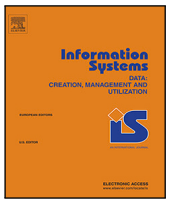
Information Systems 132 (2025) 102538



Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is



RATS: A resource allocator for optimizing the execution of tumor simulations over HPC infrastructures

Errikos Streviniotis [a,b,*], Nikos Giatrakos [a], Yannis Kotidis [c], Thaleia Ntiniakou [d], Miguel Ponce de Leon [d]

[a] School of Electrical and Computer Engineering, Technical University of Crete, University Campus., 73100 Chania, Greece
[b] ATHENA Research and Innovation Centre, Artemidos 6 & Epidavrou, 15125 Athens, Greece
[c] Department of Informatics, Athens University of Economics and Business, 76 Patission St., 10434 Athens, Greece
[d] Life Sciences Department, Barcelona Supercomputing Center, Plaça Eusebi Güell, 1-3, 08034 Barcelona, Spain

ARTICLE INFO

Keywords:
High performance computing
Optimization
Tumor simulations

ABSTRACT

In this work, we introduce RATS (Resource Allocator for Tumor Simulations), the first optimizer for the execution of tumor simulations over HPC infrastructures. Given a set of drug therapies under in-silico study, the optimization framework of RATS can: *(i)* devise the optimal number of cores and prescribe the required number of core hours; and *(ii)* under core capacity constraints, RATS schedules the execution of simulations minimizing the overall number of core hours, simultaneously prioritizing the execution of expectedly promising in-silico trials higher compared to unpromising ones. RATS is deployed by life scientists at the Barcelona Supercomputing Center to remove the burden of blindly guessing the core hours needing to be reserved from HPC admins to study various tumor treatment methodologies, as well as to rapidly distinguish effective drug combinations, thus, potentially cutting time to market for new cancer therapies. The latter is further elevated by the RATS+ extension we plug into the initial framework. RATS+ employs a Transfer Learning approach to leverage optimization models and decisions from prior in-silico studies, thereby reducing the optimization effort required for new studies in this domain.

Our experimental evaluation, on real-world data derived from the execution of more than 2500 tumor simulations on the MareNostrum4 supercomputer, confirms the effectiveness of both RATS and RATS+ across the aforementioned performance dimensions.

1. Introduction & motivation

In precision medicine scenarios, the use of virtual patients in in-silico medical trials holds the promise of revolutionizing the process of novel treatment development by providing an easy, cost-effective, and patient-friendly way to perform preliminary assessment of drug safety and efficacy. In this context, simulation frameworks incorporating computational models of incurable diseases are used to explore the effect of various treatment methodologies towards remedying or improving the quality of patient life under a wide variety of circumstances. When a particular treatment shows promising results in-silico, it is selected to be further investigated, initially in-vitro and, then, potentially in in-vivo trials. This way, the time-to-market for effective treatments is considerably cut down.

In the battle against cancer, at the Barcelona Supercomputing Center, we develop a "virtual laboratory" for studying tumor growth and evolution using a prominent, open-source simulation framework, namely PhysiBoSS 2.0 [1]. For reasons we explain below, this endeavor entails challenges both for life- and computer scientists.

PhysiBoSS incorporates in-silico models of cell systems found in in-vivo tumors and produces output streams describing the state of each individual cell agent and the tumor as a whole, as the simulation process is taking place. Tumor growth and evolution throughout an individual simulation are affected by the drugs of a particular treatment that are applied on the simulated tumor. Effective drug combinations lead tumor cells to necrosis or apoptosis, while ineffective ones result in cell proliferation. Fig. 1 shows examples of a pair of running simulations and their visualized output. The simulation shown at the upper part of Fig. 1 corresponds to an effective treatment methodology because the number of alive (green) cells is reduced, while the number of cells led to necrosis (red) or apoptosis (brown) becomes a majority

* Corresponding author.
E-mail addresses: estreviniotis@tuc.gr (E. Streviniotis), ngiatrakos@tuc.gr (N. Giatrakos), kotidis@aueb.gr (Y. Kotidis), thaleia.ntiniakou@bsc.es (T. Ntiniakou), miguel.ponce@bsc.es (M.P. de Leon).

https://doi.org/10.1016/j.is.2025.102538
Received 17 October 2023; Received in revised form 24 October 2024; Accepted 19 February 2025
Available online 28 February 2025
0306-4379/© 2025 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

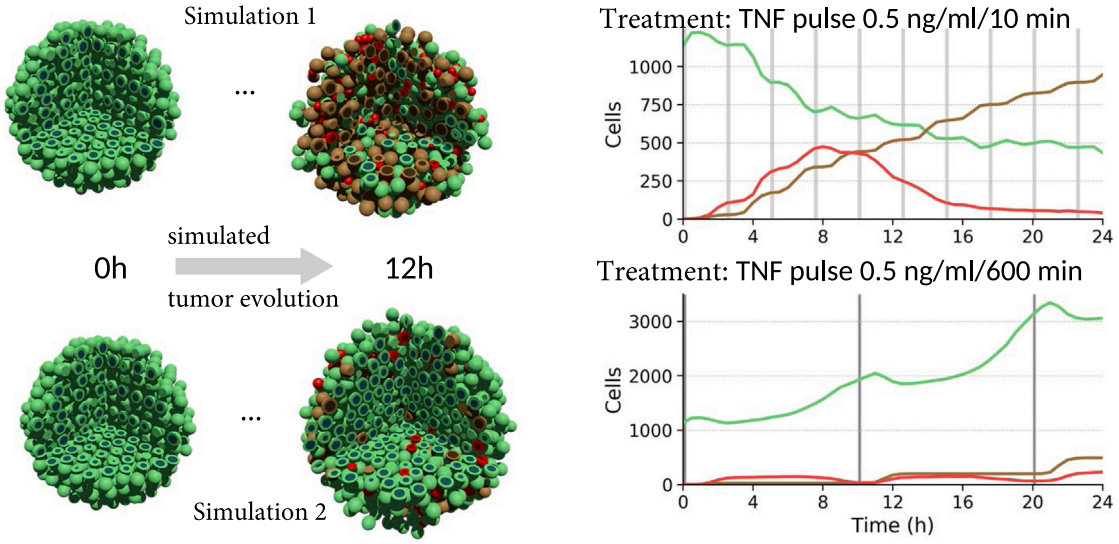


Fig. 1. Two examples of simulated tumors in PhysiBoSS. Tumors are visualized in various time instances (left). Green-colored cells are living tumor cells. Red and brown cells undergo apoptosis and necrosis, respectively. The number of cells per category (alive, necrosis, apoptosis) is also plotted (right). The horizontal axis of each time series is simulation wall time and the vertical axis the number of cells. Each time series' title mentions parameters of the applied drug treatment. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

as time passes. On the other hand, the simulation at the lower part of Fig. 1 illustrates an ineffective treatment methodology since most of the simulated tumor cells remain alive or are proliferating.

The modeled phenomena throughout the simulation are subject to the inherent complexity of biological systems and affected by the interplay between different processes that occur at different scales. For instance, they depend on the molecular mechanisms by which individual cells can develop resistance to a particular drug [2], which are complex in their own right. Moreover, they depend on various types of dynamic processes concerning populations of cells. Examples of the latter include the variability in the gene expression profiles of different cells, which gives rise to heterogeneous populations, the competition for resources such as space and nutrients, as well as the interaction or cross-talk between different cells [3]. Consequently, tumor simulations are extremely computationally demanding. For instance, studying the effect of drug combinations on simulated tumors of realistic sizes can generate cell state data of 100 GB/min [4]. These simulations cannot run on commodity hardware and traditional computer clusters, but require High Performance Computing (HPC) resources.

A fundamental treatment methodology that is used in such tumor simulation studies the effect of the Tumor Necrosis Factor (TNF), a signaling molecule that binds to cell receptors and can trigger a wide range of different responses [5]. In particular, TNF can induce death in cancer cells by activating specific downstream signaling pathways, thus restraining the growth of a tumor. To simulate the effect of TNF, a sophisticated Boolean network is used for modeling intracellular signaling and cell fate [6].

To execute a simulation, PhysiBoSS receives as input a XML file composing a number of values describing the TNF-based treatment methodology. Each combination of TNF parameters that compose a particular treatment results in simulations the utility, in term of killing cancer cells, and the computational demands of which are impossible to estimate beforehand. This is due to the complexity of the employed simulation models and the sensitivity to altering the parameters from one tested treatment to another. Nonetheless, we need to apply in advance to HPC administrators in order to reserve (a) the cores that we will need for in-silico trials and (b) the time simulations would need to run those trials. For instance, in PRACE (https://prace-ri.eu/), proposals for granting access on the provided HPC infrastructure should explicitly declare the Total Core Hours. In our setup, a core hour refers to the number of processor units (cores) used to run simulations, multiplied by the duration of the job in hours. With 24 core hours one can simulate for 1 day on a 1 core machine, half a day on a 2 core machine and so on.

In this work, we propose RATS (Resource Allocator for Tumor Simulations) [7] which is the first, to our knowledge, resource allocator for optimizing the execution of tumor simulations over HPC infrastructures. RATS focuses on learning (i) the trade-offs between *execution time* simulations require to complete vs. the available *number of cores* and (ii) the *utility* of treatment methodologies on tumor simulations. In particular, RATS incorporates a novel combination of active learning and Bayesian Optimization-based estimators to predict the above performance criteria. RATS optimizes the execution of in-silico trials solving all the aforementioned practical barriers. Our contributions are:

1. RATS is the first optimizer that simultaneously (i) dictates the optimal number of cores for executing any given number of simulation trials, (ii) reduces the total simulation time under given core capacity constraints, and (iii) predicts potentially effective treatment methodologies and prioritizes their simulated execution higher compared to expectedly ineffective ones. RATS relieves life scientists from the responsibility of guessing (and frequently miscalculating) the core hours needed for simulations. Instead, it allows to focus on effectively applying field expertise on testing treatment methodologies.
2. We present the novel internal architecture of RATS.
3. We introduce RATS+, an extended version of the initial framework that exploits optimization models acquired from prior tumor studies to more rapidly optimize new in silico studies, by fostering a Transfer Learning paradigm.
4. We outline lessons learned throughout our endeavor of developing RATS and RATS+, as best practices for future similar efforts.
5. We provide an elaborate empirical analysis on real performance data from more than 2500 tumor simulations executed on the MareNostrum 4 supercomputer, one of the most powerful HPC infrastructures in Europe. The results illustrate the ability of our optimization approach to select the most suitable resource allocations for simulations and eventually cut down time to market for new therapies facilitating the transfer of promising in-silico trials to in-vitro and in-vivo ones.

Table 1
Symbols used throughout the manuscript.

Symbol	Explanation
tnf_freq	Frequency of TNF injection in minutes
tnf_dur	For how long TNF is administered each time
tnf_conc	TNF concentration each time
SS	All combinations of $\{tnf_freq, tnf_dur, tnf_conc\}$
$\|SS\|$	Cardinality of the set SS
$SS_{done}(t)$	Completed simulations from SS at time t
S_i	Simulation $S_i \in SS$
k	Number of cores for a simulation
$ValCor$	Set of possible core configurations{2,...,max_cores}
S_i^k	Simulation $S_i \in SS$ being executed using k cores
max_cores	Maximum allowed number of cores for a simulation
$THP_i(k)$	Throughput (number of tuples being processed per time unit) using k cores for simulation S_i
UTL_i	Utility of simulation S_i assigned by the life scientist
$ST_i(k)$	Simulation Time for S_i executed with k cores
ST	Total simulation time for all simulations in SS
cap	Core capacity constraints
N	Sample simulation budget

2. Application problem description

TNF-based treatment methodologies are described by (i) the TNF Administration Frequency, which shows how often the drug is injected and is measured in minute intervals, (ii) the TNF Duration, dictating for how long the drug is administered, and (iii) the TNF Concentration each time the drug is applied. These parameters span continuous value ranges and therefore the number of possible simulations is infinite.

We first identify value ranges for the involved parameters that we want to study each time. We then discretize these parameter value ranges in a fine-tuned way to end up with a finite set of parameter combinations that corresponds to the treatment methodologies under study. This number of potential treatment methodologies is equivalent to the total number of simulations that need to run. Hence, we have a separate $[low, high, step]$ triplet for each of the TNF Frequency (tnf_freq), TNF Duration (tnf_dur) and TNF Concentration (tnf_conc) parameters. The whole simulation set (SS) is described by all possible combinations of:
`tnf_freq` $\in$ `[tnf_freq_low, tnf_freq_high, tnf_freq_step]`
`tnf_dur` $\in$ `[tnf_dur_low, tnf_dur_high, tnf_dur_step]`
`tnf_conc` $\in$ `[tnf_conc_low, tnf_conc_high, tnf_conc_step]`

2.1. Preamble on involved optimization problems

In Sections 2.2 and 2.3 we formulate two optimization problems. The first optimization stage is a prerequisite for RATS, because it helps in estimating the optimal core number per simulation, which directly relates to the total, optimal core number and core hours that are needed to submit the entire set SS for execution at once. If the first stage shows that core capacity constraints do not allow simultaneously submitting all simulations in SS, RATS proceeds to the second optimization stage which solves a Multiple-Choice Knapsack problem, accounting for core capacity constraints. Both the optimization problems do not directly optimize for core hours but, as we explain in Sections 2.5 and 3, having derived the solutions of these stages, core hours can trivially be estimated. Table 1 summarizes the main symbols used in the paper.

2.2. Number of cores per simulation

The first optimization problem that we need to solve involves the number of cores that should be devoted to each individual simulation $S_i \in SS$ characterized by a specific $\{tnf_freq, tnf_dur, tnf_conc\}$ triplet. Practically, we do not want to over-utilize or under-utilize cores for a simulation, but justly devote to it as many cores as it needs. Thus, for each individual simulation S_i we need to solve:

$$argmax\ THPRatio_i(k) = \frac{THP_i(k)}{THP_i(k/2)} \quad (1)$$

subject to $k \in ValCor = \{2, \dots, max_cores\}$.

The above problem formulation says that the optimal number of cores, i.e., the optimal value for k for a given simulation $S_i \in SS$ representing a single $\{tnf_freq, tnf_dur, tnf_conc\}$ combination, is the one that maximizes the throughput ratio (THPRatio) yielded when we choose to increase the number of cores from $k/2$ to k. Throughput (THP) is defined as the number of tuples being processed by the PhysiBoSS simulator, per time unit (secs) and there is an upper bound max_cores on the number of cores we allow to be devoted to a single simulation $S_i \in SS$.

