorithms do not rely on earlier data but usually need to run a job multiple times to find an optimal instance type; this leads to extra costs. Prediction-based algorithms use offline evaluation of data to predict performance and can immediately suggest an optimal instance type. This removes the need to actively search, but these algorithms require prior knowledge about the behavior of the application in the form of a model or historical data. A summary of related work can be found in Table I. Works are assessed based on (1) the need for prior data for training, (2) the need to explore, possibly by executing workloads multiple times, (3) the flexibility in assessing workloads for different hardware configurations, and (4) the user effort needed to make good use of the system.

Yang et al. [6] take a prediction-based approach. They argue that most HPC applications are time-step based, iterative, and very predictable when it comes to performance. This means that the total execution time of an application on a specific platform can be inferred from a partial execution of that application combined with prior knowledge about the application. This *observation-based approach* gives accurate predictions for applications for which the assumption of predictability holds, but the authors also admit that variation in the execution time per timestep leads to a much lower accuracy. Furthermore, their approach was exclusively tested on CPU-only platforms: accelerators such as GPUs, which may strongly influence application behavior, are not taken into account.

Ernest [7] is a prediction-based framework that works with a non-negative least-squares solver, using historical data about the size of the input data, the number of virtual machines used, and the execution time to fit four variables to a formula. This is then used to predict execution times, and can be extended to include more parameter values. However, Ernest is less suitable if the application behavior is unknown. It is also unsuitable for heterogeneous hardware configurations, since it only takes the number of machines into account.

Daleel [8] is a prediction-based framework to support decision making in Infrastructure-as-a-Service (IaaS) environments, using a multivariate polynomial model to predict execution times, similar to Ernest's formula-fitting approach. The number of vCPUs, RAM, and the day of the week are used as input parameters. Daleel achieved low Mean Squared Error, but is less suitable for heterogeneous-hardware configurations or complex relationships between input parameters and execution time.

PARIS [9], another prediction-based approach, decouples instance performance characterization from the workload-specific resource requirements by profiling the instance types using a set of benchmark workloads once. The user chooses and runs a representative workload to analyze the resource usage patterns to create a fingerprint of the application, in order to recommend an instance type based on the user's needs. The decoupling of application characteristics from instance-type characteristics is important, but PARIS burdens the user with choosing a representative workload, and does not take the influence of application parameter values on resource usage patterns into account.

CherryPick [10] is a search-based approach that uses Bayesian optimization to build a performance model for applications. CherryPick has similar performance to Ernest when it comes to running costs, but with lower search time and cost. However, it still needs to run a workload several times, and like PARIS, burdens the user with providing representative workloads.

Scout [11] is a search-based approach that uses past performance information to search efficiently. A key insight from Scout is that any search-based algorithm has a trade-off between exploration and exploitation. *Arrow* [12], like CherryPick, uses Bayesian optimization, but augments it with low-level metrics

[2]The Oikonomos-II+ code can be found at: https://gitlab.com/c7859/neurocomputing-lab/oikonomos-II_data.

TABLE I
QUALITATIVE COMPARISON OF RELATED SYSTEMS FOR CLOUD-HPC INSTANCE SELECTION

Work	Approach	Prior-Data Requirement	Exploration Requirement	Hardware Flexibility	User Effort
Yang et al. (2005) [6]	Prediction-based	Moderate	Moderate	Low	Moderate
Ernest (2016) [7]	Prediction-based	High	None	Low	Low
Daleel (2016) [8]	Prediction-based	High	None	Low	Low
PARIS (2017) [9]	Prediction-based	High	None	Moderate	High
CherryPick (2017) [10]	Search-based	None	High	High	High
Scout (2018) [11]	Search-based	Low	Moderate	Moderate	Low
Arrow (2018) [12]	Search-based	Low	Moderate	Moderate	Moderate
Micky (2018) [13]	Search-based	None	Moderate	Moderate	Low
Mariani et al. (2018) [14]	Prediction-based	High	Moderate	Low	High
Brunetta & Borin (2019) [15]	Prediction-based	Moderate	Moderate	Moderate	Moderate
Tamakkon (2020) [16]	Prediction-based	High	None	Low	Low
A2Cloud-RF (2020) [17]	Prediction-based	Moderate	None	Moderate	Moderate
A2Cloud-H (2022) [18]	Prediction-based	Moderate	None	Moderate	Moderate
Lamar et al. (2023) [19]	Prediction-based	High	None	Low	Low
Oikonomos (2023) [4]	Prediction-based	High	None	High	Low
Oikonomos-II+ (this work)	Hybrid (RL)	None	Low	High	Low

in order to reduce search costs. *Micky* [13] casts the problem of finding the best VM as a multi-armed bandit problem and uses the Upper Confidence Bound (UCB) algorithm to optimize rewards. All of these approaches address some of the problems of search-based algorithms, but still require running a workload multiple times to find the best configuration, which implies additional costs.

Mariani et al. [14] proposed a prediction-based approach that combines a *cloud-performance-prediction model* (CP) provided by the cloud provider with a *profile-prediction model* (PP) provided by the user. CP is generated by the cloud provider based on a set of training applications representative of applications that the users may be interested in, a training dataset, and available cloud configurations. PP is generated by the user on their own hardware, and coupled with CP in order to predict the best cloud configuration. The authors claim that the approach is hardware-independent, but do not take accelerators such as GPUs into account. Furthermore, the approach requires a lot of data and effort from both the cloud provider and the users.

Brunetta & Borin [15] made the same observation as Smaragdos et al. [2]: the optimal hardware configuration of HPC applications is highly dependent on the input datasets and parameter values. Building on Yang et al. [6], this approach assumes that most of the execution time of the HPC application is spent in iterations. To determine the best instance type choice, the job is partially executed on each cloud configuration. The first stable iterations of the job are used to select the best cloud configuration. This approach takes the specific characteristics of the job into account, but still makes assumptions about the iterative nature of the application. Furthermore, the approach was not tested on instances that make use of GPUs or other accelerators, even though applications may behave fundamentally differently on different types of hardware.

Tamakkon [16] uses historical performance data for resource recommendation of new applications or VM types, based on their similarity to known applications. This makes the algorithm useful for different applications and hardware configurations. The system does require the production of auxiliary data in the cloud, which entails additional costs. Also, Tamakkon simply labels workloads as ‘similar’ or ‘partly similar’, but does not further specify in which way the workloads are similar.

A2Cloud-RF [17] is a prediction-based approach which, like PARIS, decouples the characteristics of the applications and cloud instances by profiling them separately. A Random-Forest Classifier (RFC) directly classifies instance types as ‘excellent’, ‘good’, ‘okay’, or ‘bad’, based on these profiles. This is useful, but given the huge amount of available instance types, a classification into four categories is rather rough. Decoupling applications and instance types reduces the need for test runs, but also makes it more difficult to capture the complex interplay between application performance, resource use, and available hardware.

A2Cloud-H [18] is an expanded version of A2Cloud-RF. Rather than only using an RFC, A2Cloud-H uses a variety of machine learning algorithms, divided into two modules: an unsupervised learning module (USL) and a supervised learning module (SL). Both modules are contained in a decision module, which selects an algorithm from both modules. Even though offering a variety of algorithms might make the system more generalizable, it also makes it more complex: it creates the need for an additional algorithm to select a recommender algorithm.

Lamar et al. [19] explored HPC application run time prediction on an HPC cluster, taking application-specific input parameters into account. Five applications were evaluated against 20 machine-learning models, three traditional models, and an analytical model. It was found that using application-specific input parameters as features often leads to excellent predictions. However, this approach relies on a lot of data. Furthermore, jobs were only generated on a cluster, not in a cloud. When it comes to resources, only the number of nodes and the number of tasks were varied; resource heterogeneity was not taken into account.

Our previous work, *Oikonomos* [4], uses a multilayer perceptron trained on historical cloud HPC job data to predict execution times from input parameters and instance characteristics. It demonstrated that general neural predictors can be effective for HPC job recommendation. Oikonomos does take heterogeneity into account and includes instance types with available GPUs. However, Oikonomos relies on a large amount of historical data, which might not be practical or available, especially for new applications.

In summary, in the prediction-based approaches, there tends to exist a trade-off between more specific modeling (for instance by fitting a formula or by fingerprinting) and a reliance on considerable amounts of data. For search-based approaches, there is a trade-off between exploration and exploitation: more exploration might lead to better recommendations, but will also lead to higher overhead costs, whereas early exploitation will lead to lower overhead costs but might make the recommendations less accurate.

The work we present in this paper, *Oikonomos-II+*, like Oikonomos, has an MLP at its core, and uses data it gathers. However, like Micky, we approach the problem of cloud resource recommendation as a multi-armed bandit problem, in order to explore the search space for giving better recommendations. Oikonomos-II+ can be seen as a hybrid approach, combining the advantages of search-based and prediction-based algorithms. In this way, it overcomes the limitations of earlier approaches. A preliminary version of this algorithm appeared as Oikonomos-II [3]. Oikonomos-II+ extends this work by adding a GP prediction algorithm and a VAE. Both these additions improve performance. Furthermore, Oikonomos-II+ enables prioritized training using the RHO-LOSS algorithm, which reduces the need for MLP retraining. Additionally, the algorithm was validated on new and more realistic datasets.

III. Design

As the extensive related-work section demonstrates, there is a need for a middleware layer for resource recommendation in a heterogeneous HPC system that aids the user in selecting the hardware platform that is best-suited for the job they need to run. We *avoid* performance-model construction, as the complex interplay between the application parameters and the execution platform calls for an application-agnostic approach. Defining 'job' as the (requested) execution of the application with specific parameter values, we assume a sequential stream of jobs, with each round t corresponding to the recommendation of an instance type and subsequent execution of job j_t. Job j_t is defined by a vector $\boldsymbol{p}_t$ of parameter values. Furthermore, we assume a set of available instance types s; for every s, there is a vector $\boldsymbol{h}_s$ containing hardware-parameter values of the instance type, such as the number of vCPU cores, memory size, GPU type, etc. We make the assumption that each job j_t can start only when job j_{t-1} has finished. Each action $a \in \mathcal{A}$ is the act of assigning a job to an instance type. Action $a_{s,t}$ signifies the act of assigning job j_t to instance type s for execution.

In contrast to recommenders like Oikonomos, we assume the absence of any preexisting historical execution data. Because of this, Oikonomos-II+ is forced to take risks by recommending instance types it has not encountered before. At the same time, though, Oikonomos-II+ has to optimize performance for its users. This dilemma is known as the exploration-exploitation dilemma, which is a general problem to be found in data-driven, decision-making processes where a feedback loop exists between data gathering and decision making [20]. This becomes most clear in a class of problems known as multi-armed bandit (MAB) problems.

In this section, we explain the multi-armed bandit problem. Then, we describe our implementation of Neural-LinUCB, which forms the basis of Oikonomos-II+. After that, we explain several additional modules that were added to Oikonomos-II+: the GP predictor and the VAE, both of which enhance Oikonomos-II+'s performance; and the RHO-LOSS algorithm, which reduces retraining time.

A. The Multi-Armed Bandit Problem

Lattimore and Szepesvári [21] describe the bandit problem as a sequential game between a *learner* and an *environment*. Played over n rounds, for each round $t \in [n]$, the learner picks an *action* a_t from a set of actions $\mathcal{A}$. After the action is chosen, the environment reveals a *reward* $r_t \in \mathbb{R}$. The learner does not get to see the rewards associated with the other actions. As it cannot see into the future, in the classical multi-armed bandit problem, it has to rely on the *history* $H_{t-1} = (a_1, r_1, \ldots, a_{t-1}, r_{t-1})$ in order to make decisions. The learner is expected to adopt a policy π: a mapping from histories to actions. Most commonly, the goal is for the learner to find a policy that maximizes the cumulative reward over all rounds $\sum_{t=1}^{n} r_t$. The *regret* r of a policy is defined as the difference between the cumulative expected reward using policy π and the cumulative reward of an optimal policy π^*. Cumulative regret will often grow in a logarithmic fashion for good policies: cumulative regret will increase relatively fast in the beginning, when there is little historical data and a strong need for exploration, and will slow down with time, as the amount of historical data grows, allowing for more exploitation. Bandit algorithms are part of the wider class of RL algorithms.

The bandit problem has been studied since the 1930s [22], but interest has skyrocketed over the last two decades because of its applicability in online environments. Dynamic pricing of online airplane bookings is a good example of a bandit problem: when a visitor searches for a flight, the website picks a price to offer to the visitor. The reward is revealed when the customer either books the flight or leaves without booking. The goal of the algorithm is to maximize total cumulative profit over all visitors [23].

Contextual knowledge can be essential for adopting a policy to make decisions. For instance, in the airplane-booking example, it might be useful to know the IP address of the visitor. After all, the visitor's location might be correlated to the price they are willing to pay. Context can also consist of similarity information regarding the actions in $\mathcal{A}$. A visitor might be willing to book a flight on a different date, or to a different airport, and showing them such options could lead to a higher chance of booking. Multi-armed bandit problems where context plays a role are known as *contextual bandits*.

Two of the most widely used algorithms for solving the exploration-exploitation dilemma in multi-armed bandit problems are Upper Confidence Bound (UCB) and Thompson Sampling (TS). UCB was first proposed by Auer et al. [24], and is based on the principle of *optimism in the face of uncertainty*. This means that the algorithm estimates the expected reward, as well as a confidence bound for each action, and chooses the action that has the highest upper confidence bound (see Fig. 2). TS, in contrast, builds a probability model based on previous rewards, and then samples from this model to choose an action [22]. Both TS and UCB are widely used and have strong theoretical guarantees on the regret bound.

B. Contextual MABs and Neural-LinUCB

The original UCB and TS algorithms do not take contextual information into account. However, they have been used as bases for algorithms that do work with contextual information. One of the most popular contextual-bandit algorithms is LinUCB, proposed by Li et al. [25]. The algorithm assumes a linear relationship between the context parameters and the rewards. The relationship is represented by a vector θ, which is to be learned using matrices $\boldsymbol{A}$ and $\boldsymbol{b}$, as shown in Fig. 3. LinUCB was presented in two versions: a disjoint version (where only one vector of context parameters is used) and a hybrid version (where two context vectors are used: one for parameters describing the context in round t, and one for parameters that describe the actions in $\mathcal{A}$). Li et al. applied the algorithm to personalized news-article recommendation and showed that it performs better than the original UCB algorithm. Likewise, Thompson Sampling has been used as the basis for the LinTS algorithm [26], where the vector θ is obtained using sampling.

The requirement of a linear relationship between context parameters and rewards in LinUCB is restrictive. For instance, in the case of cloud HPC, the relationship between application parameters, hardware, and execution time is potentially complex. This requirement, however, can be overcome using an artificial neural network (ANN). We will mention three relevant publications. Zhou et al. presented NeuralUCB, which feeds the context vector to a neural network [27]; NeuralUCB is a generalized version of LinUCB, achieving the regret bound of LinUCB without the aforementioned requirement. A similar adaptation for LinTS, NeuralTS, was presented by Zhang et al. [28]. However, as the whole network is used for exploration, the algorithm is very complex and computationally expensive for large neural networks. Addressing this issue, Xu et al. presented an adaptation where representation is decoupled from exploration [5]. Their algorithm, Neural-LinUCB, is based on the principle of *deep representation and shallow exploration:* it uses the entire ANN to learn the relationship between the context vectors and the rewards, but only uses the last layer for exploration. In this way, deeper and wider ANNs can be used, allowing for the representation of more complex context-reward relationships. Additionally, the way in which the relationship vector θ is calculated after each round is highly parallelizable, allowing for better performance. The authors showed that Neural-LinUCB achieves a performance similar to NeuralUCB, while being much less computationally expensive.

Betting et al. [4] showed with Oikonomos that a deep MLP can be used to recommend an optimal cloud-instance type for HPC applications, based on the input-parameter values. However, as Oikonomos was purely prediction-based, it relied on a large amount of preexisting training data. The absence of this data creates a contextual, multi-armed bandit problem. $\mathcal{A}$ consists of all possible instance-type recommendations, whereas the rewards are a function of execution time and/or usage costs. Each round t involves the decision to recommend an instance type to a specific job. The context, therefore, consists of both round-specific context (the input parameters of the job), as well as action-specific context (the hardware parameters of the instance types). The nonlinear relationship between context and rewards rules out traditional LinUCB. Because of the complexity and computational costs of NeuralUCB for deeper neural networks, as well as the opportunities for parallelism that Neural-LinUCB offers, we chose Neural-LinUCB over NeuralUCB.

C. Oikonomos-II+ Version of Neural-LinUCB

Fig. 1 shows a schematic overview of Oikonomos-II+. An adaptation of Neural-LinUCB forms the basis of the contextual multi-armed bandit system, which is shown in full in Fig. 4. We define the context vectors $\boldsymbol{x}_{s,t}$ as a concatenation of the vectors $\boldsymbol{h}_s$ and $\boldsymbol{p}_t$. Theoretically, it is possible to implement a hybrid version of Neural-LinUCB to evaluate $\boldsymbol{p}_t$ and $\boldsymbol{h}_s$ separately, but this is less parallelizable and computationally much more expensive, as each available arm would require its own matrices $\boldsymbol{A}_{a,t}$ and $\boldsymbol{b}_{a,t}$ that need to be calculated and stored. Furthermore, combining $\boldsymbol{h}_s$ and $\boldsymbol{p}_t$ in a single context vector allows the MLP to learn possibly complex interplays between hardware and application parameters.

In line with the original Neural-LinUCB and LinUCB algorithms, and as shown in Fig. 3, vector $\boldsymbol{q}_{s,t}$ (obtained by passing $\boldsymbol{x}_{s,t}$ through the MLP), is multiplied with vector θ_{t-1} to find the expected reward, and the inverse of $\boldsymbol{A}_{t-1}$ is used to calculate a standard deviation. These can be used directly to calculate

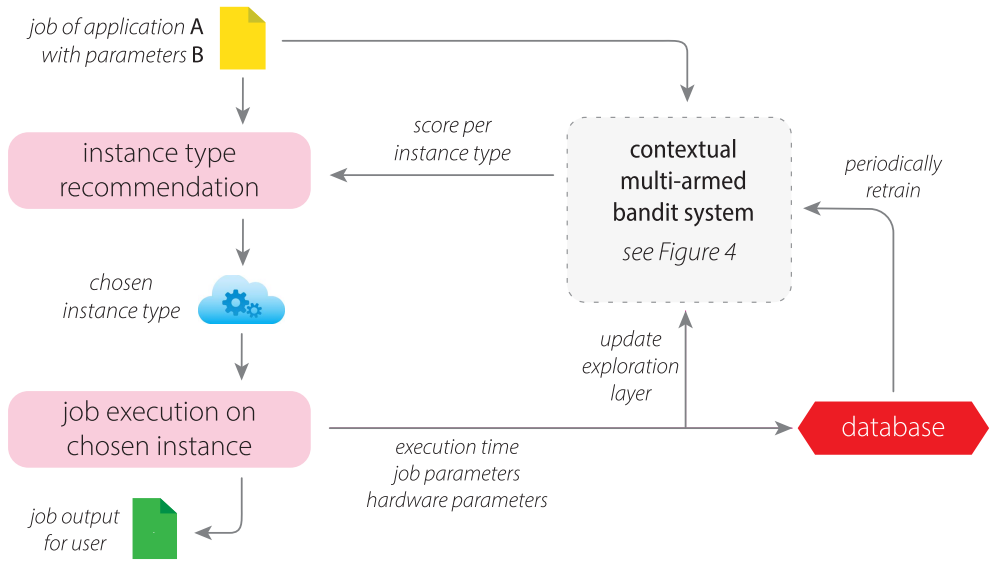

Fig. 1. *Schematic overview of Oikonomos-II+*: While the user gets the job output they want, the algorithm obtains the parameters of the job and its execution time and saves it in a database. A contextual bandit system is used to balance exploration and exploitation.

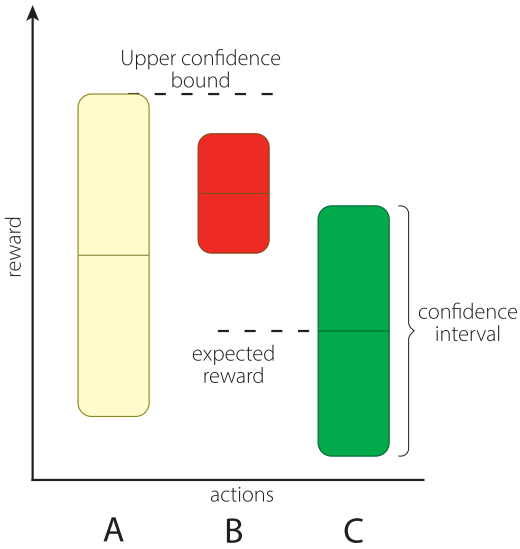

Fig. 2. *The UCB algorithm:* The expected reward is assessed for each option, as well as the confidence interval. The algorithm will choose the option with the highest upper confidence bound. Even though E$[r]$ is the highest for action B, the algorithm will choose action A, as its upper confidence bound is higher: *optimism in the face of uncertainty*.