For instance, consider the case when a simulation provides a throughput of 10M tuples/sec with $k = 1$ core, 20M tuples/sec with $k = 2$ cores, 30M tuples/sec with $k = 4$ cores. Then, upon deciding to switch from 1 to 2 cores, the above throughput ratio is 2 and upon switching from 2 to 4 cores, the respective ratio is 1.5. So the optimal core choice for this example is $k = 2$ because it constitutes the sweet spot where the benefits in terms of throughput increase are proportional to the increase in computational resources we devote.

In reality, the formulation of Eq. (1) is quite intuitive and incorporates the practical observation that after a certain sweet spot, PhysiBoSS progressively stops exploiting parallelism, since the biological processes it models commence to certain aggregation points (similar to AllReduce operations [8]). In particular, PhysiBoSS is built on PhysiCell [9], which handles the physical interactions and agent-based behaviors of cells, and MaBoSS [10], which models the intracellular signaling networks using a continuous time Boolean modeling formalism. PhysiCell includes BioFVM [11], a partial differential equation solver that handles the simulation of the micro-environment, i.e. the uptake, secretion and diffusion of molecules such as nutrients and drugs. In PhysiBoSS, each cell is treated as an agent that, to some extent as described shortly, evolves independently in its physical environment and intracellular states. However, inter-cellular state updates, such as agents competing for nutrients or diffusing molecules, do exist. Let us outline some fundamental factors favoring throughput ratio upon increasing the parallelism, as well as those that may restrict the benefits of parallelism. Important factors favoring the throughput ratio upon devoting more cores involve the different levels of parallelism that PhysiBoSS simultaneously exploits, including:

- **Intra-cell State Parallelism**: Each cell is modeled in PhysiBoSS as an independent agent. Each such agent has its own intracellular network and physical interactions. The computations that lead to a certain cell evolution, such as cell division, death, or movement and signaling pathways (via MaBoSS) can be done in parallel because these behaviors typically depend on local conditions rather than global ones. By distributing these independent computations across multiple cores, even in a per cell fashion, the intra-cell level of parallelism can accelerate the overall simulation and, thus, increase the throughput ratio.
- **Handling Multiple Cells Simultaneously**: When simulating a large population of cells, the work required scales with the number of cells. This level of parallelization is particularly beneficial when dealing with large cell populations or complex environments that require substantial computational resources. An important remark here is that the effect of parallelism depends on the evolving number of agents, for instance, a simulation with limited cell divisions will stop favoring increased parallelism after a certain sweet spot. A second important remark is that, inter-cell interactions may pose computational barriers halting parallel threads.

- **Multiscale Integration**: PhysiBoSS integrates intracellular Boolean models with extracellular physical modeling. It allows four scales of computation granularity, ranked based on their time step (from the greater to the smallest time step): (i) difussion_dt: motion of molecules from areas of high concentration to those of low (difussion solver), (ii) mechanics_dt: classical cell mechanics, for instance, cell movement and physical interactions (classical cell mechanics solver), (iii) phenotype_dt: for cell state update, (iv) intracellular_dt: time step in which MaBoSS is run. These layers of computation can be executed in parallel since they involve different scales, enhancing the overall performance. The potential independence of scales allows parallel execution of different parts of the cell and, thus, of the simulation without needing to wait for each step to finish sequentially.

The extent at which updates are interdependent involves inter-cell interactions as we describe below, which also involves the number and frequency of AllReduce operations [8]. On the other hand, the effect of increasing parallelism on the throughput ratio may be restricted by both simulated biological phenomena and technical factors:

- **Cell State Synchronizations**: Even though cells can, to some extent, evolve independently as described above, some global changes or communications (such as the impact of an applied treatment) may require coordination between cells. For instance, external elements that impact every cell (such as TNF $\{tnf_freq, tnf_dur, tnf_conc\}$ treatments or diffusing molecules) necessitate regular state changes, possibly causing synchronization barriers.
- **Environmental Factors as Global Parameters**: BioFVM can simulate release and uptake of many substrates by cell and bulk sources, diffusion and decay in large 3D domains. Certain external conditions in the experiment, like the spread of nutrients or substances outside cells (such as oxygen or substances that promote growth), affect the entire system. These factors frequently need computations that involve every cell in the environment, and their overall effect must be coordinated among the entire cell population. Hence, communication between cores or threads may hindering parallelism efficiency.
- **Thread Management Overhead**: In situations with high levels of parallelism, the overhead from handling threads may become notable, in combination with the frequency of Cell State Synchronizations and global parameters mentioned above. Thus, the speedups can be limited by the overhead of managing parallelism when multiple processes are running at the same time. Inefficient thread load balancing can also result in a lack of effective utilization of available computational resources. However, the exact load based on the discussed parallelization levels (intra-cell, inter-cell, multi-scale levels) cannot be known beforehand.

In all, the calculations for a simulation include intra-cell computations (mostly independent) and others that reduce the benefits of parallelism, like cell interactions, environment changes, and thread coordination. Consequently, manually optimizing the degree of parallelism becomes extremely challenging.

Given our above discussion, Fig. 2 plots the highest throughput ratio achieved (Fig. 2(a)) and the core increments $k/2 \rightarrow k$ that provide the highest throughput (Fig. 2(b)) for a subset of tumor simulations used in our experiments. To keep the plots readable, we plot only a sample of simulations.

In Fig. 2(a), the horizontal axis includes the respective $\{tnf_freq, tnf_dur, tnf_conc\}$ triplets. On the right hand side, we observe that there is a variety of sampled simulations where the best throughput ratio is achieved just above or below 2. Recall that the vertical axis shows the $k/2 \rightarrow k$ transition that achieves the best throughput ratio for each $\{tnf_freq, tnf_dur, tnf_conc\}$ triplet of the horizontal axis. Notice that, in this part of the sample, the annotated $k/2 \rightarrow k$ increments include all possible choices of increments $2 \rightarrow 4$, $4 \rightarrow 8$, $8 \rightarrow 16$ and $16 \rightarrow 32$. The left and middle side of Fig. 2(a) shows a variety of sampled simulations where doubling the number of cores gives a best throughput ratio between 3 and 4, while there is quite a number of simulations where doubling the number of cores achieves a best throughput ratio between 6 and 10 or more. Indicatively, notice, that for the $(300, 30, 0.035)$ triplet the best choice of increasing the cores to 16 ($8 \rightarrow 16$ annotation), achieves a throughput ratio above 10, while for the triplet $(400, 40, 0.035)$, $8 \rightarrow 16$ achieves the highest throughput ratio for that triplet, but the throughput ratio itself is around 4. This shows both the variability of the proper choice depending on the TNF parameters and the fact that the throughput ratio itself can greatly vary for equivalent of core increments.

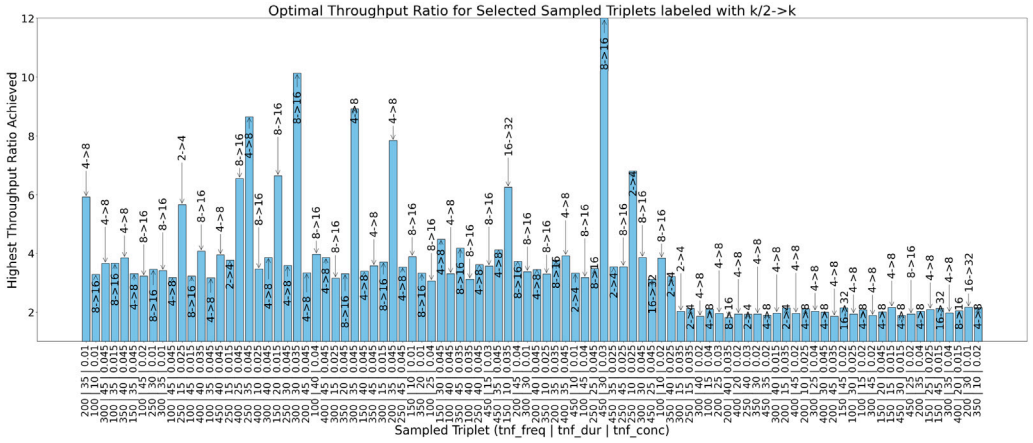
These phenomena exist due to the complexity of the simulated tumors evolution and the favoring or restrictive factors analyzed in our previous discussion. For instance, to explain the superlinear increase in throughput with the increase from 8 to 16 cores for $(300, 30, 0.035)$, one has to consider the multi-cell, intra-cell and multi-scale parallelization levels. Each time we increase the number of cores, we also increase the number of threads. We will explain what happens in this case with a small example of few cells. Assume that for the same $\{tnf_freq, tnf_dur, tnf_conc\}$, a single core has been devoted to handle 4 cells in 4 threads. Then, each thread will have to handle the processes of an entire cell. If we make the transition to 2 cores, then we have 2 cells to be handled by one core with 4 threads. This can unlock the intra-cell parallelism mentioned earlier. If we further increase the cores to 4, then we have 4 threads per cell and diffusion, cell mechanics and phenotype parallelism can also be unleashed.

The reason why these trends differ for different input parameters, such as in the case of $(400, 40, 0.035)$ and $8 \rightarrow 16$ increment, is attributed to the aforementioned limiting factors of cell interactions, environment changes, and thread coordination. For instance, if the $\{tnf_freq, tnf_dur, tnf_conc\}$ represents a successful treatment, then we will have fewer tumor cell divisions. Or, even if cell divisions exist and are really speedy, the impact of the applied treatment and competition for nutrients can increase the communication between cores, incurring synchronization barriers and elevated thread management overhead.

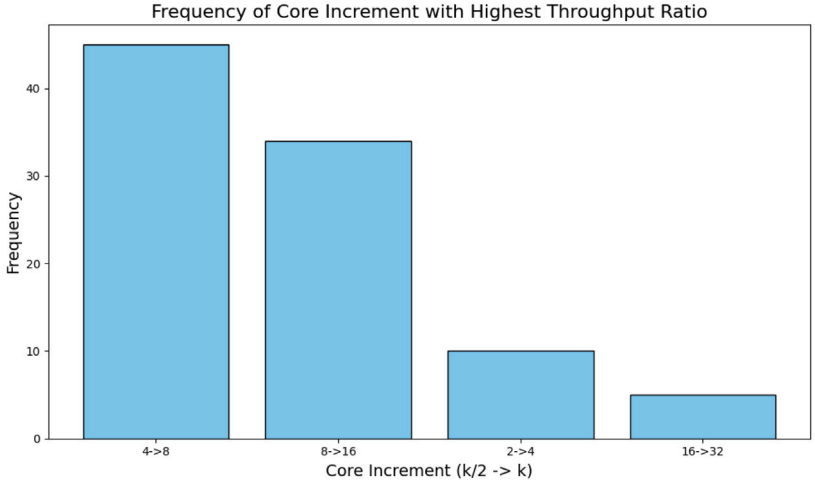
Fig. 2(b) shows a complementary view of Fig. 2(a) where for the same set of sampled simulation triplets, the horizontal axis holds the core increment that achieves the highest throughput ratio and the vertical axis the frequency (number of times) this happens in the sampled simulations. We can observe that $4 \rightarrow 8$ and $8 \rightarrow 16$ are the most frequent winners that achieve the highest throughput ratio. Later on, we will see that this is in accordance with our experiment in Fig. 6 where the optimal (and the RATS) choices lie between using always 8 or always 16 cores. However, note that (i) the number of cores that provide the optimal throughput ratio in Fig. 2 is made known only after executing all simulations and (ii) simply using the throughput ratio as a guide for resource allocation does not account for constraints in the number of available cores compared to the number of pending simulations. The later issue is discussed in Section 2.3.

Note that the exact way one factor dominates the other throughout the lifetime of a simulation cannot be known beforehand, which is the fundamental motivation of RATS as mentioned in Section 1. Back to Eq. (1), notice that, if we can derive the optimal number of cores per simulation, we can also extract the optimal, total number of cores for SS. As we explain in Section 2.5 we can also have an estimation about the execution time of SS. Therefore, we can compute the total core hours.

Although the above optimization problem is a prerequisite to obtain the total, optimal number of cores, if the cardinality $|SS|$ of SS is high, requesting to reserve the total, optimal number of cores becomes infeasible. For instance, HPC technical guidelines pose upper bounds on the allowed number of cores [12]. In such a case, we need to also solve a different optimization problem which accounts for an upper bound on core capacity.



(a) Optimal Throughput Ratio for Selected Sampled Triplets labeled with k/2 → k



(b) Core Increment Achieving the Highest Throughput Ratio

Fig. 2. Throughput ratios and core increments for a sample of triplets.

2.3. Simulation time under core capacity constraints

In this setup, we want to optimize (minimize) the total Simulation Time (ST) required for SS under given core capacity (cap) constraints. If we have knowledge of the optimal ST along with the capacity as input for the current problem, we can subsequently calculate the total number of core hours required. We model this optimization problem as a Multiple-Choice Knapsack problem [13]. Let us examine the rationale behind opting for this approach.

We have a set of simulations SS composed of all possible $\{tnf_freq, tnf_dur, tnf_conc\}$ triplets and each simulation can be executed with $k \in ValCor$. The total core capacity constraint, denoted as cap, is our knapsack and the multiple choice version of it comes from the fact that if we choose to execute S_i with a certain number of k' cores, i.e., $S_i^{k'} = \{tnf_freq, tnf_dur, tnf_conc, k'\}$, we eliminate all other choices of k for S_i, i.e. we would not repeat the same simulation with different k since we already know the effect of the respective treatment. Moreover, each possible choice S_i^k entails a specific simulation time $ST_i(k)$, which is the value of that choice and k is its weight.

Considering that simulations are executed concurrently in parallel, our objective is not to optimize the sum of $ST_i(k)$, but instead reduce the maximum $ST_i(k)$ which defines the future timepoint at which all simulations will have completed. This optimization problem would be defined as $minimize \max_{S_i \in SS, k \in ValCor} ST_i(k) \cdot b_{ik}$, for $b_{ik} = 1$ only if we choose S_i^k, and zero otherwise. The traditional Knapsack definition involves a maximization problem and thus, we alter the above formulation to match a maximization problem, also negating $ST_i(k)$s.

$$maximize \min_{S_i \in SS \setminus SS_{done}(t), k \in ValCor} -ST_i(k) \cdot b_{ik} \tag{2}$$

subject to

$$\sum_{k \in ValCor} b_{ik} = 1, \forall S_i \in SS,$$

$$b_{ik} \in \{0,1\}, \forall S_i \in SS, \forall k \in ValCor,$$

$$\sum_{S_i \in SS \setminus SS_{done}(t), k \in ValCor} k \cdot b_{ik} \leq cap(t) \leq cap$$

Notice that at time t only a subset of SS can be included in the knapsack. The rest of the simulations that we need to run remain in a queue. At any given point t there is a $SS_{done}(t) \subseteq SS$ with completed simulations. Having $SS_{done}(t)$, we need to solve the optimization problem of Eq. (2) again, this time having a Knapsack with capacity $cap(t)$. $cap(t)$ denotes the number of free/available cores at time t. This explains the temporal reference (t) in Eq. (2).

2.4. Utility-based prioritization

To align with our objective of expediting the time to market for new therapies, we aim to convert SS into a utility-based priority queue that prioritizes simulations based on their expected utility in terms of cancer cell eradication. For this, the simulations should be ordered by decreasing utility order. $UTL_i \in [0,1]$ denotes the utility of simulation

$S_i \in SS$ (Table 1), and every time $SS_{done}(t)$ frees resources, simulations will be chosen from $S_i \in SS \setminus SS_{done}(t)$ using that order.

2.5. Motivation for Bayesian optimization

In order to solve the optimization problems we describe in Sections 2.2 and 2.3, we need to have accurate estimations for ST_i and THP_i involved in Eqs. (1) and (2). Moreover, we also need UTL_i estimations for $S_i \in SS$ to transform SS into a utility-based priority queue.

Due to the complexity of the biological processes that are modeled, simulation studies under different $\{tnf_freq, tnf_dur, tnf_conc, k\}$ parameters constitute a black-box, the behavior of which cannot be described using some analytic mathematical formula. We therefore resort to learning that behavior using Machine Learning (ML) tools. In Appendix, we also provide an analysis, from real simulation executions used in our experimental section, showing that all $\{tnf_freq,\ tnf_dur,\ tnf_conc\}$ parameters appear equally important for the involved predictions and, at the same, a white box approach cannot reveal any obvious trends or correlations with respect to these input parameters.

The challenge we face is that for each new study we wish to conduct on treatment methodologies, we lack any training data. Therefore, it is necessary to carry out a small set of simulations and train ML models to learn the variables THP_i, ST_i, and UTL_i. We would like to keep this number of sample simulations to a minimum for two reasons:

- Simulations are time-consuming and occupy system resources.
- UTL_i can only be determined by life scientists after a simulation is completed and vizualized (see Fig. 1). Human experts can have the capacity of manually evaluating only a few tens of sample simulations on the effect of respective treatment methodologies.

Note that ST_i, THP_i are system metrics that can be automatically derived using workload managers. In MareNostrum 4, we use Slurm (https://slurm.schedmd.com/) for that purpose.

To sum up, we have an unknown black-box function that characterizes the system's behavior and our objective is to learn this behavior in terms of THP_i, ST_i, and UTL_i using a limited number of samples, because the number of function evaluations (simulations in our case) is severely limited by time, cost and human effort. Bayesian Optimization (BO) is the most suitable ML approach for this scenario [14,15].

BO optimizes an objective function f by iteratively evaluating the value of f in sampled points and constructing an estimate for the mean value of f over the set of all feasible points [15]. Formally, given an unknown objective function f we denote the search space as $\mathcal{X}$. Moreover, we assume that f has no simple closed form while it can be evaluated for each feasible point, i.e., $\forall x \in \mathcal{X}$. In particular, a noisy observation y_i of f can be derived for any $x_i \in \mathcal{X}$:

$$y_i = f(x_i) + \varepsilon \quad \text{,where} \quad \varepsilon \sim \mathcal{N}(0, \sigma^2)$$

$$\mathbb{E}[y|x] = f(x)$$

BO attempts through an iterative process to construct a belief that represents the f function. Specifically, at iteration $n \leq N$, it exploits the already evaluated information in order to guide the search of the next point x_{n+1} which will return the observation y_{n+1} of f. This process terminates after N iterations.

In general, BO is composed of two parts: (a) a statistical model used to estimate the objective function; and (b) an acquisition function used to efficiently sample the next points to be evaluated. A Gaussian Process Regression (GPR) approach is often followed to model the objective function. The Gaussian Process (GP) model used in the GPR approach serves as the statistical model of the optimization and provides a probability distribution that estimates the value of the objective function over the set of feasible points $\mathcal{X}$ [16,17]. The model consists of a probability distribution over possible functions that fit the set of evaluated points. This distribution is updated with each new evaluation of the objective function. A GP is defined by a mean function m and a covariance function or kernel:

$$f(x) \sim \mathcal{N}(m(x), kernel(x, x')), (x, x') \in \mathcal{X}$$

The kernel of the Gaussian Process describes the smoothness of the distribution and specifies the covariance between the values of the objective function at different points. In other words, it quantifies the similarity of the objective function evaluations between nearby points. In particular, the kernel function determines the functions that are most likely under the GP prior distribution and, thus, incorporates prior beliefs we have about the objective function. In our study, due to the discrete nature of $\{tnf_freq, tnf_dur, tnf_conc, k\}$, we use a Rational Quadratic Kernel [16].

The acquisition function allows us to select the next evaluation point in an informed manner, as it serves as a utility estimate for all feasible points. In particular, the acquisition function quantifies the contribution of the evaluation of the objective function to our estimation for each point. Commonly used acquisition functions are the Expected Improvement (EI), Probability of Improvement (PI), Lower Confidence Bound (LCB), Upper Confidence Bound (UCB) and Entropy Search (ES) [15]. Different acquisition functions estimate the importance of evaluating the objective function at potential points, taking into account different perspectives. They strike a balance between the exploration of unexplored regions and the exploitation of promising areas, considering the trade-off between these two aspects. Acquisition functions whose primary aim is the exploration of the parameter space select points for which the estimate of the objective function is of higher uncertainty, while those that aim at the exploitation of the space, sample points in which the expected mean value of the objective function is high.

Algorithm 1 Bayesian Optimization

1: Provide Sample Simulation Budget N
2: Place a Gaussian Process prior on f.
3: Evaluate f at n_o initial points. Set $n = n_0$. Update distribution based on initial evaluations.
4: **while** $n \leq N$ **do**
5: Find x_n that maximizes the acquisition function over $\mathcal{X}$.
6: Observe $y_n = f(x_n) + \varepsilon$,where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$.
7: Update posterior distribution of f using all observed f values.
8: Increment n
9: **end while**
10: Return trained GPR model to get queried and provide estimations for f.

Pseudo-code of the BO algorithm is shown in Algorithm 1. The initial step involves evaluating the objective function at n_0 randomly selected points from the feasible set. Subsequently, the acquisition function is employed to determine the next point to evaluate. This is done by selecting the point that maximizes the acquisition function. Once the next point is chosen, the objective function is evaluated at that point,[1] and the posterior distribution is updated using all the observed values of f. The aforementioned process (sampling of next point → evaluation of objective function → update of posterior distribution) is repeated until we reach a declared number of total evaluated points N.

In our application, function evaluations correspond to runs of sample simulations. Additionally, we have 3 objective functions, namely throughput (THP), simulation time (ST) and simulation utility (UTL), while $\mathcal{X}$ is composed of $S_i \in SS$ for UTL and $\{S_i^k\}, \forall S_i \in SS$, $\forall k \in ValCor$ for ST, THP. For reasons we explain in Section 3.1,

[1] Note that in the general case of the BO, the observations of the objective function can be noisy and as such a noise-related term has been included in line 6 of the Algorithm 1, denoted as ε.

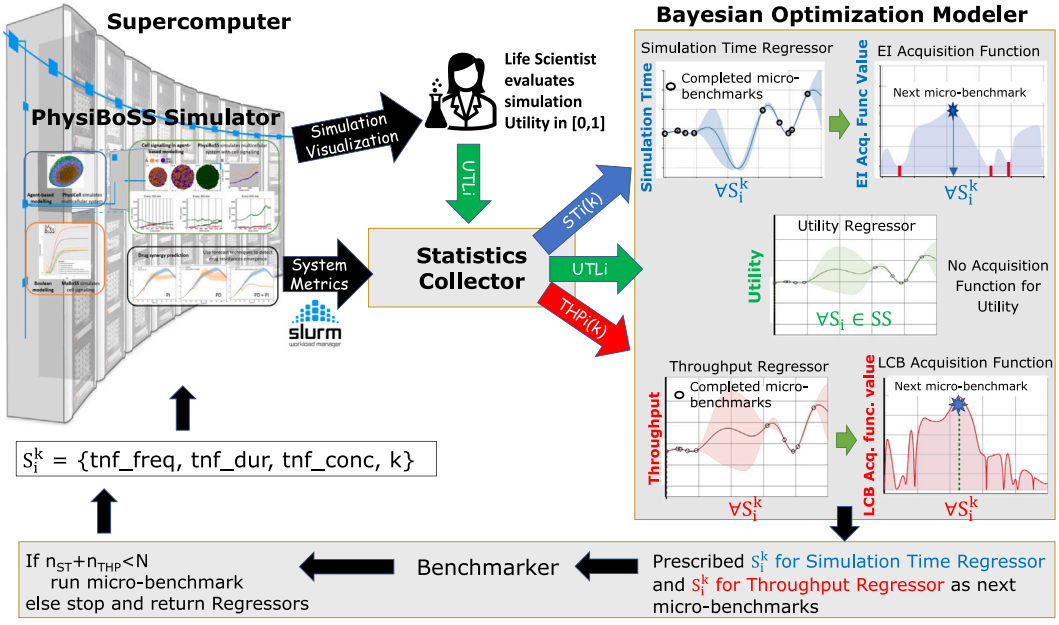


Fig. 3. Overview of the RATS modeler.

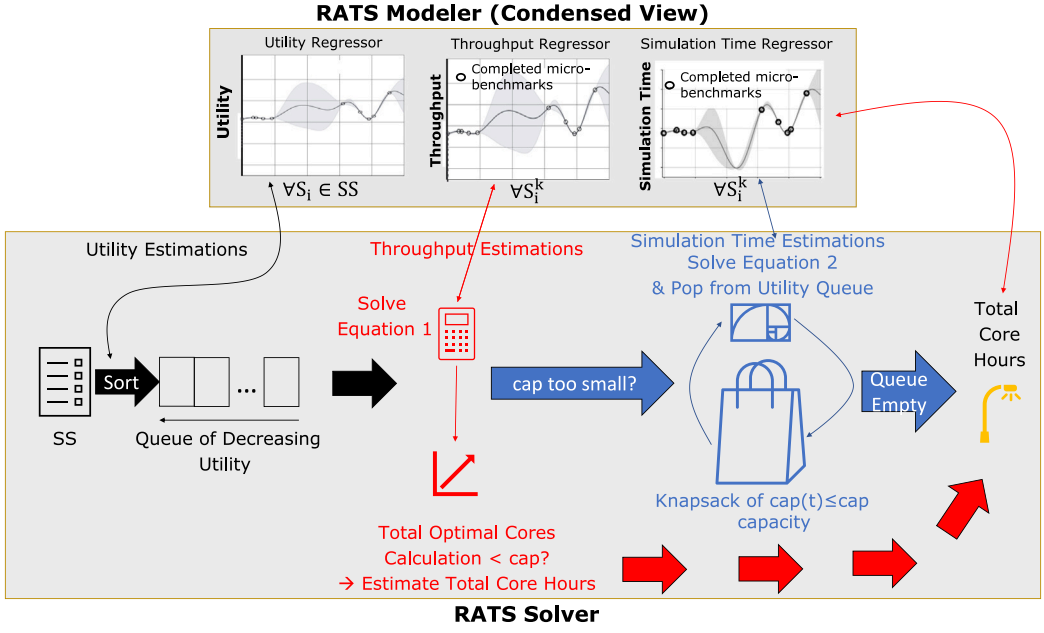


Fig. 4. RATS Architecture — RATS solver and Built RATS modeler. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

instead of using 3 acquisition functions for the 3 objective functions, we only use 2. Moreover, we set $n_0 = 0$ for each of them. We found out that this approach dramatically decreases the required number of sample simulations and we experimentally prove that this happens without practically sacrificing the accuracy of GPRs.

3. The RATS resource allocator

The RATS resource allocator we have developed is composed of two principal components: the RATS Modeler and the RATS Solver. These components have distinct internal architectures and serve different purposes.

3.1. The RATS modeler

The RATS Modeler is used to build estimators for $THP_i(k)$, $ST_i(k)$ and UTL_i involved in Eqs. (1) and (2). The assumption is that RATS already has access to a minimum, small-scale core capacity, which is typically the case for life scientists employed by organizations hosting HPC infrastructures. To facilitate the resolution of the optimization problems stated in Eqs. (1) and (2), the RATS Modeler executes sample simulations in order to build GP regressors. This process iterates in loops, as illustrated in Fig. 3 and Algorithm 2.

Step 1: The RATS Modeler takes as input a set SS, which consists of triplets as described at the beginning of Section 2. Additionally, it receives the budget N for sample simulations that it

Algorithm 2 RATS Modeler

1: Receives the SS set and the Simulation Budget N.
2: Defines BO hyperparameters (Section 5).
3: $n = 0$.
4: **while** $n \leq N$ **do**
5: Find x_n that maximizes the acquisition function over $\mathcal{X}$ for the Throughput Regressor.
6: Find x'_n that maximizes the acquisition function over $\mathcal{X}$ for the Simulation Time Regressor.
7: Benchmarker creates the XML files and submits them to PhysiBoSS simulator.
8: PhysiBoSS simulator executes the simulations and produces the $ST_i(k)$, the $THP_i(k)$ and the visualization of simulated tumors.
9: Life scientists evaluate the visualization providing the UTL_i.
10: Statistics Collector feeds $ST_i(k), THP_i(k), UTL_i$ to the corresponding regressors.
11: Increment n
12: **end while**
13: Return trained GPR model to get queried and provide estimations for f.

can execute to train GPRs using Bayesian Optimization. The values of the simulation budget N and the BO hyperparameters are determined based on the knowledge and experience gained during the development of RATS, which are discussed separately in Section 5.

Step 2: The acquisition function for the Throughput Regressor asks for a sample simulation S_i^k to get executed. The same applies for the acquisition function for the Simulation Time Regressor. All BO-based Regressors and their acquisition functions are included in a separate subcomponent termed, Bayesian Optimization Modeler, on the right of Fig. 3. The BO Modeler interacts with a Benchmarker subcomponent asking for this pair of simulations to get executed.