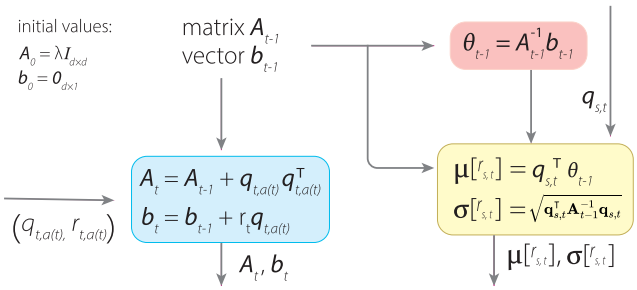

Fig. 3. *LinUCB strategy*: Matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$ are updated every episode t. $\boldsymbol{A}_{t-1}$ and $\boldsymbol{b}_{t-1}$ from the previous episode are used with ANN output vector $\boldsymbol{q}_{s,t}$ to generate an estimated reward $\mu[r_{s,t}]$ and confidence bound $\sigma[r_{s,t}]$ for the given job combined with every available instance type. The values of λ and d can be chosen by the user. In classical LinUCB and Neural-LinUCB, the confidence bound would be scaled and then added to the estimated reward. However, in our case, we pass them to the combined bandit algorithm separately. After an action $a(t)$ was chosen and a reward $r_{t,a(t)}$ was received, $\boldsymbol{A}_t$ and $\boldsymbol{b}_t$ can be calculated using $\boldsymbol{q}_{t,a(t)}$ and $r_{t,a(t)}$.

the upper confidence bound (as is done in the original Neural-LinUCB and LinUCB algorithms), or they can be combined with the output of a Gaussian Process Predictor (which will be introduced in Section III-F).

After the algorithm recommends an instance type, the job is executed there. The job output is then returned to the user. The reward $r_{t,a(t)}$, as well as vectors $\boldsymbol{p}_t$ and $\boldsymbol{h}_s$ are stored in a database. The ANN is retrained only periodically; it is unnecessary and computationally expensive to update the weights with each new data point. However, $\boldsymbol{A}_t$, $\boldsymbol{b}_t$, and $\boldsymbol{\theta}_t$ are calculated after every round, and are used for UCB calculation and instance-type recommendation in the next round. In other words, LinUCB-style exploration is decoupled from ANN retraining. In this way, the performance of the bandit algorithm is maintained.

We made several adaptations to the Neural-LinUCB algorithm to make it suitable for our application. The original Neural-LinUCB algorithm retrains the ANN every k steps. We noticed that the network requires frequent retraining in the beginning, and requires less frequent retraining later, when there is more data available. Therefore, rather than defining k as an integer, we define $\boldsymbol{k}$ as a vector of positive integers. If $t \in \boldsymbol{k}$, the ANN is retrained after round t. The size and content of $\boldsymbol{k}$ can be chosen by the user.

It was also noted that, when training the ANN, there exists a feedback loop between the weights of the ANN and the feature vector $\boldsymbol{\theta}$: after all, $\boldsymbol{\theta}$ depends on $\boldsymbol{A}$ and $\boldsymbol{b}$, and $\boldsymbol{A}$, $\boldsymbol{b}$

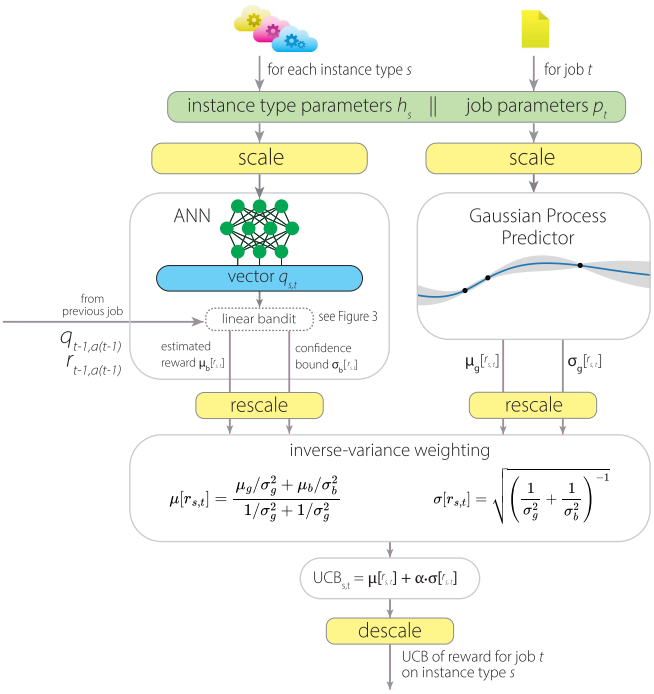

Fig. 4. *Combined bandit algorithm*: For each instance type s, a context vector $\boldsymbol{x}_{s,t}$ is created by concatenating job parameter vector p_t with instance type parameter vector h_s. The values are scaled and fed to a neural network to produce vector $\boldsymbol{q}_{s,t}$. $\boldsymbol{q}_{s,t}$ serves as input for a linear bandit algorithm, which produces an estimated reward and a standard deviation. In parallel, $\boldsymbol{x}_{s,t}$ is also scaled and fed to the Gaussian Process Predictor, which also produces an estimated reward and a standard deviation. The outputs of the linear bandit and the Gaussian Process Predictor are combined using inverse-variance weighting. This is used to calculate the UCB. The instance type s with the highest UCB for the given job t will be the one recommended by Oikonomos-II+.

are updated after every round using the MLP output vector $\boldsymbol{q}$. This led to instability and reduced performance during backpropagation. We resolved the issue by applying soft updating, as described by Lillicrap et al. [33], where the target network is used to recalculate θ for each data point at the start of each epoch, and backpropagation is applied to the training network. A soft-update step is performed at the end of each epoch. This allowed us to use deeper neural networks, which makes it possible to represent more complex relationships between inputs and rewards. We also improved the MLP training process by applying best-practice techniques, such as data scaling, training with mini-batches, and early stopping with separate training and validation sets in Oikonomos-II+.

D. Reducible Holdout Loss Selection

Even though Neural-LinUCB is computationally less expensive than NeuralUCB, its neural network periodically needs to be retrained, as the increasing amount of data allows for a more accurate representation of the relationship between parameters and rewards. However, repeatedly training the network anew on the whole dataset is wasteful and - since the dataset grows indefinitely as the application is used - is infeasible in the long term. Mindermann et al. [30] presented Reducible Holdout Loss Selection (RHO-LOSS), a data-selection algorithm that selects those data points for training that contribute most to reducing a network's generalization loss. RHO-LOSS first trains a network on a subset of the data: the *holdout set* $\mathcal{D}_{\text{ho}}$. For each labeled data point (x, y), the *irreducible holdout loss* (IL) $\mathcal{L}[y \mid x; \mathcal{D}_{\text{ho}}]$ can be calculated. While training the target model, the training loss $\mathcal{L}[y \mid x; \mathcal{D}_{\text{t}}]$ (the loss of (x, y) on training step t) is determined. The *reducible holdout loss* (RHO-LOSS) value for a point (x, y) is defined as:

$$\mathcal{L}[y \mid x; \mathcal{D}_{\text{t}}] - \mathcal{L}[y \mid x; \mathcal{D}_{\text{ho}}] \tag{1}$$

The RHO-LOSS value gives information about the following types of low-contribution data points:

1) *Redundant points* have already been learned by the model. These points have a low training loss and therefore a low RHO-LOSS value.
2) *Noisy points* may have ambiguous or incorrect labels. These points have a high training loss *and* a high IL. Thus, the RHO-LOSS value for these points is low.
3) *Less relevant points*, such as outliers, have a high training loss. Outliers are less likely to have been represented in $\mathcal{D}_{\text{ho}}$, which gives them a high IL, and therefore a low RHO-LOSS value.

Selecting for backpropagation only the data points with high RHO-LOSS values likely excludes redundant, noisy, and less relevant points, and therefore allows for more efficient training. RHO-LOSS could be especially suitable for use in deep contextual bandit algorithms: as the dataset grows with use, the data on which the network was previously trained is *by definition* a subset of any later version of the dataset. Therefore, the IL for each point can simply be calculated by using the previously trained network – no additional holdout training is needed. The algorithm, implemented as part of Oikonomos-II+'s version of Neural-LinUCB, is visualized in Fig. 5.

E. Variational Autoencoders

The original Neural-LinUCB algorithm uses a deep MLP, yet it is possible to use more advanced architectures as well, which can lead to better performance. An example of such an architecture is a Variational Autoencoder (VAE), first proposed by Kingma et al. [31], [32].

Rather than producing a fixed representation for each input, the encoder is trained to map data to a latent distribution, typically a mean μ and variance σ^2 of a Gaussian distribution. Encoding uncertainty in such a way helps to regularize the latent space and avoid overfitting. This is enforced through a loss function, which typically consists of two components: the reconstruction loss, and the Kullback–Leibler (KL) divergence between the latent distribution and a prior distribution (typically a standard Gaussian). Minimizing the KL divergence encourages the latent distribution to remain smooth and continuous. The VAE can be trained using stochastic backpropagation (the 'reparameterization trick'), where a latent vector z is sampled from the latent distribution.

In Neural-LinUCB, the vector $\boldsymbol{q}_{s,t}$ can be viewed as a representation of the parameters of context vector $\boldsymbol{x}$. This vector captures only the latent factors which contribute to the reward.

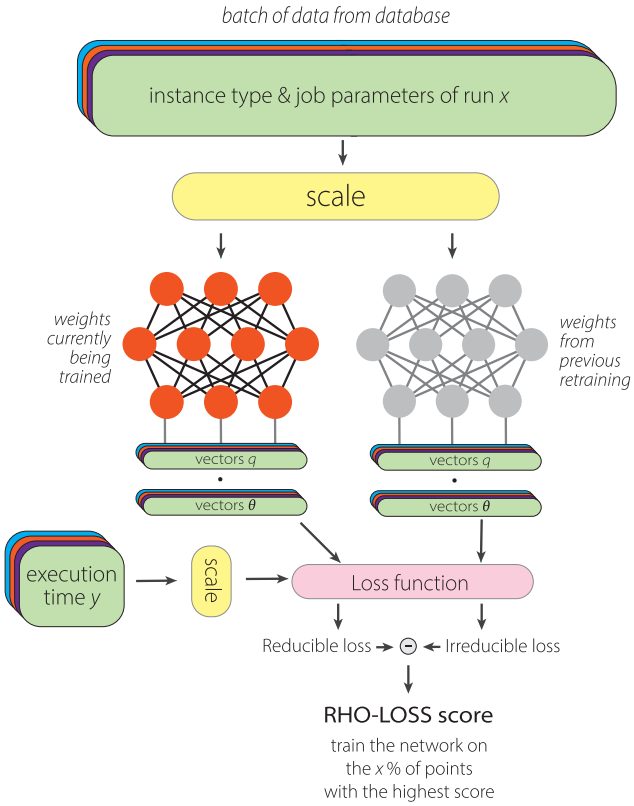

Fig. 5. *RHO-LOSS algorithm*, based on the work by Mindermann et al. [30] and as implemented in Oikonomos-II+. The loss function can be any standard regression loss function. For each data point, its reducible loss is determined by using the trained weights from the previous retraining session. The irreducible loss is determined using the weights of the network that is currently being trained. The RHO-LOSS score is defined as the difference between the reducible loss and the irreducible loss. The data points with the highest RHO-LOSS score are most suitable to be used as training data.