Step 3: The Benchmarker subcomponent (bottom of Fig. 3) creates a pair of XML files that configure the respective runs of the PhysiBoSS simulator and submits each of this pair of jobs (left of Fig. 3). In particular, PhysiBoSS input could have, for instance, the following form and information:

```
...
<user_parameters>
  ...
  <tnf_pulse_period type="double" units="
      min">150.0</tnf_pulse_period>
  <tnf_pulse_duration type="double" units=
      "min">10.0</tnf_pulse_duration>
  <tnf_pulse_concentration type="double"
      units="TNF/um^3">0.0015</
      tnf_pulse_concentration>
  ...
</user_parameters>
...
```

Note that, tnf_freq corresponds to `tnf_pulse_period`, tnf_dur to `tnf_pulse_duration` and tnf_conc to `tnf_pulse_concentration`.

Step 4: Upon the completion of a submitted simulation there are two types of outputs: (a) system metrics regarding the simulation time $ST_i(k)$ and $THP_i(k)$ and (b) visualization of the simulated tumor under the $S_i \in SS$ treatment methodology used. Simulation visualizations are presented to the life scientist who assigns a value in $[0, 1]$ for the utility UTL_i of $S_i \in SS$. This happens for the simulations, either initially requested by the Throughput Regressor or the Simulation Time Regressor.

Step 5: Both system measured and human-assigned metrics are collected by a Statistics Collector subcomponent (middle of Fig. 3). The Statistics collector feeds system metrics to the Simulation Time and Throughput Regressors, respectively. It also feeds the utility value to the Utility Regressor which, on the one hand, does not have its own acquisition function, on the other hand receives utility value input for all simulations requested by either of the other 2 acquisition functions.

The above steps are repeated until the available simulation budget N is depleted. One could potentially train the Throughput Regressor and then proceed with training the Simulation Time Regressor. Moreover, sample simulations that are asked by one acquisition function and have already been requested and executed due to the other can be directly fed to the BO Modeler by having the Benchmarker and Statistics collector subcomponents keeping a recent history of S_i^ks and statistics, respectively. In that case, we do not increase the counter at Line 11 of Algorithm 2. All these optimizations are orthogonal to the procedure we describe above.

3.2. The RATS solver

The RATS Solver component comes into play after the RATS Modeler has built its estimators. The RATS Solver focuses on (a) prioritizing simulations based on their expected utility, (b) solving the optimization problems defined in Eqs. (1) and (2) using predictions provided by the regressors of the RATS Modeler. It is important to highlight that the RATS Solver solely relies on the RATS Modeler to obtain estimations of THP_i, ST_i and UTL_i. It does not utilize any HPC resources whatsoever. The pseudo-code of the RATS Solver is illustrated in Algorithm 3. In more detail the Solver operates as follows:

Part 1: The RATS Solver receives as input the set SS in text format as the RATS Modeler does. It also receives the core capacity constraint cap. It then converts SS into a priority queue by sorting the text tuples based on the expected utility of each S_i. For that purpose, the Solver derives the expected utility of each $S_i \in SS$ by querying the Utility Regressor previously built by the RATS Modeler. This is illustrated on the left-hand side of Fig. 4. Then, it replicates the sorted tuples adding possible core numbers k, the throughput $THPRatio_i(k)$ of which we want to estimate (not shown in the figure for readability purposes). These are also the items we consider in our Knapsack (if needed).

Part 2: The RATS Solver solves Eq. (1) deriving the optimal number of cores for each and every $S_i \in SS$. To compute $THPRatio_i(k)$ for $S_i \in SS$, it queries the Throughput Regressor of the RATS Modeler (middle of Fig. 4). The optimal number of cores per simulation is summed up to compute the total, optimal number of cores for SS.

Part 3: If this total number is smaller than cap, all simulations can be submitted at once. Then, the RATS Solver queries the Simulation Time Regressor of the RATS Modeler to find out the estimated total simulation time, based on which it computes the total core hours and returns it to the life scientist. In that case, the RATS Solver stops. This is shown in Fig. 4 using red-colored arrows. Otherwise, it proceeds to Part 4.

Part 4: If the capacity (cap) constraint prevents the submission of the set SS with the optimal number of cores all at once, the RATS Solver may continue to solving the optimization problem in Eq. (2) as shown with the blue-colored arrows and icons in Fig. 4. Recall that this problem is solved in rounds. In each round, as described in Section 2.3, some

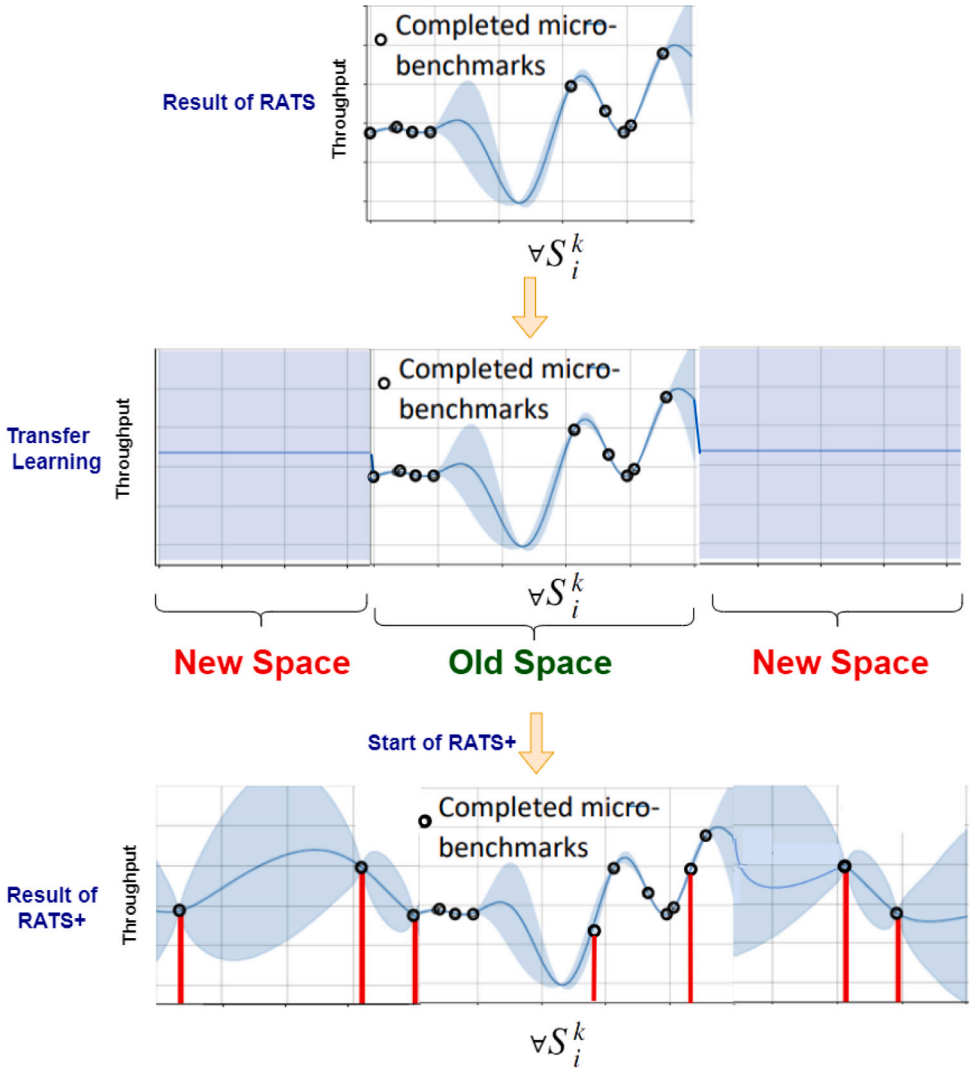


Fig. 5. Illustration of transfer learning application of RATS+ over the throughput regressor. In the general case, D_S, D_T may only partially overlap. The case of utility and simulation time regressors is analogous.

S_i^ks from the utility-based priority queue are selected to be included in the Knapsack so that the minimum value of the included simulations (negated simulation times ST_i) is maximized. In that case, the RATS Solver queries the Simulation Time Regressor of the RATS Modeler to obtain estimations about $ST_i(k)$ at the beginning of this step, but also at any subsequent time t upon there are simulations entering $SS_{done}(t)$ and freeing resources to $cap(t)$.

Final Part: If the priority queue is empty at some point, the execution of the RATS Solver concludes and the number of core hours is returned as the product of cap and the *aggregated* minimum execution time from solving the Multiple-Choice Knapsack at the various timepoints t.

4. RATS+: Reducing optimization effort via a transfer learning plug-in

As mentioned earlier, the RATS approach incorporates two different optimization problems (see Sections 2.2 and 2.3). We remind the reader that each TNF-based treatment methodology can be described by the (i) TNF Frequency, (ii) TNF Duration and (iii) TNF Concentration. In the challenging and time-consuming research task of identifying the most effective drug combination, life scientists may modify the combinations under investigation over time. For instance, at the early stages of the study, the range of the examined TNF Concentration could be defined within the interval $tnf_conc \in [0.01, 0.045, 0.005]$, but after a number of in silico trials, we may observe the output and suspect that higher values of concentration (i.e., $tnf_conc > 0.045$) may result to more effective treatments. In such a case, it would be beneficial for the tumor study, and as such for RATS, to be able to exploit the knowledge gained so far from the already optimized simulations in order to reduce the optimization effort in the new series of simulations whose domain, in terms of tnf_freq, tnf_dur, tnf_conc ranges, partially overlaps or fully contains the parameter ranges we have optimized so far. The task involves the integration of an additional module into RATS, aimed at decreasing the simulation budget N and consequently reducing the training time of the RATS Modeler for the new study, without sacrificing the accuracy of the developed regressors. In response to the aforementioned scenario, *Transfer Learning* [18] provides an appropriate framework [19] that is employed in order to avoid the need for training the RATS Modeler from scratch. Therefore, here we present an extended version, namely RATS+, which seamlessly incorporates into the original RATS framework to effectively attain the aforementioned objective.

Formally, in Transfer Learning, *domain* D contains (i) a feature space $\mathcal{X}$; and (ii) a marginal probability distribution $Pr(X)$, where $X =$

Algorithm 3 RATS Solver

Receives the SS set and the core capacity constraint cap.
for $S_i \in SS$ **do**
    Query to Utility Regressor which returns the expected utility UTL_i of S_i.
end for
SortedQueue $\leftarrow$ Sort in descending order SS wrt the expected utility of each simulation.
$opt_num_cores = 0$
for $S_i \in SS$ **do**
    For S_i solve Equation (1) by asking Throughput Regressor the $THP_i(k)$ values.
    Update opt_num_cores
end for
if $opt_num_cores \leq cap$ **then**
    Query Simulation Time Regressor to estimate the total simulation time.
    Compute the total core hours.
else
    while SortedQueue is not empty **do**
        Query Simulation Time Regressor.
        Solve Equation (2).
        Pop simulation(s) from SortedQueue.
        Update opt_num_cores.
    end while
end if
Return total core hours.

$\{x_1, \ldots, x_n\} \in \mathcal{X}$. Subsequently, given a domain $\mathcal{D} = \{\mathcal{X}, Pr(X)\}$, a *task*, $\mathcal{T}$, consist of (i) a label space $\mathcal{Y}$ and an objective predictive function $f(\cdot)$ [18]. Thus, Transfer Learning can be defined as follows [18]:

Definition 1. Given a source domain $\mathcal{D}_S$ and learning task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$ and learning task $\mathcal{T}_T$, Transfer Learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in $\mathcal{D}_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$.

Pan and Yang [18] present a categorization of Transfer Learning based on the various conditions between $\mathcal{D}_S, \mathcal{D}_T, \mathcal{T}_S$ and $\mathcal{T}_T$. Based on the aforementioned categorization, RATS+ can be characterized as a *transductive Transfer Learning* setting, since $\mathcal{D}_S \neq \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T$. The interested reader is referred to [18] for a more detailed description of the categories.

In RATS+, we employ the notion of Transfer Learning in order to equip RATS with knowledge gained from related domains, saving computational resources and training time in both optimization problems (see Sections 2.2 and 2.3). Formally, during the first phase, i.e., in domain $\mathcal{D}_S = \{\mathcal{X}, Pr(X)\}$, RATS approach which is guided by the acquisition function, evaluates some sample points in order to fit our estimator (i.e., GP regressor). As such, after N iterations, where N is the simulation budget, the posterior that has been generated by RATS is $Pr(f|X, y)$, where $X \in \mathcal{X}$ is the set containing the evaluation points and y are the observed samples. The posterior can then be utilized to produce predictions $f^\star$ given an input of new samples, denoted as $X^\star \in \mathcal{X}$, using the formula $Pr(f^\star|X^\star, X, y)$. Now, when we move to domain $\mathcal{D}_T$, we have to adjust the posterior generated earlier in $\mathcal{D}_S$ in order to construct our prior and as such to be able to exploit the gained knowledge. Specifically, we define our prior in $\mathcal{D}_T = \{\mathcal{X}_+, Pr(X_+)\}$ as follows:

$$Pr(f|X_+, X, y) \sim \mathcal{N}(f|(m_+, m), (K, A))$$

where $X_+ \neq X$ and $X_+ \cap X \neq \emptyset$, m_+ is the mean vector in the new regions, i.e., regions that were not defined in $\mathcal{D}_S$ and m is the mean vector that was constructed by RATS. Moreover, K denotes the Rational Quadratic kernel and is utilized for the new regions, while A is the kernel constructed by RATS in $\mathcal{D}_S$. Given the prior described earlier and the acquisition function we can perform our next calls in $\mathcal{D}_T$ and update our Gaussian process regressor. In particular, the posterior that we derive after performing our evaluations has the following form:

$$Pr(f|X_+, y_+, X, y)$$

Note that by using the obtained posterior we can produce predictions $f^\star$ given an input of new samples, denoted as $X_+^\star \in \mathcal{X}_+$, using the formula $Pr(f^\star|X_+^\star, X_+, y_+, X, y)$.

Fig. 5 illustrates the idea of Transfer Learning in the RATS+ approach. In particular, we observe that given the result of the (first) RATS Modeler in an initial domain and task $\mathcal{D}_S$ and $\mathcal{T}$, our approach exploits the gained knowledge and "extends" it in a different domain, $\mathcal{D}_T$, in order to perform the same task $\mathcal{T}$. Note that, in Fig. 5 we illustrate an exemplary situation where $\mathcal{D}_S \subset \mathcal{D}_T$. In the general case, $\mathcal{D}_S$, $\mathcal{D}_T$ may only partially overlap. As shown in Fig. 5, our approach initially adopts a uniform belief with high uncertainty for the regions of $\mathcal{D}_T$ that were not contained in $\mathcal{D}_S$. Thus, our RATS+ Modeler will perform most of its initial acquisition function calls (for each of the involved regressors) to these un-explored regions in order to gain necessary information. However, we mention that some of the performed calls may refer to the region that belongs to $\mathcal{D}_S$ as well (see the red lines in the last figure that are located in the "Old Space"). Such a behavior can be observed in the following cases: (i) when the first RATS Modeler is unable to perform a sufficient number of calls, i.e., it has a very limited simulation budget, and as such its produced model has a high amount of uncertainty for several regions in $\mathcal{D}_S$; and (ii) when the parameters of acquisition function (Section 5) have a relatively small value resulting in a decrease of exploration rate, since small values benefit the exploitation of the knowledge that model already possess. The pseudo-code of the RATS+ approach is presented in Algorithm 4.