More precisely, $\boldsymbol{q}$ contains features for which the linear combination estimates the expected reward. Using a VAE instead of an ordinary MLP leads to a more structured latent representation, which will improve the performance of the algorithm.

The loss function $\mathcal{L}$ is defined as a weighted sum of the KL-loss (averaged per latent variable) and the MSE loss between the predicted reward x and the actual reward y, and is calculated as follows:

$$\mathcal{L} = \beta_0 \mathcal{L}_{\text{KL}} + \beta_1 \mathcal{L}_{\text{MSE}} \tag{2}$$

with

$$\mathcal{L}_{\text{KL}} = -\frac{1}{2d} \sum_{i=1}^{d} \left(1 + \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2\right) \tag{3}$$

The purpose of $\mathcal{L}_{\text{KL}}$ is to regularize the latent space. This improves the performance of the LinUCB algorithm, which takes the latent representations as its input. During training, the sampled vector $\boldsymbol{z}_{s,t}$ is used as context vector $\boldsymbol{q}_{s,t}$, which is multiplied with vector $\boldsymbol{\theta}_{s,t}^{\intercal}$ to obtain the estimated reward $\mathbb{E}[r_{s,t}]$. $\mathbb{E}[r_{s,t}]$ is compared to the actual reward $r_{s,t}$ using Mean Squared Error (MSE), which is a standard loss function for regression. Since the goal here is regression, not reconstruction, no further reconstruction loss term is required.

F. Gaussian-Process Regression

ANNs, such as the multilayer perceptron, can be seen as universal function approximators and are therefore capable of representing complex, nonlinear relationships between context and reward in contextual MAB problems. However, deep ANNs tend to require a lot of training data to function properly. This poses a problem for contextual MAB problems where data is scarce in the beginning. In such situations, Gaussian Process Regression (GPR) can be a better choice. Gaussian Processes (GPs) can be defined as nonlinear random functions which can be represented in a linear space of infinite dimensions. This means that GPs allow for the representation of nonlinear relationships between context and reward. A GP$(\mu(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$ is specified by a mean function $\mu(\boldsymbol{x})$ and a covariance function $k(\boldsymbol{x}, \boldsymbol{x}')$. When used for regression in a contextual MAB problem, GPR provides both an estimated reward $\mu[r_{s,t}]$ and a standard deviation $\sigma[r_{s,t}]$, like LinUCB and Neural-LinUCB. This means that GPR can be combined with UCB to provide a bandit algorithm, which has been done by Srinivas et al. in their algorithm GP-UCB [29].

Even though GPR is expected to outperform deep-learning-based contextual MAB algorithms early on, when there is a paucity of data, as the amount of data grows, deep-learning-based methods are likely to provide a better representation of the relationship between context and reward. It is possible to combine the two algorithms, for instance using inverse-variance weighting, a standard statistical technique which we can use in the following way to combine the estimated rewards and variances from both algorithms:

$$\mu[r_{s,t}] = \frac{\mu_g/\sigma_g^2 + \mu_b/\sigma_b^2}{1/\sigma_g^2 + 1/\sigma_b^2} \tag{4}$$

$$\sigma[r_{s,t}] = \sqrt{\left(\frac{1}{\sigma_g^2} + \frac{1}{\sigma_b^2}\right)^{-1}} \tag{5}$$

where μ_g and σ_g are the estimated reward and standard deviation produced by GPR, and μ_b and σ_b the estimated reward and standard deviation produced by Neural-LinUCB.

This means that when the GP model is more confident, its estimates are given more weight. When the ANN becomes more confident over time, it will become more dominant. This allows the algorithm to transition from relying on GPR when data is scarce to relying on the ANN as more data is gathered. When combining bandit algorithms in such a way, one needs to pay attention to scaling; if the different bandit algorithms scale the estimated rewards and confidence bounds differently, it is necessary to rescale them before weighting them. The design of this algorithm, as used in Oikonomos-II+, is summarized in Fig. 4.

IV. Implementation

The design of Oikonomos-II+ was visualized in Fig. 1, and details of its subcomponents are visualized in Figs. 3, 4 and 5. The system was implemented in Python. As the machine-learning

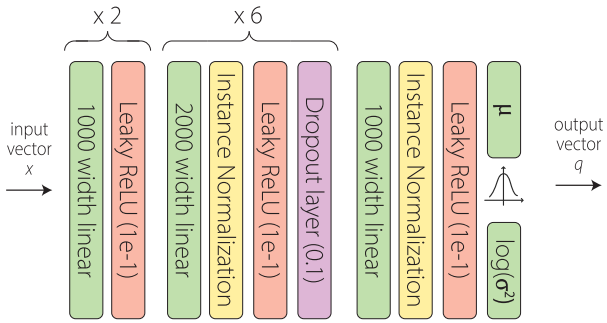

Fig. 6. The architecture of the MLP used in Oikonomos-II+.

framework we used PyTorch [35]. An MLP of nine linear layers was used, with a maximum width of 2,000. Normalization and dropout layers were used to enhance performance. (Leaky) ReLU functions were used as activation layers. The full architecture is summarized in Fig. 6. The length d of the output vector (which corresponds to the length of vector θ) is 700. As loss function $\mathcal{L}$ we used the Mean Squared Error (MSE) loss function. The minibatch size (defined by the number of training points in a minibatch *after* filtering using the RHO-LOSS algorithm) was initialized to 1 and slowly increased as $\mathcal{D}$ increased, to a maximum of 16.

The MLP was retrained after 50 episodes, 200 episodes, and then at intervals of 500 episodes. We used 4-fold cross-validation, which means that at each fold, 75% of the dataset was used as the training set, and the remaining 25% was used as the validation set. Backpropagation is performed using the AdamW optimizer [36], [43] with AMSGrad [44]. The number of training epochs is determined by early stopping: when the lowest validation loss is older than 300 epochs, training is stopped and the weights of the lowest validation loss are retained. As for the reward function: to determine the estimated rewards and confidence bounds, the bandit algorithm uses $r(x) = -x$, with x the execution time in seconds. If the user decides to optimize for cost rather than execution time, Oikonomos-II+ multiplies the reward scores with the costs per second for the specific instance types to make recommendations. The parameter sets and the reward sets were both scaled using the Yeo-Johnson power transformation [45]. This power transformation makes the datasets more Gaussian-like, which aids Oikonomos-II+ in estimating the rewards and determining the confidence bounds.

V. Evaluation

A. Experimental Setup

Oikonomos-II+ is designed to be application-agnostic: for any application, the user needs to provide the system with a list of job parameters that may influence the performance and optimal hardware configuration of individual jobs. Additionally, providing the ranges of possible parameter values can be useful, as this gives an indication as to how to best scale these values. For applications with large numbers of parameters that can influence performance, such as computer-vision applications, it may be advisable to pre-process and/or compress the parameter values before offering them to Oikonomos-II+. However, this requires some technical knowledge from the user. Since Oikonomos-II+ relies on a relatively simple MLP architecture and GP regression kernel, it does not require retraining after every recommendation, and the RHO-LOSS algorithm reduces the amount of training data needed. As a result, it can be deployed on a single node with a single GPU or even a local server. This means that the overhead costs of running the Oikonomos-II+ system are minimal compared to the costs of running HPC and AI workloads.

TABLE II
HPC & AI Applications Used for the Oikonomos-II+ Evaluation, and Their Parameter Domains

(A) WhiskEras 2.0

parameter	domain
Number of frames	$\mathbb{N}_+$
Snout coordinates	$\mathbb{N}_+^4$
Gamma	$\mathbb{R}_+$
Minimum length	$\mathbb{N}_+$
First frame hash	$\mathbb{N}_0^4$
Number of whiskers	$\mathbb{N}_+$

(B) SimHH

parameter	domain
Step size	$\mathbb{R}_+$
Nr. of neurons	$\mathbb{N}_+$
Connectivity	$\mathbb{R}$
Nr. of time steps	$\mathbb{N}_+$
Simulation time	$\mathbb{R}_+$

(C) Oceananigans

parameter	domain
Horizontal grid size	$\mathbb{N}_+$
Vertical grid size	$\mathbb{N}_+$
Domain height	$\mathbb{R}_+$
Wall velocity	$\mathbb{R}_0$
Reynolds number	$\mathbb{N}_+$
Prandtl number	$\mathbb{R}_+$
Richardson number	$\mathbb{R}_+$
Buoyancy frequency	$\mathbb{N}_+$
Simulation end time	$\mathbb{R}_+$
Nr. of repetitions	$\mathbb{N}_+$

(D) MLP (MNIST)

parameter	domain
Training batch size	$\mathbb{N}_+$
Test batch size	$\mathbb{N}_+$
Layer 1 size	$\mathbb{N}_+$
Layer 2 size	$\mathbb{N}_+$
Nr. of epochs	$\mathbb{N}_+$

(E) RNN (sunspots)

parameter	domain
Nr. of time steps	$\mathbb{N}_+$
Hidden layer size	$\mathbb{N}_+$
Training batch size	$\mathbb{N}_+$
Test batch size	$\mathbb{N}_+$
Nr. of epochs	$\mathbb{N}_+$

(F) CNN (CIFAR-10)

parameter	domain
Model nr.	$[1, 3]$
Hidden layer size	$\mathbb{N}_+$
Training batch size	$\mathbb{N}_+$
Test batch size	$\mathbb{N}_+$
Nr. of epochs	$\mathbb{N}_+$

For the evaluation of our system, we used six different benchmark applications, divided into two categories: scientific HPC applications and AI applications. From each category, we use three applications. All applications have been selected for their ability to run on various types of hardware, with or without the presence of a GPU, and also to permit high parameterization, so as to evaluate the recommendation capabilities of Oikonomos-II+. One of the applications is a computer-vision application for which the performance depends on a large number of parameters. For this application, we performed some preprocessing and compression on a subset of parameters. The tunable parameters for each application are summarized in Table II.

The first scientific HPC application, SimHH, is a neurosimulator developed at the Erasmus Medical Center (EMC), Rotterdam [42]. It simulates a wide range of biologically plausible, conductance-based Hodgkin-Huxley neural models. These models work on non-embarrassingly parallel workloads and use

TABLE III
AMAZON EC2 INSTANCE TYPES USED FOR OIKONOMOS-II+ EVALUATION

Instance type	CPU type	vCPU no.	Memory (GiB)	GPU type	GPU mem. (GiB)
c5a.xlarge	AMD EPYC 7R32 @ 3.3 GHz	4	8	–	–
c5a.4xlarge	AMD EPYC 7R32 @ 3.3 GHz	16	32	–	–
c5a.8xlarge	AMD EPYC 7R32 @ 3.3 GHz	32	64	–	–
c6g.xlarge	AWS Graviton2 Processor @ 2.5 GHz	4	8	–	–
c6g.4xlarge	AWS Graviton2 Processor @ 2.5 GHz	16	32	–	–
c6g.8xlarge	AWS Graviton2 Processor @ 2.5 GHz	32	64	–	–
g3.4xlarge	Intel Xeon E5-2686 v4 (Broadwell) @ 2.3 GHz	16	122	NVIDIA Tesla M60	8
g4dn.2xlarge	Intel Xeon Family @ 2.5 GHz	8	32	NVIDIA T4 Tensor Core	16
g5.2xlarge	AMD EPYC 7R32 @ 2.8 GHz	8	32	NVIDIA A10G	24
m5a.xlarge	AMD EPYC 7571 @ 2.5 GHz	4	16	–	–
m5a.4xlarge	AMD EPYC 7571 @ 2.5 GHz	16	64	–	–
m5a.8xlarge	AMD EPYC 7571 @ 2.5 GHz	32	128	–	–
r5.xlarge	Intel Xeon Platinum 8175 @ 3.1 GHz	4	32	–	–
r5.4xlarge	Intel Xeon Platinum 8175 @ 3.1 GHz	16	128	–	–
r5.8xlarge	Intel Xeon Platinum 8175 @ 3.1 GHz	32	256	–	–

basic solvers operating on short time intervals. SimHH is written in C++ and uses OpenMPI, OpenMP, and CUDA to make use of multi-core CPUs and GPUs, making it an excellent benchmark application.

The second scientific HPC application, WhiskEras 2.0 [38], is an application that allows fast and accurate whisker tracking in videos of head-fixed rodents. The application is an improved and accelerated version of WhiskEras, a whisker-tracking algorithm developed at the EMC [39]. WhiskEras 2.0 is written in C++ and uses CUDA and OpenMP to achieve a speedup over the original algorithm, and works with or without a GPU. For this application, rather than feeding all pixels of all the frames of the video fragment to Oikonomos-II+, we compressed the first frame of the video into a hash code consisting of four integers. This was done in such a way that similar frames have similar hash codes, and dissimilar frames have dissimilar hash codes. In this way, Oikonomos-II+ gets the opportunity to compare new videos to videos that were already processed, and use similarity as a way to estimate execution time.

The third scientific HPC application is a benchmark for Oceananigans. Oceananigans is a software package for finite-volume simulations. The package was written in Julia and makes use of CPUs and GPUs. The authors provide several experiments that can be used as benchmarks; the one we use is a simulation of stratified Couette flow, as used by Vreugdenhil and Taylor [34].

The remaining three benchmark applications consist of AI workloads; more specifically, scripts to train deep ANNs of three different types on standard benchmarking datasets. The first script trains an MLP. The MLP was built with Google TensorFlow and consists of two layers. It was trained on the MNIST database [41], which contains a large number of handwritten digits. The task of the MLP is to recognize the digit on each image. The second script trains a Convolutional Neural Network (CNN) on the CIFAR-10 database [40], which consists of thousands of color images from ten classes. The task is to classify the images correctly. The third script trains a Recurrent Neural Network (RNN) to predict the number of sunspots.

The Amazon instance types that we have used for our evaluation are shown in Table III. Without loss of generality, they were selected so as to create a diversity of hardware options, but we also selected some instance types from the same family. This allows us to test if Oikonomos-II+ can discern between instances that are relatively similar. Three instance types have a GPU available, six are compute-optimized, three are memory-optimized, and three are general-purpose instances. Of the six compute-optimized instances, three are equipped with AMD EPYC 7R32 CPUs and three are equipped with Amazon's AWS Graviton2 processors.

To evaluate the performance of Oikonomos-II+, we used datasets where all the jobs have been executed fully on each of the fifteen instance types (with the exception of the 'Oceananigans' and 'WhiskEras' datasets, which were run on fourteen of the instance types). We call these datasets *oracle sets*, since they provide us with full insight into the best possible policy and the regret of each different policy.[3] By using these sets as a simulation environment, it was possible to evaluate the regret for each application. For our six applications, we used oracle sets of about 5,000 jobs.

The sets were created by executing jobs on the Amazon EC2 instances, and then augmenting the data by manually studying the behavior of these applications, in order to create sets that reliably represent the application behavior on the cloud instances. The data points obtained from executing jobs were shuffled randomly, after which each data point was used to generate 11 new data points. Shuffling the measured data points before augmenting the data constrains the amount of data leakage to at most 11 episodes. This is acceptable given that Oikonomos-II+ takes the order of incoming jobs into account when making predictions. The algorithm only gets to see the execution time of a job for the instance type it has chosen, and it cannot see into the future. Each of our six applications was tested both for the cost-optimized and the time-optimized scenario; this gave us a total of twelve scenarios.

Comparing Oikonomos-II+ to other work is challenging, since each author uses their own HPC application to evaluate performance – the absence of a good benchmark suite for cloud HPC recommendation is a persistent issue in this field. Even when the same applications are used, differences in parameter ranges can lead to vastly different datasets. Most standard HPC

[3]The oracle sets that were used for evaluation can be found at: https://dx.doi.org/10.21227/vzpy-ef73.

TABLE IV
DISTRIBUTION OF BEST INSTANCE TYPE IN ORACLE SETS (TIME / COST)

	CNN (CIFAR-10)	MLP (MNIST)	Oceananigans	RNN (sunspots)	WhiskEras 2.0	SimHH
c5a.xlarge	0.00% / 0.22%	0.14% / 84.61%	35.20% / 99.56%	0.04% / 28.05%	0.02% / 3.94%	0.00% / 63.02%
c5a.4xlarge	0.00% / 0.00%	0.18% / 0.00%	9.24% / 0.00%	0.58% / 0.00%	0.02% / 0.00%	11.56% / 0.00%
c5a.8xlarge	0.00% / 0.00%	1.45% / 0.00%	0.44% / 0.00%	0.60% / 0.02%	0.00% / 0.00%	13.26% / 0.00%
c6g.xlarge	0.00% / 0.00%	0.28% / 14.66%	0.00% / 0.22%	0.00% / 1.14%	0.00% / 0.00%	0.36% / 2.80%
c6g.4xlarge	0.00% / 0.00%	0.16% / 0.00%	0.00% / 0.00%	0.00% / 0.00%	0.00% / 0.00%	3.50% / 0.14%
c6g.8xlarge	0.00% / 0.00%	4.40% / 0.00%	0.00% / 0.00%	0.02% / 0.00%	0.00% / 0.00%	2.04% / 0.00%
g3.4xlarge	0.00% / 0.00%	0.18% / 0.00%	– / –	0.00% / 0.00%	– / –	0.00% / 0.00%
g4dn.2xlarge	1.42% / 70.20%	28.11% / 0.28%	3.50% / 0.00%	1.60% / 16.17%	0.00% / 59.50%	41.52% / 33.00%
g5.2xlarge	98.58% / 29.58%	12.02% / 0.00%	49.20% / 0.22%	87.29% / 54.62%	99.94% / 36.56%	27.72% / 1.04%
m5a.xlarge	0.00% / 0.00%	0.00% / 0.44%	0.00% / 0.00%	0.00% / 0.00%	0.00% / 0.00%	0.00% / 0.00%
m5a.4xlarge	0.00% / 0.00%	0.00% / 0.00%	0.00% / 0.00%	0.00% / 0.00%	0.00% / 0.00%	0.00% / 0.00%
m5a.8xlarge	0.00% / 0.00%	0.08% / 0.00%	0.00% / 0.00%	0.00% / 0.00%	0.00% / 0.00%	0.00% / 0.00%
r5.xlarge	0.00% / 0.00%	0.00% / 0.00%	0.44% / 0.00%	0.68% / 0.00%	0.00% / 0.00%	0.00% / 0.00%
r5.4xlarge	0.00% / 0.00%	0.12% / 0.00%	1.98% / 0.00%	2.04% / 0.00%	0.02% / 0.00%	0.04% / 0.00%
r5.8xlarge	0.00% / 0.00%	52.87% / 0.00%	0.00% / 0.00%	7.14% / 0.00%	0.00% / 0.00%	0.00% / 0.00%

benchmark suites are unsuitable for our purpose: they are designed to characterize specific HPC platforms, and fail to capture the complex interplay between application characteristics, individual job input-parameter values, and hardware. We therefore decided to evaluate the performance of Oikonomos-II+ in its own right. Nevertheless, for each of our twelve scenarios, we were able to test six different configurations of Oikonomos-II+:

1) Combined bandit (GP predictor + Neural-LinUCB) with VAE, trained on all data points
2) Combined bandit without VAE, trained on all data points
3) Combined bandit with VAE, trained on 25% of data points with the highest RHO-LOSS score
4) Singular bandit (Neural-LinUCB only) with VAE, trained on all data points
5) Singular bandit without VAE, trained on all data points
6) Singular bandit with VAE, trained on 25% of data points with the highest RHO-LOSS score

We employed the following *three metrics:*

a) The *percentage of all rounds* for which the best instance type was recommended. This shows the performance of Oikonomos-II+ in selecting the best instance type, including the exploration phase.
b) The *percentage of the last 1,000 rounds* for which the best instance type was recommended. By this time, the algorithm has had the opportunity to explore and should be mostly exploiting.
c) The *cumulative regret*. Regret is usually defined as the difference between the optimal policy and the actual policy. The unit and size of the regret differs for every application. In order to compare the applications, it was decided to express cumulative regret as a percentage of the cumulative regret of a random policy. In contrast to metrics A and B, a *lower value* for metric C signifies a *better performance* of Oikonomos-II+ - after all, the goal of a multi-armed-bandit algorithm is to minimize regret.

This set of three metrics, applied to twelve different scenarios and six different configurations of Oikonomos-II+, will give a thorough overview of the performance of Oikonomos-II+. Along with these metrics, we also visualize the performance of Oikonomos-II+ for the SimHH application by plotting the cumulative regret over time, as well as the confusion matrices for the first and last 500 episodes.