Algorithm 4 RATS+ Transfer Learning based Optimization

Load a pre-trained model from a relevant domain $\mathcal{D}_S$.
Define the new domain $\mathcal{D}_T$.
Provide Sample Simulation Budget N
Extend the pre-trained model of $\mathcal{D}_S$ by adding an uninformative prior to the regions $\mathcal{D}_T - \mathcal{D}_S$ in order to create and place a Gaussian Process prior on f.
Evaluate f at n_o initial points. Set $n = n_0$. Update distribution based on initial evaluations.
while $n \leq N$ **do**
    Find x_n that maximizes the acquisition function over $\mathcal{X}$.
    Observe $y_n = f(x_n) + \varepsilon$,where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$.
    Update posterior distribution of f using observed f values.
    Increment n
end while
Return trained GPR model to get queried and provide estimations for f.

5. Insights and lessons learned

Based on the experience gained from performing hyperparameter tuning of RATS, we have learned several valuable lessons. These lessons can be summarized as follows:

Kernel Selection: The kernel that provides the best estimation performance is the Rational Quadratic kernel parameterized by a length scale of 1.0 and α parameter of 2.5. Please refer to [16] for further details on these parameters.

Simulation Budget: The total simulation budget N for training RATS regressors can be set to a value between 5% and 10% of the cardinality

of SS. In other words, the sample simulations are 10 to 20 times fewer compared to $|SS|$.

No initial set n_0: Although the traditional BO, as described in Algorithm 1, entails a number of simulations n_0 in a warmup phase, our practical experience says that having set the kernel function as described above, this n_0 does not aid significantly in improving the final accuracy of the regressors. Therefore, we advise that the whole experimental budget is used with sample simulations driven by the respective acquisition functions.

Acquisition Function Choice for RATS: For training the Throughput Regressor of the RATS Modeler, an LCB acquisition function should be used. LCB favors exploration vs. exploitation (Section 2.5) and better captures trends in throughput instead of absolute values [20]. This seems to favor accurate estimations of a ratio, such as $THPRatio_i(k)$. The acquisition function of choice for the Simulation Time Regressor is EI instead, because now we are interested in absolute values. In both functions, the ξ_i and κ parameters respectively, should be set to a high value above 100 tuning the exploration vs. exploitation trade-off. Please refer to [15] for further details.

Acquisition Function Choice for RATS+: As mentioned earlier for the Throughput Regressor of the RATS Modeler, an LCB acquisition function should be used. However, our practical experience indicates that in the Transfer Learning scenario, the RATS+ approach achieves higher performance when the first RATS Modeler utilizes an LCB acquisition function while the extended RATS+ one uses an EI. The intuition behind this is that in RATS+ scenario, when we transfer our knowledge to a new domain, there are regions that are unexplored (since these regions were not available on the initial domain). Thus, such regions are characterized by uncertainty that is significantly higher compared to all the other regions that were defined in the initial domain.[2] As a result, the employment of the EI acquisition function guides RATS+ to focus on these new regions enabling our approach to achieve a better generalization since it can efficiently handle the explorations vs. exploitation dilemma.

Transfer Learning instead of an 'expert' pre-trained Optimizer: In our problem of interest, initially, we have domain $\mathcal{D}_S$ and the task $\mathcal{T}$. Subsequently, we attempt to exploit this gained knowledge in order to boost our performance on the same task $\mathcal{T}$ on a different domain $\mathcal{D}_T$, where $\mathcal{D}_S \neq \mathcal{D}_T$ and $\mathcal{D}_S \cap \mathcal{D}_T \neq \emptyset$. In this scenario, one has the option to utilize Transfer Learning or opt for an alternative approach. In particular, one could exploit the pre-trained model that was trained in $\mathcal{D}_S$ in order to derive all useful information for the regions of $\mathcal{D}_T$ that are overlapped with $\mathcal{D}_S$, i.e., $\mathcal{D}_S \cap \mathcal{D}_T$, while for the remaining regions a new model could be trained from scratch. Intuitively, in such an approach, the life scientist uses the pre-trained model as an 'expert' for common regions of the old and new domains, while for the new regions, a new model could be trained. However, such an approach suffers from a major drawback. In detail, the 'expert' optimizer derived from initial domain could achieve very poor performance due to several reasons. For instance, if its training process in $\mathcal{D}_S$ had a limited simulation budget then the produced model would be unable to capture the trends or absolute values and its expertise could be extremely limited. As a result, when we proceed to $\mathcal{D}_T$ its poor performance could drastically reduce the effectiveness of our approach. Moreover, the performance of the 'expert' optimizer could be also affected by the inefficient handling of the exploitation vs. exploration dilemma, since a limited exploration ratio could block the expert from gaining wide knowledge of the initial domain and focusing only on specific regions. Transfer Learning instead, provides an appropriate framework for exploiting and transferring expertise gained from $\mathcal{D}_S$ to $\mathcal{D}_T$. Thus, our approach does not have to be trained from scratch in the new domain, i.e., $\mathcal{D}_T$. Moreover, Transfer Learning enables RATS to handle the aforementioned scenario of poor performance in $\mathcal{D}_S$, since the utilized acquisition function could perform new evaluations not only to the new regions of $\mathcal{D}_T$ but also in several regions of $\mathcal{D}_S$ in case that these regions suffer from high uncertainty.

[2] Note that this effect is also illustrated graphically in Fig. 5.

6. Related work

Optimization via SS Pruning: There is a line of work that focuses on pruning the size of SS by predicting and distinguishing treatment parameters that yield useful simulations [21–25]. All these approaches are orthogonal to RATS and can be used in order to prune simulations of expected low utility from the set SS, thus reducing the size of the priority queue. RATS stands out from these approaches by effectively providing the necessary core hours to life scientists, which cannot be achieved by any of the alternative methods.

ML-based Resource Allocators: A different, relevant line of work comes from workload optimization over cluster, database and Big Data infrastructures [26]. EasyFlinkCEP [20] employs Bayesian Optimization to auto-tune the optimal parallelism of FlinkCEP programs. Wayeb [27] applies BO to find (near-)optimal training configurations for the Complex Event Forecasting Engine. Seagull [28] uses machine learning models to predict customer load per server, and optimize service operations for database systems in the cloud. MuMMI [29] optimizes resource allocation on the Sierra supercomputer by executing GPU-intensive CG simulations on nodes with four GPUs and 44 CPU cores. In this work, authors schedule four CG simulations per node, utilizing several cores for simulation and allocating the remaining CPU cores for other tasks. The Flux resource manager is employed in order to address the scheduling challenges and ensure the efficient utilization of the available resources while offering flexibility for future scheduler changes through a Maestro plugin. However, the MuMMI approach constitutes a resource allocator rather than an optimization tool. CBTune [30] uses reinforcement learning and utilizes a deep deterministic policy gradient method to find the optimal database instance configurations in high-dimensional continuous spaces adopting a try-and-error strategy to learn knob settings. While all these approaches primarily emphasize Big Data and data management analytics, RATS sets itself apart by specifically catering to the more complex case of modeling and predicting simulated biological processes. RATS effectively synthesizes multiple BO models and builds them side-by-side leveraging shared samples. It also incorporates the RATS Solver, which allows life scientists to observe the entire lifespan of the study on the desired treatment methodology set SS, in advance. Kunjir and Babu [31] compare black box methods, including BO and Deep Distributed Policy Gradient, against a proposed white-box algorithm to determine close-to-optimal tuning for memory-based analytics. Such an approach is orthogonal to RATS to make it also account for memory consumption.

Transfer Learning: In general, conventional ML models and algorithms [32] have shown substantial results in many real-world scenarios, including Natural Language Processing [33], Computer Vision [34], Recommendation Systems [35], Genomics and Bio-informatics [36]. However such models require sufficient labeled training data that are distributed similarly to the test data, while, developers have to train them from scratch on specific datasets via a time-consuming process. On top of that, traditional ML methods exhibit limitations in terms of generalization, i.e., they perform poorly in the face of new data, and as such a retraining procedure has to be performed. Given these limitations and the need to address this disparity, Transfer Learning [18] emerges as a highly compelling solution [19], which can be characterized as a fundamental concept in both artificial intelligence and ML domains. At its core, Transfer Learning relies on leveraging knowledge and insights gained from one task to enhance the performance of a related, but distinct, task or domain. Intuitively, in Transfer Learning, models are able to exploit previously acquired expertise in order to

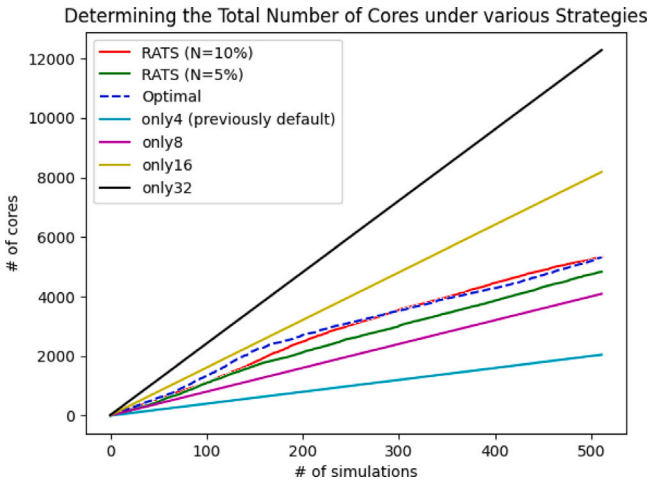


Fig. 6. Determining the total number of cores applying Part 1 and Part 2 of the RATS Solver, having trained the RATS Modeler using 5% and 10% of all possible simulation configurations $\{tnf_freq, tnf_dur, tnf_conc, k\}$. Naive assignment using only 4, 8, 16, 32 cores to each simulation are included. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

adapt to different, but related, tasks. As such, developers avoid the necessity of retraining the model from scratch. This process mirrors how humans learn—building upon prior experiences and adapting that knowledge to solve new problems. In essence, Transfer Learning acts like a connector between different domains, making it easier to transfer expertise among different tasks and ultimately improving the generalization and efficiency of machine learning models. The Transfer Learning approach we adopt in this work is inspired by Sequential Hierarchical GP (SHGP) which adopts a modular approach leading to a large decrease in the training complexity of GPs for Bayesian Optimization. The recent survey by Bai et al. [37] highlights all methodological aspects and technical details related to transfer learning methods for BO.

7. Experimental evaluation

RATS is implemented in Python also exploiting scikit-optimize (https://scikit-optimize.github.io/) for the RATS Modeler. The code of the PhysiBoSS 2.0 simulator is available open source (https://github.com/PhysiBoSS/PhysiBoSS). To test the performance of RATS, we collect real data by executing 2560 simulations on the MareNostrum 4 supercomputer, one of the most powerful supercomputers at the European continent level. These simulations are composed of $|SS| = 512$ TNF combinations with $[low, high, step]$ triplets (see Section 2):

`tnf_freq` ∈ [100, 450, 50]
`tnf_dur` ∈ [10, 45, 5]
`tnf_conc` ∈ [0.01, 0.045, 0.005]

and core configurations per simulation up to $max_core = 32$. We emphasize that we do not use all 2560 simulations to train the RATS Modeler. We utilize the complete set of simulations as the ground truth dataset, encompassing all possible simulations and core configurations. This dataset serves as a benchmark to evaluate and compare the performance of RATS against the ground truth system and simulation utility statistics.

RATS is configured according to the best practices we outline in Section 5 and we test two different versions of RATS operating with simulation budgets of $N = 5\%$ and $N = 10\%$, respectively. These are labeled as "RATS (N = 5%)" and "RATS (N = 10%)" in our graphs. We compare RATS against a series of competitors which we term "only4", "only8", "only16", "only32" which constitute manual simulation core configurations by having the life scientist executing each and every simulation with 4,8,16,32 cores, respectively. We begin our search with "only4" because this is what we would use as the default number of cores per simulation, before the development of RATS. There is also the "Optimal" competitor in our graphs which computes the optimal number of cores, simulation time and utility based on the entire, ground truth dataset. In our graphs we note the "previously default" property of "only4" just once in Fig. 6 and omit it from the rest of the graphs, for readability purposes. To speed up the execution of the RATS Solver, we limit the number of Knapsack solutions that are explored, each time $cap(t)$ cores are available, to 100K.

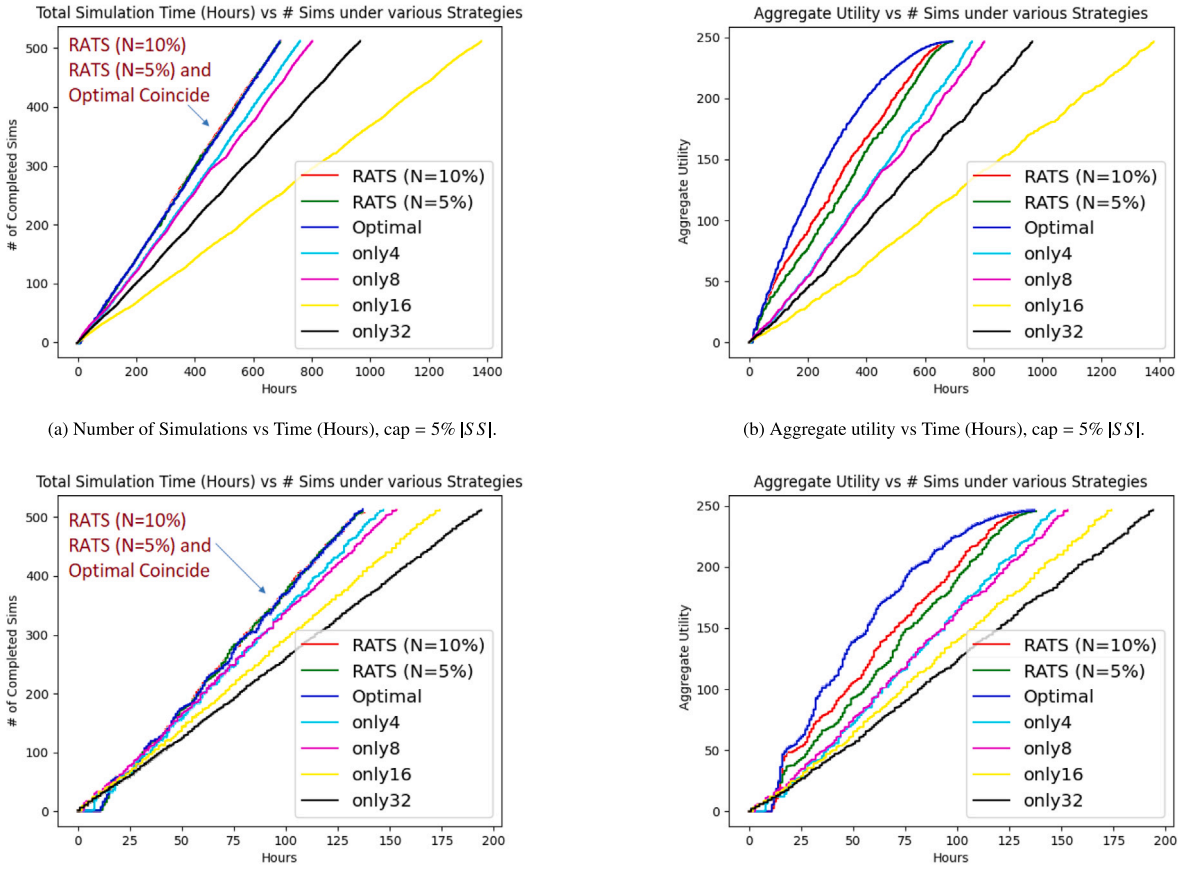
Table 2
Error in core hours under no capacity constraints (∓ stands for under-/over-estimation respectively).