B. Results

We analyzed the oracle sets to determine the distribution of the best option. As shown in Table IV, for each of the scenarios, there is some variation regarding the best instance type, albeit sometimes limited (e.g., in the cost-optimized Oceananigans and time-optimized WhiskEras cases, where more than 99% of the jobs have the same optimal instance type). The most interesting cases are those where the optimal choice of instance type is not

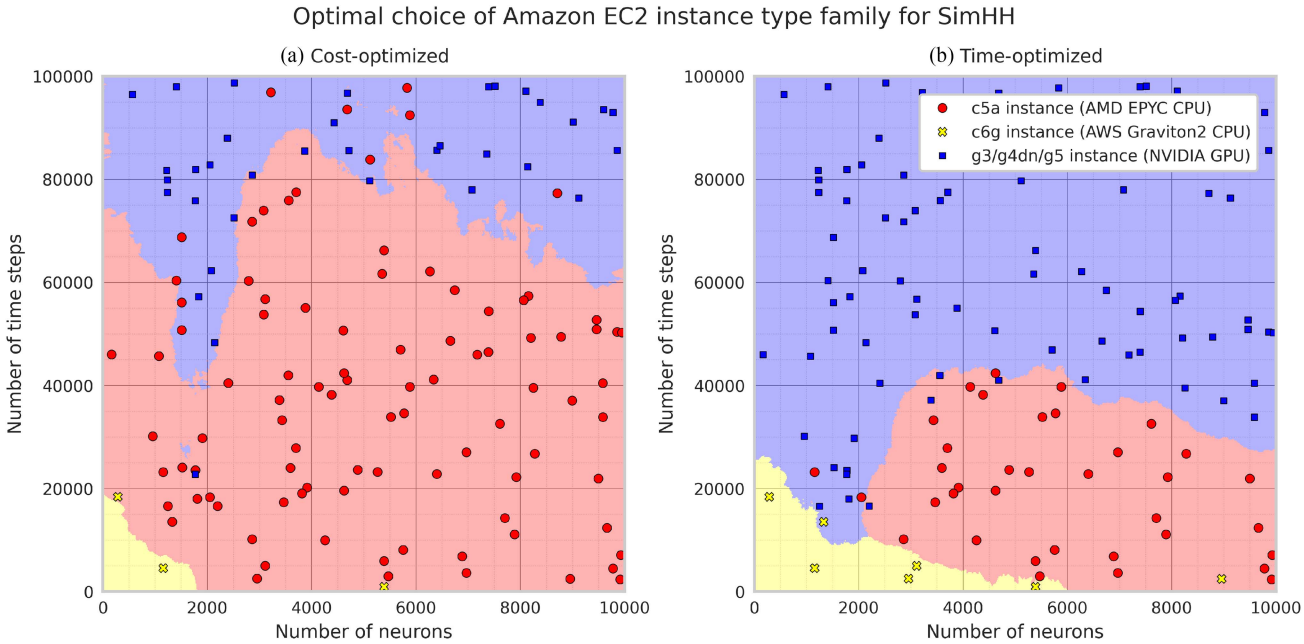

Fig. 7. Optimal choice of Amazon EC2 instance type in one of our datasets, for jobs of the SimHH application with different input parameters, for cost- and time-optimized scenarios. The scatterplot shows a random sample of 5% of our dataset. To determine the shaded regions of optimal choice, we used the kNN algorithm with $k = 50$ on the whole dataset. In both the cost- and time-optimized scenarios, the optimal choice of instance type heavily depends on the input parameters, confirming the findings of Smaragdos et al. [2].

obvious and is highly dependent on the parameter values (e.g., in the time-optimized Oceananigans and cost-optimized CNN cases).

Fig. 7 shows plots of the family of the best instance type for various combinations of two parameters (the number of simulated biological neurons and the number of simulation steps) on a random sample of our oracle set for the SimHH application. The figure clearly shows that the choice of best instance type is heavily dependent on the input-parameter values, confirming the findings of Smaragdos et al. [2], and validating the main premise behind our approach.

Table V shows the evaluation results for all twelve scenarios, for all six configurations, employing the three metrics. Several observations can be made:

1) *Accuracy improves over time:* For almost every scenario, the percentage of correct instance-type recommendations in the last 1000 rounds (metric B) is higher than the percentage of correct recommendations in all the rounds (metric A). This shows that Oikonomos-II+ is an *effective contextual multi-armed bandit algorithm:* in the beginning, it explores in order to collect information about the behavior of the application, and exploits this knowledge in later episodes. For the few cases where metric A is higher than metric B, the difference is very small; in such cases, the exploration phase is likely to have been very short.
2) *GP helps in exploration:* We observe that the difference between metric A and B is much smaller in configurations where a GP predictor is included (1, 2, and 3) than in the configurations without a GP predictor (4, 5, and 6). A large difference between metric A and metric B points to a longer exploration phase; a small difference suggests a short exploration phase. We can conclude that *a GP predictor is especially helpful in the exploration phase*. This is consistent with our expectations: after all, in the exploration phase, there is a paucity of data, and these are the situations where a GP predictor is especially helpful. However, both the combined bandit and the singular bandit learn to make better predictions over time.
3) *VAE regularizes latent space:* In the case of a singular bandit, the configuration with VAE (configuration 4) *always has a lower value* for metric C than the configuration without VAE (configuration 5). This shows that including the VAE has the desired effect of regularizing the latent space, improving the performance of the LinUCB algorithm. For the case of the combined bandit with and without VAE (configurations 1 and 2), this is the case for 11 of the 12 scenarios - only for time-optimized Oceananigans, the configurations without VAE perform marginally better.
4) *RHO-LOSS effective in combined bandit:* It can be seen that the desired goal of RHO-LOSS, a dramatic reduction in required training data with limited performance loss, is consistently achieved in configurations with a combined bandit. In two cases (time-optimized Oceananigans and cost-optimized CNN), there is even a small *performance improvement*. In singular-bandit configurations, the results are mixed - the performance loss ranges from small (e.g., in the case of cost-optimized MLP) to devastating (cost-optimized Oceananigans). A possible explanation for this difference is that reducing the amount of training data is particularly risky in the exploration phase, when there is already a paucity of data. The GP predictor possibly makes up for this loss.

TABLE V
OIKONOMOS-II+ EVALUATION RESULTS (METRICS A/B/C)

	Combined bandit 100% of data with VAE (1)	Combined bandit 100% of data without VAE (2)	Combined bandit 25% of data with VAE (3)	Singular bandit 100% of data with VAE (4)	Singular bandit 100% of data without VAE (5)	Singular bandit 25% of data with VAE (6)
Oceananigans (cost-optimized)	88.66% / 95.00% / 1.40%	80.44% / 89.30% / 7.34%	81.54% / 89.70% / 2.04%	70.22% / 94.30% / 3.55%	35.62% / 74.40% / 17.51%	4.14% / 6.10% / 28.23%
Oceananigans (time-optimized)	59.16% / 58.20% / 2.48%	56.16% / 55.30% / 1.93%	55.42% / 55.30% / 2.27%	25.06% / 38.20% / 5.46%	24.38% / 30.30% / 9.25%	16.40% / 16.40% / 12.58%
SimHH (cost-optimized)	78.34% / 77.70% / 2.20%	76.66% / 78.80% / 2.63%	75.64% / 75.50% / 2.73%	66.24% / 82.00% / 5.88%	59.80% / 80.60% / 10.55%	54.40% / 72.70% / 9.78%
SimHH (time-optimized)	55.00% / 66.80% / 2.28%	47.84% / 60.40% / 2.85%	46.68% / 54.10% / 2.71%	56.48% / 67.10% / 3.12%	57.02% / 68.40% / 5.00%	55.06% / 66.60% / 4.14%
WhiskEras (cost-optimized)	67.84% / 74.50% / 0.68%	65.06% / 71.30% / 1.53%	69.40% / 78.50% / 0.88%	53.90% / 61.50% / 1.62%	42.72% / 47.50% / 9.34%	43.86% / 48.70% / 5.86%
WhiskEras (time-optimized)	97.18% / 97.90% / 0.26%	96.88% / 98.40% / 0.86%	96.88% / 97.30% / 0.51%	74.28% / 80.60% / 0.96%	74.60% / 94.40% / 7.47%	62.62% / 81.10% / 4.31%
CNN (cost-optimized)	67.94% / 71.60% / 1.14%	69.52% / 77.30% / 1.23%	68.48% / 72.90% / 1.12%	50.58% / 66.40% / 4.29%	51.70% / 62.30% / 5.52%	42.10% / 54.90% / 5.02%
CNN (time-optimized)	92.28% / 89.60% / 0.48%	89.96% / 92.50% / 0.49%	92.50% / 98.60% / 0.50%	71.12% / 81.80% / 2.28%	56.20% / 87.10% / 4.36%	68.10% / 78.20% / 3.37%
RNN (cost-optimized)	73.51% / 78.60% / 1.41%	72.39% / 77.00% / 1.88%	72.03% / 76.70% / 1.59%	58.46% / 67.00% / 3.34%	60.52% / 77.70% / 7.77%	49.96% / 54.90% / 5.57%
RNN (time-optimized)	84.45% / 87.10% / 0.61%	85.83% / 93.40% / 0.73%	83.55% / 85.70% / 0.84%	75.99% / 83.70% / 0.88%	69.55% / 85.50% / 5.94%	75.23% / 85.90% / 2.18%
MLP (cost-optimized)	79.31% / 81.70% / 0.74%	78.14% / 81.60% / 1.05%	78.97% / 83.10% / 0.76%	71.35% / 79.30% / 6.86%	60.86% / 66.30% / 12.21%	69.15% / 77.20% / 7.58%
MLP (time-optimized)	59.37% / 60.90% / 7.77%	62.07% / 67.70% / 8.16%	60.82% / 66.70% / 8.52%	56.30% / 59.30% / 8.61%	61.54% / 67.50% / 11.11%	50.37% / 57.30% / 12.35%

5) *Outliers disrupt one scenario:* For one of the scenarios, time-optimized MLP, Oikonomos-II+ scores significantly worse for metric C across all configurations. Upon inspection of the dataset for this scenario, it was observed that the dataset contains several executed jobs that are far outliers in terms of execution time, that are not easily explained by the job parameters. Possibly, these jobs represent an irregularity in AWS. Removing the outliers from the dataset led to an improvement in performance, which suggests that Oikonomos-II+'s performance was indeed thrown off by a small number of outliers. A possible explanation for this is that the scaler, which is fitted on the full set of training data, is strongly influenced by these outliers; however, this requires further research. For all other benchmarks, Oikonomos-II+ performs well.

In addition to our three metrics, we can also visualize the performance of Oikonomos-II+ by plotting cumulative regret over the full set of episodes, i.e., over time. Furthermore, the recommendations that Oikonomos-II+ makes can be compared to the optimal recommendations by means of confusion matrices. For conciseness, we only show this for the cost-optimized SimHH scenario. Fig. 8(a) shows the cumulative regret over time for all six configurations. The figure confirms the above observations: both the singular bandit and the combined bandit converge over time, albeit without reaching 100% accuracy (observation 1), but in the combined-bandit configurations, the exploration phase is much less expensive (observation 2). Configurations with VAE outperform configurations without VAE (observation 3). The one configuration that shows much less convergence than the other ones is the singular bandit with a reduced training dataset. This shows that RHO-LOSS can be used with limited performance loss in combined-bandit configurations but has a more pronounced negative impact on performance in singular-bandit configurations (observation 4).

Fig. 8(b) and (c) show confusion matrices for the best-performing configuration in the cost-optimized SimHH case (the combined bandit with VAE and without RHO-LOSS). Fig. 8(b) shows the confusion matrix for the first 500 episodes, during which Oikonomos-II+ is still in the exploration phase. It can be observed that Oikonomos-II+ explores by recommending a variety of instance types, especially for jobs where the optimal instance type is `c5a.xlarge`. For jobs where the optimal instance type is `g4dn.2xlarge`, Oikonomos-II+ already shows some convergence towards `g4dn.2xlarge`, but also picks one

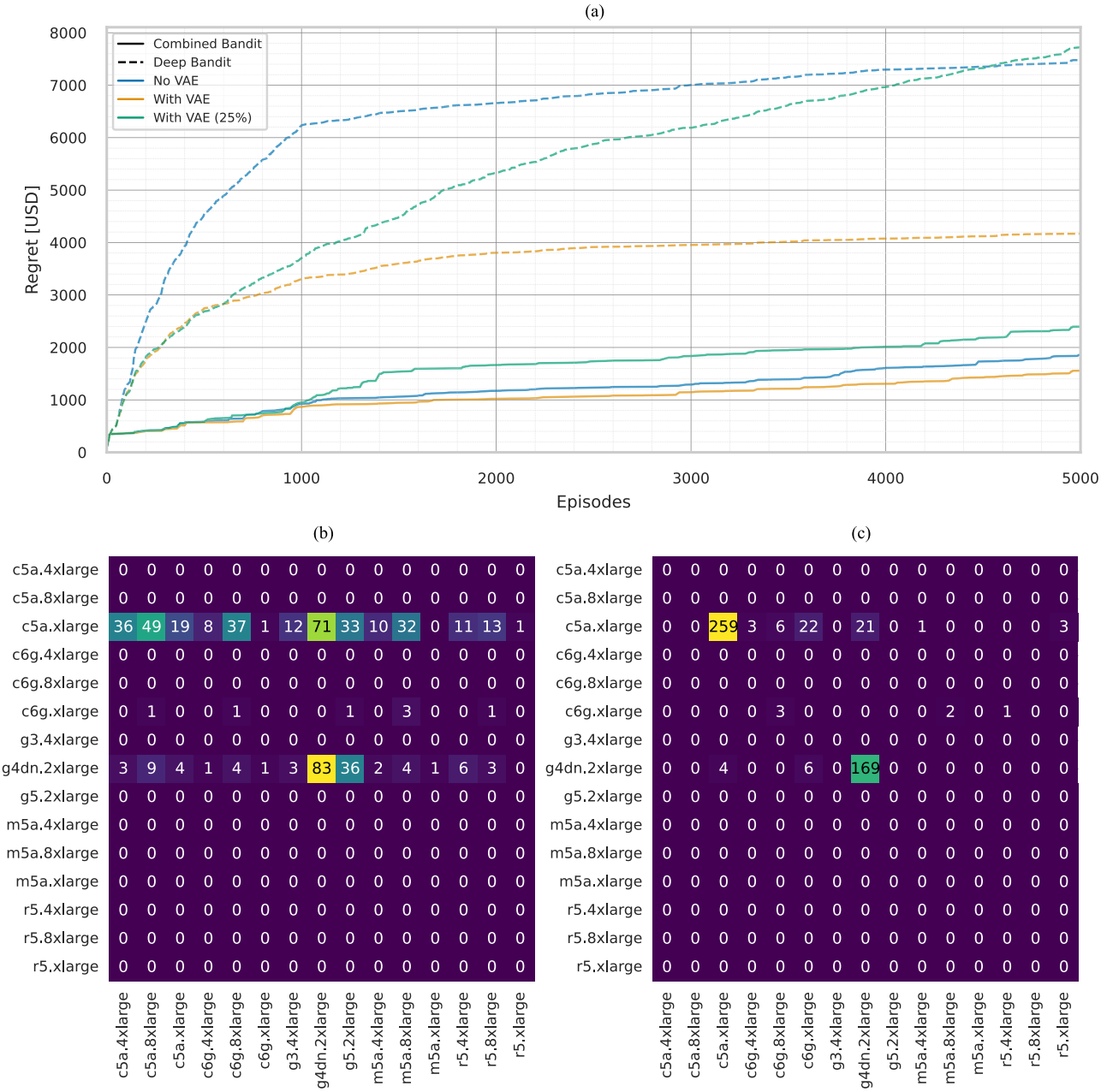

Fig. 8. (a) Cumulative regret for SimHH (cost-optimized), for all six configurations. Regret increases rapidly in the beginning, but mostly flatlines. The configurations that combine Neural-LinUCB with a GP predictor (the combined-bandit configurations) outperform the configurations that only use Neural-LinUCB (the singular-bandit configurations). The configurations with VAE outperform the configurations without VAE. Strongly reducing the amount of training data, to 25% of the original amount, impacts performance negatively, but more significantly in singular-bandit configurations. (b) Confusion matrix of recommendation choices for the first 500 episodes for SimHH (cost-optimized, optimal configuration). True labels are on the x-axis, whereas recommendations are on the y-axis. Since Oikonomos-II+ has not explored the space yet, it is forced to explore and make suboptimal choices. (C) Confusion matrix for the last 500 episodes for the same application. Now that Oikonomos-II+ has explored the relationship between parameters and performance, it mostly exploits and makes optimal choices: most recommendations coincide with the true best option.

of the other instances with a GPU, `g5.2xlarge`, relatively often. It appears that, *even in the early phase of exploration*, Oikonomos-II+ already makes a rough distinction between jobs that are best executed on a GPU and jobs that should best be run on a CPU. Fig. 8(c) shows the confusion matrix for the last 500 episodes, when Oikonomos-II+ has mostly ceased exploring and switched to exploitation. Here, we clearly see that Oikonomos-II+ converged to the two instance types that are optimal for the vast majority of jobs (`g4dn.2xlarge` and `c5a.xlarge`), and is able to recommend the right instance type for individual jobs, based on the job parameters. In other words, Oikonomos-II+ has achieved its goal of modeling the relationship between different input parameters, instance types, and execution times using user-submitted jobs, and is now approaching an optimal policy.

VI. Discussion

Oikonomos-II+ shows the robustness of a reinforcement-learning approach for resource recommendation for HPC applications in a cloud environment. However, several assumptions were made when validating the algorithm.

One assumption is that cloud instances are up and running, and readily available; *startup times* were not taken into account. In a production environment, it might be useful to keep instances running in some situations (for example, when there is a continuous stream of jobs), whereas in other situations, it would be better to shut them down between runs. Developing an algorithm that takes this into account would be useful, but requires additional information about usage patterns. This could be an interesting extension of the current work for future projects, when combined with a suitable scheduling algorithm.

Another assumption is that jobs arrive and are dispatched in a sequential manner: a new job arrives when the previous job has completed. In reality, however, jobs may arrive simultaneously, and a new job may arrive before the previous ones have finished. This might affect recommender performance. However, the problem of delayed feedback in bandits is well-studied [37], and the structure of Oikonomos-II+ is suitable for expansion to incorporate solutions to challenges that may arise in practice.

Furthermore, in our aim to build a general-purpose, resource-recommendation system for cloud HPC, we defined application behavior as a black box. Oikonomos-II+ only gets to see the values of the input variables and the hardware parameters of the instance types, and its only feedback is the reward it obtains after a job is run on a particular instance type. In reality, there is often more information or knowledge available regarding the characteristics and behavior of the application. Some applications give output while processing a job that can be helpful for recommender systems to learn the application's behavior. In future work, it would be valuable to assess how application-specific information can be used to improve the performance of Oikonomos-II+, without compromising its nature as a general-purpose system.

Oikonomos-II+ requires no pretraining or auxiliary runs, but there are costs: It needs to learn from experience and will make suboptimal choices in the beginning. These costs are well-quantified: this is the cumulative regret as shown in Fig. 8(a). A recommender with a low cumulative regret over the full lifetime of an HPC application is superior to a recommender with a higher cumulative regret. Oikonomos-II+ only uses jobs submitted by users to learn the relationship between parameters and hardware. This makes the system easy and practical for users without any technical knowledge. Nevertheless, in some situations, running auxiliary or representative workloads with no useful output for the user may be beneficial. For such workloads, the *immediate regret* will always be higher than zero and equal to the costs of running the workload, but the knowledge gained from this workload might very well reduce the *cumulative regret*. However, choosing representative workloads either requires technical knowledge from the users or an additional algorithm that can choose or create useful representative workloads. Such an algorithm could be integrated into a future version of Oikonomos-II+, and its advantages are quantifiable by comparing cumulative regret. This, however, lies beyond the scope of this paper.

The current *number of instance types* offered by AWS is over 800. Oikonomos-II+ was tested on data from 15 instance types, which is only a fraction of the number of instances offered. The oracle datasets were constructed based on actual measurements done by running the benchmark applications on actual AWS instances. Nevertheless, we still had to go through several postprocessing steps to attain the sets we used to evaluate Oikonomos-II+. There is currently no standard benchmark suite for resource recommendation in cloud HPC, and we did not have access to actual user data for these applications in a cloud environment. Therefore, it was necessary to collect our own oracle datasets to evaluate performance. This required us to limit ourselves to a small selection of instance types, and synthesize data points based on the measurements. However, synthesis was done in such a way that data leakage is minimized, and the small set of 15 instance types contains sufficient diversity. The fact that oftentimes, there is not one overall 'best' instance type, attests to this. As Oikonomos-II+ takes the hardware characteristics of each instance type into account rather than viewing them as black boxes, scaling the environment to the full number of offered instance types will not lead to significant additional overhead costs of deploying Oikonomos-II+.

In practice, the execution of HPC and AI workloads is often distributed over multiple nodes. Aggregating the computing resources of multiple nodes may lead to better performance, and picking the optimal number of nodes for a particular workload is another challenge. The benchmark applications we used to validate Oikonomos-II+ only contain jobs executed on single nodes. However, the Oikonomos-II+ framework can easily handle multi-node scenarios by simply representing the number of nodes as another context parameter. This is an advantage of the multi-armed bandit approach: it is designed to reward based on context parameters, regardless of the nature of these parameters. The same goes for the fine-tuning of the applications, such as compilation flags. In addition, application-specific optimizations could be implemented on the application level, without the need for adaptations to Oikonomos-II+.

Finally, the metrics and applications that were used to evaluate the performance of Oikonomos-II+ have their limitations. Metrics A and B show the percentages of correct recommendations given by Oikonomos-II+ for various configurations. However, these metrics only distinguish between correct and incorrect recommendations, and do not take into account that incorrect choices can differ when it comes to costliness. Cumulative regret does take these differences into account, and expressing the cumulative regret of Oikonomos-II+'s policy by the regret of a random policy as was done for metric C, allows for comparison across benchmark cases. Nevertheless, even metric C is partially dependent on characteristics of the dataset, as the regret of a random policy might vary depending on the difference of the application's performance on different instance types. That said, the chosen combination of metrics gives a clear view of Oikonomos-II+'s performance, and shows that Oikonomos-II+ behaves consistently well on a wide variety of applications and scenarios.