#Simulations	RATS (N = 5%)	RATS (N = 10%)
100	−10.33%	−8.78%
200	−14.48%	−8.36%
300	−17.12%	2.26%
400	−13.08%	4.10%
500	−13.32%	2.53%

Main Findings: Before presenting our experimental evaluation in detail, we outline our main findings. When the entire simulation set can be submitted at once to the supercomputer (red-colored path in Fig. 4), "RATS (N = 10%)" approximates the optimal core hours within a 9% error in the worst case and below 3% in the best. When the RATS Modeler is trained with fewer samples ($N = 5\%$) this error increases reaching 10% to 18%. Under core capacity constraints (blue-colored path in Fig. 4) RATS coincides with the "Optimal" approach in all the cited cases: (i) when severe capacity constraints exist, which we model by setting $cap = 5\%|SS|$, RATS reduces the total simulation time by up to 4 days and the total core hours by up to 15% compared to the second best (manual configuration) competitor. RATS behavior coincides with what "Optimal" does, (ii) when capacity constraints are looser ($cap = 25\%|SS|$), RATS reduces the total simulation time by up to 10 h and the total core hours by up to 12% compared to the second best (manual configuration) competitor, again coinciding with the "Optimal". Furthermore, in both scenarios (i) and (ii), RATS holds an advantage by prioritizing simulations according to their utility, which is the value assigned by the life scientist within the range $[0, 1]$. Over the entire lifespan of the simulation, RATS achieves an aggregate utility that is up to 36% higher compared to the closest competitor, making it the superior choice. As a result, RATS not only accurately assigns near-optimal values to the required core hours for a treatment methodology study, but also substantially accelerates the completion of useful simulations. This gives life scientists the important option to conclude the study before the execution of the entire SS. Finally, our conducted experiments indicate that the Transfer Learning approach fostered by the RATS+ plug in effectively reduces the optimization effort without compromising accuracy, because it achieves equivalent performance to a RATS version that is trained from scratch with the same total simulation budget.

7.1. Total optimal number of core hours for the entire SS

In this first set of experiments we evaluate the performance of RATS when the entire set SS can be submitted for execution simultaneously (red-colored path in Fig. 4). Fig. 6 plots the number of cores devised by RATS vs. the number of simulations one may submit to the supercomputer. RATS is tested under two different configurations: one that spends a N = 5% budget on training the RATS Modeler and a second one with N set to 10% (green and red lines in the figure, respectively). RATS is compared with manual competitors, as we discussed above, and the blue-colored line corresponds to the "Optimal" total number of cores across various numbers of simulations. As illustrated in Fig. 6, "RATS (N = 10%)" almost perfectly fits on the "Optimal" plot line,



(a) Number of Simulations vs Time (Hours), cap = 5% $|SS|$.

(b) Aggregate utility vs Time (Hours), cap = 5% $|SS|$.

(c) Number of Simulations vs Time (Hours), cap = 25% $|SS|$.

(d) Aggregate Utility vs Time (Hours), cap = 25% $|SS|$.

Fig. 7. RATS solver performance in simulation time and aggregate utility. Comparison against various competitor strategies under various constraint and sample simulation budget N configurations. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

while "RATS (N = 5%)" emerges as the second-best alternative, slightly under-estimating the necessary number of cores. The best manual configuration approach is "only 8" which often under-utilizes more than 500 and up to 700 cores. All other manual approaches are impractical since they excessively over-utilize thousands of cores (for "only 16" and "only 32") or severely under-utilize resources (for "only 4").

Now, following the process in the red-colored path of Fig. 4, we use the number of cores devised by "RATS (N = 10%)" or "RATS (N = 5%)" to derive an accurate estimation about the total core hours required for SS. As can be deduced by Table 2, "RATS (N = 10%)" approximates the optimal core hours within a 3% to 9% error in the worst case. When the RATS Modeler is trained with fewer samples ($N = 5\%$) this error increases reaching 10% to 18%. An interesting observation arises from the scenario where the simulation set can be submitted at once. In such cases, it becomes advantageous for the life scientist to dedicate additional effort to train the RATS Modeler. This upfront investment in training pays off as evidenced by the performance of "RATS (N = 10%)", which achieves highly accurate optimization results. As we show in the next plots, if capacity constraints do not allow to submit $|SS|$ at once, "RATS (N = 5%)" may be preferable instead.

7.2. RATS under capacity constraints

When the entire set of SS cannot be submitted at once, the RATS solver follows the blue-colored path in Fig. 4. In such cases, simulations are retrieved from the utility-based priority queue and the Knapsack problem is solved at any given time t so that the number of simulations that run at t satisfies the constraint $cap \geq cap(t)$. The process concludes when the priority queue becomes empty, unless the life scientist chooses to halt the Solver before completion for reasons we explain below. It is important to recall that the RATS Solver itself does not perform the actual execution of simulations. Instead, it relies on the estimations provided by the trained RATS Modeler to predict simulation times and utility metrics. The manual approaches in this experiment simply execute simulations without prioritizing them.

Fig. 7 plots the number of completed simulations and the aggregate utility as more simulations are completed vs. the total simulation time in hours. Each row of plots should be examined side by side since they offer complementary information that enhances understanding. In the first row, Fig. 7(a) illustrates the progression of completed simulations over time, while Fig. 7(b) displays the aggregate utility of the simulations that have been completed up to each specific timestamp. It provides a visualization of how the aggregate utility evolves over time as more simulations are executed and completed. Both figures are generated under the scenario of severe core capacity constraints, modeled by setting $cap = 5\%|SS|$. Figs. 7(c) and 7(d) repeat this experiment, but this time under loose core capacity constraints modeled by setting $cap = 25\%|SS|$. By examining each row of the plot, one can deduce not only the speed at which a specific number of simulations is completed but also the level of usefulness in terms of their effectiveness in killing cancer cells.

Throughout the entire set of experiments depicted in Fig. 7, both the "RATS (N = 10%)" and "RATS (N = 5%)" lines align perfectly with the "Optimal" one. The second best competitor is the "only4" manual configuration approach, which practically means that it may be more preferable to execute more simulations with fewer cores under given capacity constraints, than few simulations with more cores attributed to each. Still, both versions of RATS in Fig. 7(a), reduce the total

simulation time by up to 4 days and therefore the total core hours (as the total simulation time across the horizontal axis multiplied by cap) by 15% compared to "only 4" for $cap = 5\%|SS|$. This is because RATS, at any given time across the horizontal time axis, elects for execution the set of simulations that minimize the maximum execution time by solving the Knapsack problem at time t. The simulations that are completed at any given time, based on RATS's prioritization, exhibit significantly higher aggregate utility as illustrated in Fig. 7(b). Similarly, under looser capacity constraints, $cap = 25\%|SS|$, RATS reduces the total simulation time by up to 10 h and the total core hours by up to 12% compared to "only4" also providing higher aggregate utility up to 36%. Therefore, based on the results of Figs. 7(b) and 7(d), the life scientists may decide not to execute the entire set SS, but restrict themselves to the top-K most useful simulations. Based on the graphs of Fig. 7, RATS will dictate the nearly-optimal number of core hours in every such case. When capacity constraints are in place, both "RATS (N = 5%)" and "RATS (N = 10%)" demonstrate comparable performance in terms of total simulation time. As a result, if the life scientist is willing to accept a slightly lower aggregate utility, we may utilize fewer samples for training the RATS Modeler.

7.3. RATS comparative analysis

In this section we compare RATS against alternative approaches, adapted for the tumor simulation setup we operate on. In particular, as mentioned in Section 6, there is no related work that combines the prediction power of the RATS Modeler with the targeted, utility and system resource-oriented operation of the RATS Solver. Therefore, to provide comparative results, we split our analysis into two parts. In the first part, we compare the RATS Modeler against Reinforcement Learning (RL) approaches, inspired by CBTune [30]. In the second part, we compare the RATS Solver with an adapted version of the Flux scheduler [38] used in MuMMI [29].

7.3.1. RATS modeler vs. reinforcement learning alternatives

We begin by comparing the prediction accuracy of the Bayesian Optimization-based RATS Modeler, with alternative modelers that utilize other Machine Learning models. In particular, we utilize Reinforcement Learning alternatives, namely the white-box, non-linear Tabular Q-Learning and the black-box Deep Q-Network (DQN) approaches that have already been used in the literature for general purpose cloud resource management [30]. Our analysis focuses on predicting the simulation time. That is, for any given simulation described by $\{tnf_freq, tnf_dur, tnf_conc\}$ being executed with a specified number of cores (k), we derive predictions for $ST_i(k)$ for varying number of training samples between 1 and 256. That is, up to 256 episodes for Tabular Q-Learning, DQN and up to 256 micro-benchmarks for RATS. Note, though, that the results are similar for $THP_i(k)$, as well.

For the RL approaches please refer to Sutton et al. [39] for a comprehensive study of the involved hyperparameters discussed hereafter. In both the RL approaches a factor of 0.9 is used for future reward discounting and an exploration rate of 0.1 controls the balance between exploration and exploitation. A 10% exploration rate is a reasonable trade-off between exploration and exploitation in most RL tasks. It allows the training process to explore new actions periodically without compromising too much on exploiting the best-known actions. For Tabular Q-Learning, we directly fill in the visited Q-Table cells with the observed simulation times and we interpolate the rest, unvisited cells, using a KNN Regressor where $K = 3$. In this case, given the limited number of episodes in our setup, the Q-Table is expected to be sparse, because many state–action pairs are unvisited. Choosing $K = 3$ provides enough values to interpolate unvisited Q-Table cell values without relying on too many distant, possibly irrelevant cells. The learning rate is set to 0.01.

For the DQN approach, the neural network consists of three fully connected layers: an input layer with 4 units, one for each of $\{tnf_freq, tnf_dur, tnf_conc, k\}$, two hidden layers with 64 units each, and an output layer corresponding to the number of possible actions. Similar neural network architectures have been empirically shown to work well in many RL tasks, providing a good balance between model complexity and training efficiency [40]. Besides, note that the more complex is the neural network, the larger number of episodes (training samples) it needs to be effectively trained. The batch size is set to a typical value of 64 and the Q-Learning updates in DQN use a learning rate of 0.001. A replay buffer is used to store past experiences (state, action, reward, next state) during the training process. The replay buffer samples a batch of experiences (randomly) from past interactions, and the model learns from these experiences, improving the stability and efficiency of learning. Finally, the Adam optimizer is used with a learning rate of 0.01.

Recall, that a key argument for fostering BO, as mentioned in Section 2.5, is that it is the most preferable ML approach when we need to build models of the higher possible accuracy with as few training samples as possible, given resource, time and human effort constraints. Fig. 8 stresses this observation by quantifying the accuracy of the involved models under various measures (L_1 Mean, L_1 Median, Pearson's Correlation, R^2 Score) varying the number of training samples (termed episodes for the RL approaches and micro-benchmarks for BO). The lower L_1 Mean, L_1 Median are, the better the model performance. On the contrary, the closer to 1 the Pearson's Correlation and R^2 values, the more able is the examined model to, at least, capture the trends in the provided predictions. In other words, Pearson's Correlation and R^2 quantify the ability of the model to capture trends and relative distances, i.e., distinguish which $ST_i(k)$ is expectedly lower than another, even though $ST_i(k)$ value predictions may not be accurate in terms of absolute values.

As shown in the figure, the white-box, non-linear Tabular Q-Learning approach does not improve its accuracy across any measure. All lines in the respective plots remain stable, which means that the model does not learn anything important for the restricted number of episodes that are feasible in our tumor simulation setup. This is natural because with so few episodes, only 256 out of 2560 Q-Table cells are visited, while the rest are interpolated by sparse measurements. Therefore, it is evident that Tabular Q-Learning is not the right way to go when the micro-benchmark budget should be kept to a minimum due to resource, time and human effort constraints. On the other hand, the DQN approach, plotted in Fig. 8, does improve its performance with more episodes, but the restricted number of such episodes do not appear enough for the model to converge.

The BO approach incorporated in RATS shows remarkable prediction accuracy improvements with each and every new micro-benchmark. This is happens from the very beginning of the simulation process due to the right kernel configurations, as described in Section 5. In Fig. 8, L_1 Mean, L_1 Median values progressively approach zero. At the same, Pearson's Correlation and R^2 values approach 1. We can therefore conclude that the use of a proper kernel, acquisition function in RATS along with the rest of the best practices cited in Section 5, not only enable RATS to effectively capture the trends and relative distances among different $\{tnf_freq, tnf_dur, tnf_conc, k\}$ options, but also accurately predict $ST_i(k)$s in terms of absolute values.