VII. Conclusion

Oikonomos-II+ casts the problem of cloud instance-type selection for different HPC and AI jobs as a contextual multi-armed bandit problem, with the application behavior considered a black box and the input parameter values and instance type characteristics as context so as to deliver general usability. It applies a combination of two algorithms: an improved version of the Neural-LinUCB algorithm and a Gaussian-Process predictor, leveraging the strengths of both algorithms using inverse-variance weighting. The system starts off without knowledge of the application behavior, and is forced to explore when recommending instances for incoming jobs. However, as it gathers knowledge, Oikonomos-II+ starts exploiting and converges towards optimal choices. We evaluated Oikonomos-II+ on twelve diverse benchmark scenarios using three HPC applications, three AI applications, and two reward functions for six different configurations. Oikonomos-II+ was shown to behave consistently across applications and to converge towards optimal choices, demonstrating its effectiveness and robustness. Oikonomos-II+ avoids the main issues of both prediction-based and search-based recommenders. Combining the best elements of these two approaches into a reinforcement-learning recommender system, Oikonomos-II+ is both generalizable and accessible, making it a promising tool for harnessing the power of the cloud for deploying HPC and AI applications.

Acknowledgment

The authors wish to thank Lennart Landsmeer for his valuable input and insights on making the datasets more robust and optimizing the training of Oikonomos-II+.

References

[1] AWS, "Amazon EC2 instance recommendations," *Amazon Web Services Documentation*. Accessed Jun. 12, 2023. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-recommendations.html

[2] G. Smaragdos et al., "BrainFrame: A node-level heterogeneous accelerator platform for neuron simulations," *J. Neural Eng.*, vol. 14, no. 6, 2017, Art. no. 066008.

[3] J. L. F. Betting, C. I. De Zeeuw, and C. Strydis, "Oikonomos-II: A reinforcement-learning, resource-recommendation system for cloud HPC," in *Proc. IEEE 30th Int. Conf. High Perform. Comput., Data, Analytics*, 2023, pp. 266–276.

[4] J. L. F. Betting, D. Liakopoulos, M. Engelen, and C. Strydis, "Oikonomos: An opportunistic, deep-learning resource recommendation system for cloud HPC," in *Proc. IEEE 34th Int. Conf. Appl.-Specific Syst., Architectures Processors*, 2023, pp. 188–196.

[5] P. Xu, Z. Wen, H. Zhao, and Q. Gu, "Neural contextual bandits with deep representation and shallow exploration," in *Proc. Int. Conf. Learn. Representations*, 2022.

[6] L. T. Yang, X. Ma, and F. Mueller, "Cross-platform performance prediction of parallel applications using partial execution," in *Proc. ACM/IEEE Conf. Supercomputing*, 2005, pp. 40–40.

[7] S. Venkataraman et al., "Ernest: Efficient performance prediction for large-scale advanced analytics," in *Proc. 13th USENIX Symp. Netw. Syst. Des. Implementation*, 2016, pp. 363–378.

[8] F. Samreen, Y. Elkhatib, M. Rowe, and G. S. Blair, "Daleel: Simplifying cloud instance selection using machine learning," in *Proc. IEEE/IFIP Netw. Operations Manage. Symp.*, 2016, pp. 557–563.

[9] N. J. Yadwadkar et al., "Selecting the best VM across multiple public clouds: A data-driven performance modeling approach," in *Proc. Symp. Cloud Comput.*, 2017, pp. 452–465.

[10] O. Alipourfard et al., "CherryPick: Adaptively unearthing the best cloud configurations for Big Data analytics," in *Proc. 14th USENIX Symp. Netw. Syst. Des. Implementation*, 2017, pp. 469–482.

[11] C.-J. Hsu et al., "Scout: An experienced guide to find the best cloud configuration," 2018, *arXiv:1803.01296*.

[12] C.-J. Hsu, V. Nair, V. W. Freeh, and T. Menzies, "Arrow: Low-level augmented Bayesian optimization for finding the best cloud VM," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 660–670.

[13] C.-J. Hsu, V. Nair, T. Menzies, and V. Freeh, "Micky: A cheaper alternative for selecting cloud instances," in *Proc. IEEE 11th Int. Conf. Cloud Comput.*, 2018, pp. 409–416.

[14] G. Mariani et al., "Predicting cloud performance for HPC applications before deployment," *Future Gener. Comput. Syst.*, vol. 87, pp. 618–628, 2018.

[15] J. R. Brunetta and E. Borin, "Selecting efficient cloud resources for HPC workloads," in *Proc. 12th IEEE/ACM Int. Conf. Utility Cloud Comput.*, 2019, pp. 155–164.

[16] F. Samreen, G. Blair, and Y. Elkhatib, "Transferable knowledge for low-cost decision making in cloud environments," in *IEEE Trans. Cloud Comput.*, vol. 10, no. 3, pp. 2190–2203, Third Quarter 2022.

[17] D. Samuel et al., "A2Cloud-RF: A random forest based statistical framework to guide resource selection for high-performance scientific computing on the cloud," *Concurrency Comput.: Pract. Experience*, vol. 32, no. 24, 2020, Art. no. e5942.

[18] X. Ai, T. Jena, S. Khan, R. Hughes, and V. K. Pallipuram, "A2Cloud-H: A multi-tiered machine learning framework for cost-effective cloud resource selection," in *Proc. Future Technol. Conf.*, Springer, 2022, pp. 272–291.

[19] K. Lamar et al., "Evaluating HPC job run time predictions using application input parameters," in *Proc. 17th ACM Int. Conf. Distrib. Event-Based Syst.*, 2023, pp. 127–138.

[20] J. Rocca, "The exploration-exploitation trade-off: Intuitions and strategies," *Towards Data Sci.*, 2021. Accessed Jun. 20 2023. [Online]. Available: https://medium.com/data-science/the-exploration-exploitation-dilemma-f5622fbe1e82

[21] T. Lattimore and C. Szepesvári, *Bandit Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2020.

[22] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933.

[23] A. Slivkins et al., "Introduction to multi-armed bandits," *Found. Trends Mach. Learn.*, vol. 12, no. 1-2, pp. 1–286, 2019.

[24] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, pp. 235–256, 2002.

[25] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proc. 19th Int. Conf. World Wide Web*, 2010, pp. 661–670.

[26] S. Agrawal and N. Goyal, "Thompson sampling for contextual bandits with linear payoffs," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 127–135.

[27] D. Zhou, L. Li, and Q. Gu, "Neural contextual bandits with UCB-based exploration," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 11492–11502.

[28] W. Zhang, D. Zhou, L. Li, and Q. Gu, "Neural Thompson sampling," 2020, *arXiv:2010.00827*.

[29] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," 2009, *arXiv:0912.3995*.

[30] S. Mindermann et al., "Prioritized training on points that are learnable, worth learning, and not yet learnt," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 15630–15649.

[31] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, *arXiv:1312.6114*.

[32] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *Found. Trends Mach. Learn.*, vol. 12, no. 4, pp. 307–392, 2019.

[33] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.

[34] C. A. Vreugdenhil and J. R. Taylor, "Large-eddy simulations of stratified plane couette flow using the anisotropic minimum-dissipation model," *Phys. Fluids*, vol. 30, no. 8, 2018, Art. no. 085104.

[35] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[37] A. Grover et al., "Best arm identification in multi-armed bandits with delayed feedback," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2018, pp. 833–842.
[38] P. Arvanitis, J.-H. L. F. Betting, L. W. J. Bosman, Z. Al-Ars, and C. Strydis, "WhiskEras 2.0: Fast and accurate whisker tracking in rodents," in *Proc. Int. Conf. Embedded Comput. Syst.*, 2021, pp. 210–225.
[39] J.-H. L. F. Betting, V. Romano, Z. Al-Ars, L. W. J. Bosman, C. Strydis, and C. I. De Zeeuw, "WhiskEras: A new algorithm for accurate whisker tracking," *Front. Cellular Neurosci.*, vol. 14, 2020, Art. no. 588445.
[40] A. Krizhevsky et al., "Learning multiple layers of features from tiny images," Toronto, ON, Canada, Tech. Rep., 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html
[41] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[42] M. C. W. Engelen, R. Betting, and C. Strydis, "SimHH: A versatile, Multi-GPU simulator for extended Hodgkin-Huxley networks," *IEEE Access*, vol. 13, pp. 46865–46880, 2025.
[43] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2017, *arXiv:1711.05101*.
[44] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," 2019, *arXiv:1904.09237*.
[45] I.-K. Yeo and R. A. Johnson, "A new family of power transformations to improve normality or symmetry," *Biometrika*, vol. 87, no. 4, pp. 954–959, 2000.

River Betting received the BSc degree in mechanical engineering and the MSc degree in computer engineering from the Delft University of Technology, in 2015 and 2018, respectively, and the PhD degree from the Neuroscience Department of the Erasmus Medical Center in Rotterdam, in 2025. She also received the BA degree in history and the MA degree in russian and eurasian studies from Leiden University. She currently works as a researcher with the Neuroscience Department, Erasmus MC. Her current research interests include artificial intelligence, machine learning, computer vision, and algorithm development.

Qilin Chen received the BSc degree in computer science and engineering from the Delft University of Technology, in 2022 and the MSc degree in biomedical computing from the Technical University of Munich, in 2025. She is currently working toward the PhD degree in computational neuroscience with the Max Planck Institute for Biological Intelligence.

Chris De Zeeuw received the medical and PhD cum laude degrees from Erasmus University, Rotterdam. He is a founder and chair of the Department of Neuroscience, Erasmus MC, Rotterdam, Netherlands, and he is a principal investigator with the Netherlands Institute for Neuroscience of the Royal Netherlands Academy of Arts and Sciences in Amsterdam. He is a member of the Royal Netherlands Academy of Arts and Sciences and he received many prestigious awards including the Huygens Award, Pioneer Award and Beatrix Award. He has acted as program-committee member in many international conferences, in particular in relation to the organisation and function of the cerebellum. He published 3 books and well more than 400 manuscripts on cerebellar function in well-known international journals, and he delivered more than 100 keynote lectures. Moreover, he has been awarded many international research projects, including ERC-advanced and EU Program grants. He has taught neuroscience courses for thousands of master students and he supervised more than 150 PhD students. His current research interests span the fields of cerebro-cerebellar interactions and brain-machine interfacing. Many of his fundamental findings found their way into society and he founded two companies, Neurasmus and BlinkLab.

Christos Strydis (Senior Member, IEEE) received the bachelor's (magna cum laude) degree in electronics & computer engineering from the Technical University of Crete, Greece, in 2003, the MSc (magna cum laude) degree in computer engineering from the Delft University of Technology, The Netherlands, in 2005, with a minor in biomedical engineering, and the PhD degree in computer engineering from the Delft University of Technology and funding from the ICT Delft Research Centre (DRC-ICT) and Google Inc. Dr. Strydis holds a joint associate-professorship with the Neuroscience Department of the Erasmus Medical Center, Rotterdam (NL), and with the Quantum & Computer Engineering Department of the Delft University of Technology. He is the founder and head of the Neurocomputing Laboratory at Erasmus Medical Center. He has acted as program-committee member in various international conferences and has also peer-reviewed for as well as published manuscripts in well-known international conferences and journals, and delivered invited talks in various venues. He has been awarded many national- and EU-level research projects. He has supervised multiple BSc, MSc, and PhD students, and teaches various bachelor- and master-level courses. His current research interests span the fields of biologically plausible brain simulations, next-generation neural implants and ultrasound-based brain imaging.