7.3.2. RATS solver vs. flux-like scheduling

We then proceed to comparing RATS with a scheduling approach similar to Flux [38] used in MuMMI [29]. Here, instead of concentrating on the accuracy of the RATS Modeler, we focus on comparing the effectiveness of the RATS Solver to use the RATS Modeler, prioritize simulations based on their utility UTL_i and allocate cores so that the total simulation time is minimized. Let us elaborate on how we implement Flux-like scheduling to match the setup RATS operates on, so that a comparison is possible:

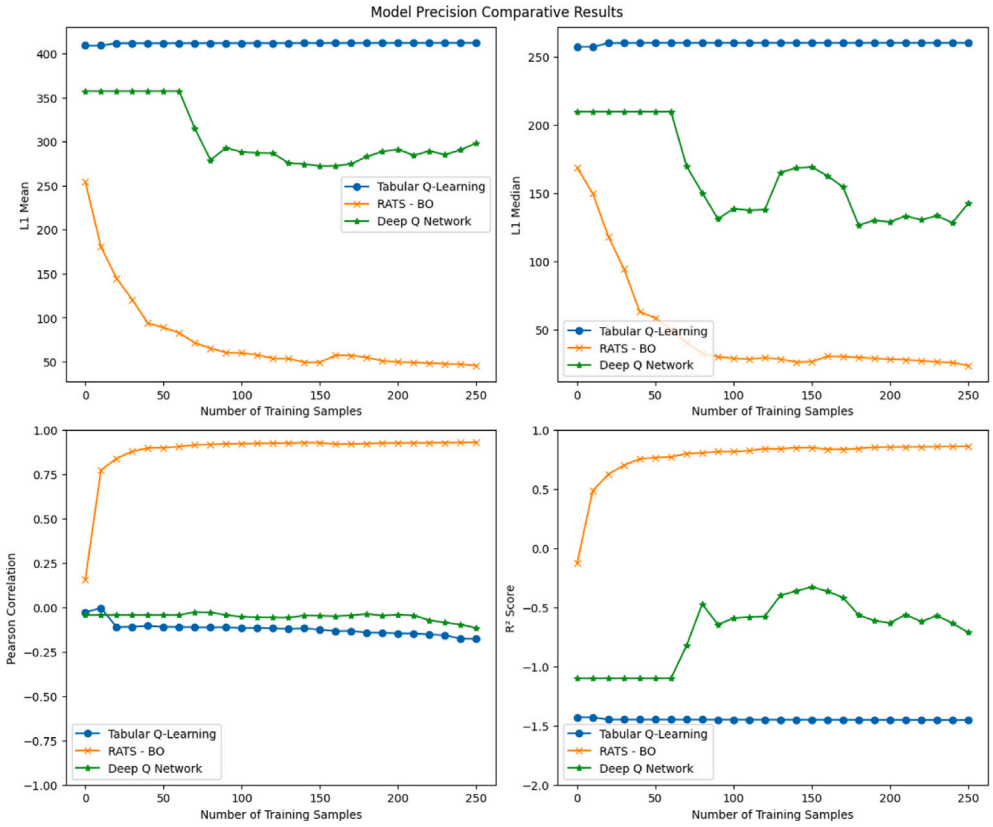


Fig. 8. Comparison of RATS Modeler vs. Tabular Q-Learning and DQN approaches on simulation time prediction accuracy.

Table 3
Comparison of RATS 5% and flux-like simulation results under 5% $|SS|$ capacity.

Capacity 5% $\|SS\|$				
Time (Hours)	RATS 5% completed simulations	RATS 5% aggregate utility	Flux-like completed simulations	Flux-like aggregate utility
100.00	71	46.77	59	29.79
200.00	153	82.56	118	57.33
300.00	235	123.98	180	86.14
400.00	320	167.15	239	114.95
500.00	398	204.34	298	144.29
600.00	479	240.63	357	173.29
640.00	512	246.41	–	–
700.00	–	–	416	203.09
800.00	–	–	473	230.75
873.58	–	–	512	246.41

Table 4
Comparison of RATS 5% and flux-like simulation results under 25% $|SS|$ capacity.

Capacity 25% $\|SS\|$				
Time (Hours)	RATS 5% completed simulations	RATS 5% aggregate utility	Flux-like completed simulations	Flux-like aggregate utility
25.00	64	42.74	71	34.73
50.00	170	92.95	144	69.65
75.00	278	147.24	215	104.02
100.00	369	189.35	286	138.76
125.00	466	236.53	362	176.62
142.00	512	246.41	–	–
150.00	–	–	434	211.74
175.00	–	–	500	242.03
179.62	–	–	512	246.41

- Job Queue and Prioritization: Flux uses a set of policies to prioritize jobs, but it allows users to manually create custom policies for such prioritization. In our implementation, we use a job queue which prioritizes jobs based on their utility. For that, we give the Flux-like approach all the ground truth utility values to allow it for job prioritization. On the other hand, the RATS Solver still uses the RATS Modeler to predict utilities. This a priori provided ground truth utility knowledge forms a favorable scenario for the Flux-like approach.
- Dynamic Core Assignment: In Flux, the scheduler dynamically assigns available cores to jobs based on available resources, user-defined policies and historical data from previous execution of

such jobs. In our setup, we do not assume that historical data exists.[3] Additionally, we aim at totally alleviating the life scientist from technical considerations such as manually defining scheduling policies. Therefore, the Flux-like approach implemented here does retain dynamic core assignment, by assigning the maximum [4,8,16,32] cores to a simulation depending on how many cores are freed from completed simulations, during the execution of the medical study.

- Backfilling: Flux uses a backfilling rationale for smaller jobs. In our Flux-like scheduling implementations, we also incorporate backfilling to fill gaps in resource usage with smaller jobs, ensuring better core utilization. That is, if a job cannot find at least 4 cores available (the best manual configuration strategy according to Fig. 7), the job is placed in a backfill queue. The backfill queue is checked for whether the currently available cores meet the at least 4 core requirement, every time cores are freed from completed simulations.

Flux also attempts to exploit job dependencies in its scheduling decisions. It also applies workflow-level scheduling for groups of dependent tasks rather than just independent jobs. These do not apply in our setting, because individual simulations described by the triplets $\{tnf_freq, tnf_dur, tnf_conc\}$, do not affect one another or, more precisely, if one $\{tnf_freq, tnf_dur, tnf_conc\}$ is executed with a specific number of cores k, all other $\{tnf_freq, tnf_dur, tnf_conc, k\}$ possible configurations are omitted.

Having clarified the above, Tables 3 and 4 provide comparative results between RATS 5% and the Flux-like approach for the two scenarios with resource capacities of 5% $|SS|$ and 25%$|SS|$, respectively. In the more resource constrained setup of Table 3, we observe that RATS executes the entire set $|SS| = 512$ of simulations reducing the overall simulation time (leftmost Time column in the table) by more than 230 h. In addition, throughout the simulation process, monitored in intervals of 100 h, RATS completes up to 122 more simulations, simultaneously achieving up to 67 units higher aggregate utility. This shows that RATS both reduces the overall simulation time and allows the life scientist to stop the medical study earlier having the most promising simulations, of higher utility, already completed.

Similarly, in Table 4, with loosened capacity constraints, RATS executes the entire set $|SS| = 512$ of simulations, reducing the overall simulation time by more than 37 h. Despite the fact that in the first 25 h of simulation execution, Flux-like achieves a higher number of completed simulations, these simulations give a lower aggregate utility compared to RATS due to the backfilling property of Flux. In other words, the otherwise beneficial backfill strategy that Flux uses, in this setup does not seem to contribute so that not only simulations are executed faster, but also the most promising ones, are truly prioritized higher. Beyond the first 25 simulation hours, the simulation execution progress being monitored in intervals of 25 h, RATS completes up to 104 more simulations, simultaneously achieving up to 60 units higher aggregate utility.

Based on the above facts, we can conclude that the multiple choice Knapsack modeling approach followed in the RATS Solver (Section 2.3), paired with the predictive power of the RATS Modeler, better suits the problem at hand. Commonly used RL-based techniques in cloud resource allocation settings, cannot converge with limited number of executed simulations in the RATS setup. Moreover, plausible strategies such as backfilling used by the Flux-like scheduler, which are beneficial under pure resource utilization criteria, seem to backfire in prioritizing expectedly useful simulations reducing the aggregate utility during the medical study. This holds despite the fact that the Flux-like scheduling approach in our experiments was provided all ground truth utility values for simulation prioritization.

[3] We only assume previous Bayesian Optimization models may exist when we develop the transfer learning approach of RATS+.

7.4. RATS+ effectiveness evaluation

In this section we study the effectiveness of RATS+ and its ability to exploit prior knowledge gained from optimizing treatment methodologies from a relevant domain $\mathcal{D}_S$ to reduce the optimization effort in a new domain $\mathcal{D}_T$, where $\mathcal{D}_S \neq \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T = \mathcal{T}$. In particular, we define $\mathcal{D}_S$ as a subset of set SS, denoted as SS', with the following $[low, high, step]$ triplets:
`tnf_freq` $\in$ [100, 450, 50]
`tnf_dur` $\in$ [10, 45, 5]
`tnf_conc` $\in$ [0.01, 0.03, 0.005] and core configurations per simulation up to $max_core = 32$. Similarly, we define $\mathcal{D}_\mathcal{T}$ as the entire set SS (see Section 7).

7.4.1. RATS+ performance on estimating the total optimal number of core hours

Here, we study the performance of our approach on $\mathcal{D}_\mathcal{T}$ where RATS+ is able to exploit prior knowledge gained from domain $\mathcal{D}_S$. Our goal, i.e., the task $\mathcal{T}$ is identical to the goal in Section 7.1, but RATS+ performs the described two-step process, where in the first step regressors are built on $\mathcal{D}_S$ and at the second step they are transferred to $\mathcal{D}_\mathcal{T}$ with the rest of the BO-based training following up. Thus, we developed six different configurations for RATS and RATS+: in two configurations RATS+ is used, while in the remaining four configurations RATS is trained from scratch in $\mathcal{D}_\mathcal{T}$, i.e., without exploiting Transfer Learning. In more detail, "RATS+ (N = (3%+2%))" operates with a simulation budget of $N = 5\%$, which is distributed among $\mathcal{D}_S$ and $\mathcal{D}_\mathcal{T}$ accordingly.[4] As such, first RATS+ gains expertise in $\mathcal{D}_S$ and then transfers this knowledge in domain $\mathcal{D}_\mathcal{T}$. Similarly, "RATS+ (N = (6%+4%))" operates with a simulation budget of $N = 10\%$, which is distributed among its two optimizers for $\mathcal{D}_S$ and $\mathcal{D}_\mathcal{T}$ accordingly.[5] The "RATS (N = 10%)", "RATS (N = 5%)", "RATS (N = 4%)" and "RATS (N = 2%)" configurations do not perform Transfer Learning. We highlight that: (i) ("RATS (N = 10%)", "RATS+ (N = (6%+4%))") and ("RATS (N = 5%)", and "RATS+ (N = (3%+2%))") pairs hold the same total simulation budget respectively; and (ii) ("RATS+ (N = (6%+4%))", "RATS (N = 4%)") and ("RATS+ (N = (3%+2%))", "RATS (N = 2%)") have the same simulation budget only on $\mathcal{D}_\mathcal{T}$. The idea behind "RATS (N = 10%)" and "RATS (N = 4%)" is to compare them against the performance of "RATS+ (N = (6%+4%))". This is because "RATS (N = 10%)" trains the regressors from scratch with an overall budget equivalent to "RATS+ (N = (6%+4%))", while "RATS (N = 4%)" uses a budget equivalent only to the Transfer Learning part of RATS+. Similarly for "RATS (N = 5%)", "RATS+ (N = (3%+2%))" and "RATS (N = 2%)".

Fig. 9 depicts our experimental results. Notably, "RATS+ (N = (6%+4%))" and "RATS (N = 10%)" achieve the best performance since they are able to converge to the optimal solution. Next, "RATS+ (N = (3%+2%))" and "RATS (N = 5%)" perform in a similar way achieving a near-optimal performance. As such, our results indicate that Transfer Learning in RATS+ approaches exhibits similar performance as the performance of "Optimal" and that of RATS. This means that it exploits Transfer Learning without compromising the accuracy of the built regressors. Therefore, RATS+ is able to boost the performance of our framework not only saving many hours in terms of simulation time but also reducing the optimization effort itself by avoiding a training from scratch procedure. Furthermore, we note that all of the examined

[4] The optimizer for $\mathcal{D}_S$ takes ~60% of total simulation budget, while the optimizer for $\mathcal{D}_\mathcal{T}$ holds the remaining budget.

[5] Note that neither scikit-optimize nor any other known to us library implemented in python enables the direct transfer of the base estimator of one optimizer, i.e., a fitted Gaussian Process regressor, to another optimizer. Thus, in our implementation, we manually define the first calls of the second optimizer (in $\mathcal{D}_\mathcal{T}$) to be identical to the calls that were performed by the first optimizer (in $\mathcal{D}_S$) in order to achieve the notion of Transfer Learning

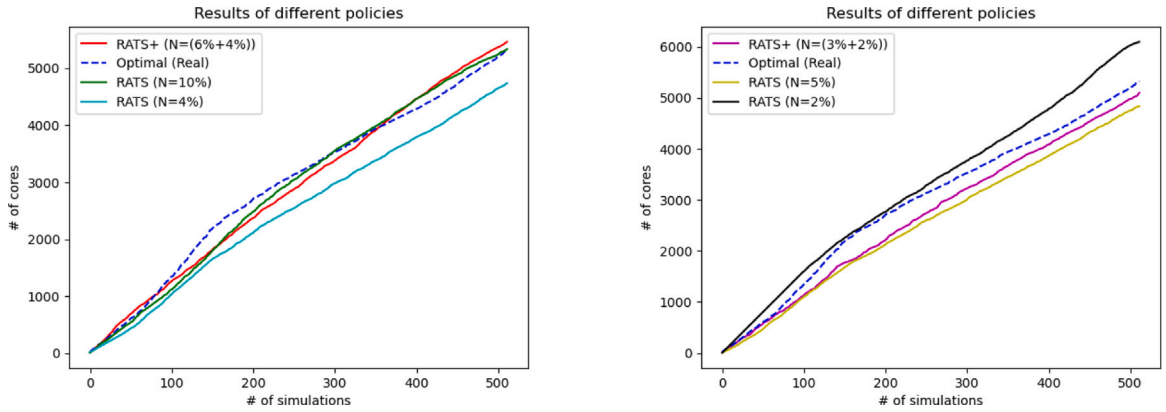


Fig. 9. Effectiveness of RATS+ in prescribing the optimal number of cores, measured based on its proximity to "Optimal", RATS trained from scratch and RATS trained under a reduced simulation budget. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

configurations of RATS perform better than the manual configurations presented earlier demonstrating the effectiveness of our approach (we do not include the manual configurations here for readability purposes). Last but not least, we stress the correlation between simulation budget and performance, i.e., as we increase the simulation budget our approach is able to achieve more near to optimal performance.

7.4.2. RATS+ under capacity constraints

Following the experiment presented in Section 7.2 we study the performance of our approach on $D_{\mathcal{T}}$ when RATS+ exploits prior knowledge gained from domain D_S in the constrained problem setup. Our goal, i.e., the task $\mathcal{T}$ is identical to the goal in Section 7.2. For this series of experiments we developed six configurations which are identical with those of Section 7.4.1 in order to evaluate our method, this time under capacity constraints.

Figs. 10 and 11 present our experimental results in scenarios that our setting is restricted by a core capacity constraint of $cap = 5\%|SS|$ and $cap = 25\%|SS|$. We observe that the RATS approach as a whole (all versions of RATS and RATS+), for both scenarios involving capacity constraints, is able to reduce the completion time of simulations, even for cases with a limited simulation budget, since each configuration of RATS lies near the "Optimal" approach (see Figs. 10(c), 11(c), 10(d) and 11(d)). Note that RATS and RATS+ configurations that are presented in Figs. 10(c) and 11(c) are perfectly aligned with the "Optimal" line compared to the corresponding configurations that are illustrated in Figs. 10(d) and 11(d) which exhibit a minor discrepancy in comparison to the "Optimal" line. This is natural since the former ones possess a larger simulation budget than the latter ones and as such achieve higher performance. Consequently, it is evident that our proposed approach, to a significant extent, results in substantial time savings in scenarios where core capacity constraints have to be satisfied even for the RATS+ plug in.

The distinction between the RATS+ and RATS configurations come from studying their effect on the measured aggregate utility. Recall that in our domain of interest, where the sequence of simulation executions is crucial for cutting down to market of potential therapies, it becomes apparent that the simulation budget that will be used in a configuration plays a key role. Specifically, as our results indicate, (see Figs. 10(a), 11(a), 10(b) and 11(b)) "RATS (N = (6%+4%))" exhibits equivalent aggregate utility to "RATS (N = 10%)", while "RATS (N = (3%+2%))" exhibits equivalent aggregate utility to "RATS (N = 5%)". This validates our claim that the Transfer Learning approach in RATS+ is as effective as the much more time and resource consuming training from scratch process. Therefore, RATS+ reduces the optimization effort retaining the ability to prioritize "the most promising" simulations in the very first iterations. Finally, "RATS (N = 4%)" and "RATS (N = 2%)" perform up to 20% worse compared to "RATS (N = (6%+4%))" and "RATS (N = (3%+2%))" respectively validating the fact that avoiding the training from scratch by using considerably fewer iterations will compromise the accuracy of the utility regressor, in contrast with RATS+. Again, we can observe that all of the examined configurations of RATS perform better than the manual configurations which we omit for readability purposes.

8. Conclusions & future work

The RATS approach takes away from life scientists the burden of determining the total core hours for in-silico studies on tumor treatment methodologies and automates the system configurations for such trials. This fact, combined with the prioritization of simulations based on their expected utility, helps in more rapidly distinguishing effective drug combinations, cutting time to market for new cancer therapies. Moreover, we introduced an extension of RATS, namely RATS+, which is able to exploit knowledge gained from previous medical studies to more rapidly optimize new ones, under different drug application configurations. RATS+ builds up a knowledge base of previous optimizations to further reduce time-to-market for effective treatments by reducing the optimization effort in any future tumor study. Our future work is on extending RATS to more complex treatment methodologies and on integrating it with relevant open source frameworks [21].

CRediT authorship contribution statement

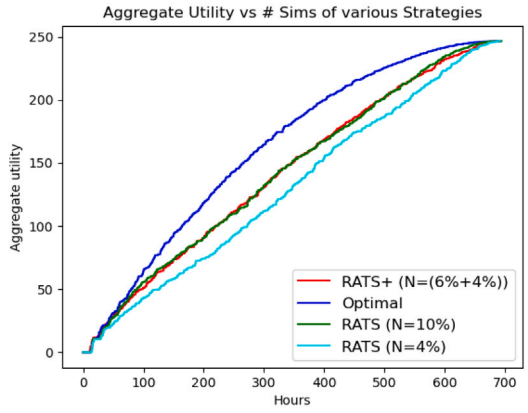
Errikos Streviniotis: Conceptualization, Data curation, Formal analysis, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Nikos Giatrakos:** Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing. **Yannis Kotidis:** Conceptualization, Formal analysis, Supervision. **Thaleia Ntiniakou:** Data curation, Investigation. **Miguel Ponce de Leon:** Data curation, Formal analysis, Supervision.

Declaration of competing interest

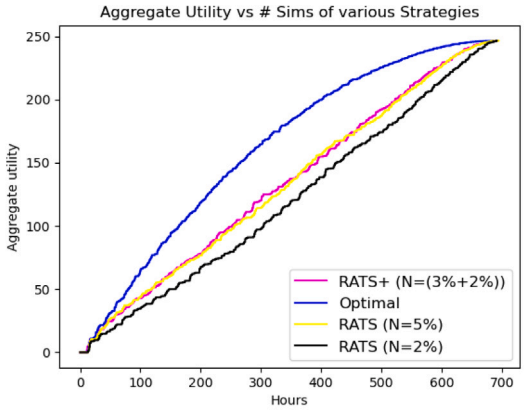
The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: All Authors including the corresponding author. Acks on the manuscript reports financial support was provided by Horizon Europe.

Acknowledgments

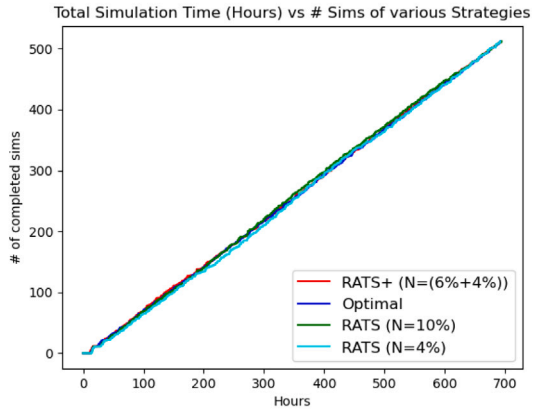
This work is supported by the EU project EVENFLOW under Horizon Europe agreement No. 101070430 and by the EU project CREXDATA under Horizon Europe agreement No. 101092749.



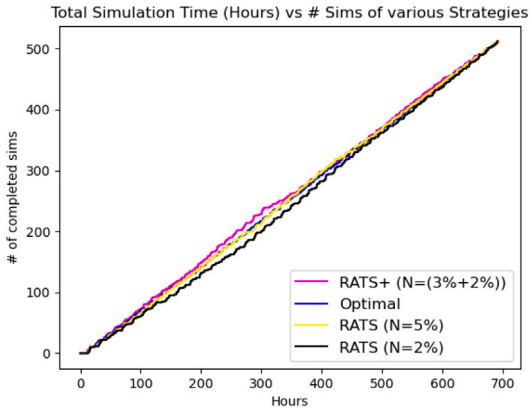
(a) Aggregate utility vs Time (Hours), cap = 5% $|SS|$.



(b) Aggregate utility vs Time (Hours), cap = 5% $|SS|$.



(c) Number of Simulations vs Time (Hours), cap = 5% $|SS|$.



(d) Number of Simulations vs Time (Hours), cap = 5% $|SS|$.

Fig. 10. Effectiveness of RATS+, under core capacity constraints of $cap = 5\%|SS|$, in terms of Simulation Time and Aggregate Utility, based on its proximity to "Optimal", RATS trained from scratch and RATS trained under a reduced simulation budget. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table A.5
Correlation of $\{tnf_freq, tnf_dur, tnf_conc\}$ with normalized throughput.

Feature	Pearson correlation	Spearman correlation
tnf_freq	0.135903	0.139594
tnf_dur	−0.142806	−0.151502
tnf_conc	0.017116	0.005248

Table A.6
ElasticNet regression Coef. & Variance Inflation Factor (VIF).

Feature	ElasticNet coefficient	VIF
tnf_freq	0.000125	1.000014
tnf_dur	−0.001270	1.000220
tnf_conc	0.000000	0.972654

Table A.7
Permutation feature importance of Gaussian process regressor.

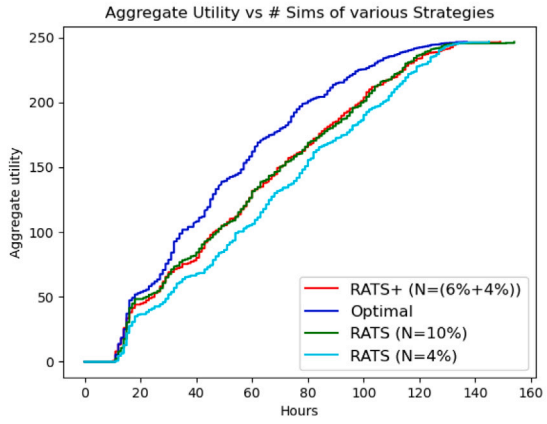
Feature	PFI
tnf_freq	1.146855
tnf_dur	1.053872
tnf_conc	1.057870

Appendix. Input parameter importance analysis

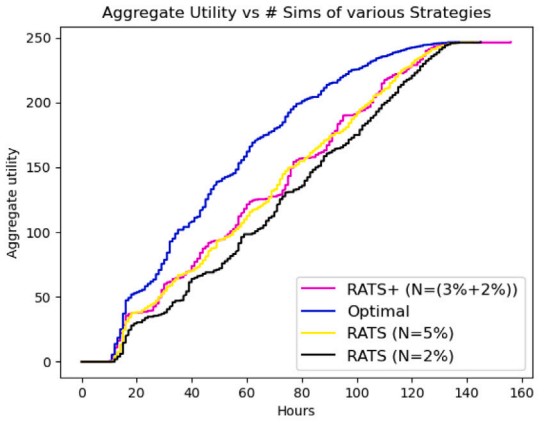
To better motivate our choice of modeling the problem at hand using a black-box, Bayesian Optimization approach with Gaussian Process Regressors as the surrogate model, we here perform an analysis on input feature importance. We indicatively concentrate our analysis on throughput and, in particular, we normalize throughput values for comparability purposes.

In Fig. A.12 we present three scatter plots, one for each of the $\{tnf_freq, tnf_dur, tnf_conc\}$ input parameters (horizontal axis) vs. the achieved throughput. All graphs show that the respective values of all these quantities exhibit high variability across a wide range and there does not seem to exist a clear relationship or trend between the input triplet and the achieved throughput. This is further validated by Table A.5 where we cite the Pearson's Correlation Coefficient and Spearman's Correlation Coefficient values for each feature with respect to throughput. It can be observed that the respective values are very close to zero, indicating no clear correlation.

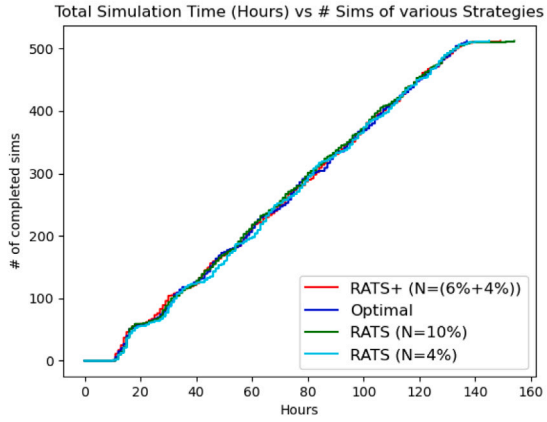
To make our case even stronger, we train an Elastic Net Regression model as a white-box approach to check if a linear relationship can be detected in a more interpretable context, compared to our black-box approach. We use 5-fold cross validation and an L_1 ratio of 0.5 for our training. Table A.6 shows that all Elastic Net coefficients are negligible (very close to zero). This suggests that a simple, white-box, linear model like Elastic Net is not capturing meaningful relationships in the data. On the other hand, Table A.6 also includes the Variance Inflation Factor (VIF) values on the Elastic Net model. The VIF values are all close to 1 indicating no colinearity. These VIF values, combined with the small Elastic Net coefficients, indicate that the features do not strongly influence the target variable (normalized throughput) in a linear setting. In turn, this proves that the relationship is a more complex, non-linear one.



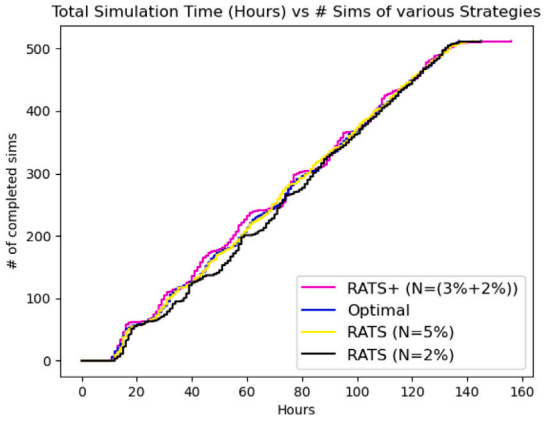
(a) Aggregate utility vs Time (Hours), cap = 25% $|SS|$.



(b) Aggregate utility vs Time (Hours), cap = 25% $|SS|$.

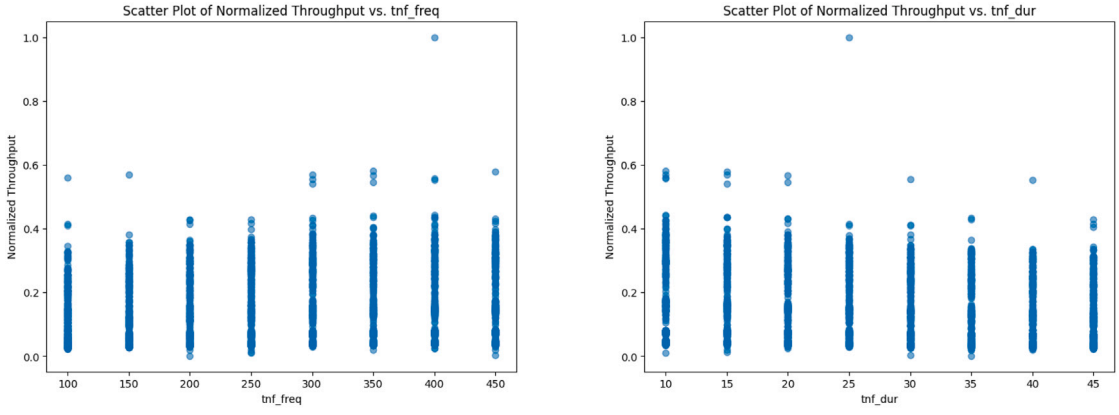


(c) Number of Simulations vs Time (Hours), cap = 25% $|SS|$.



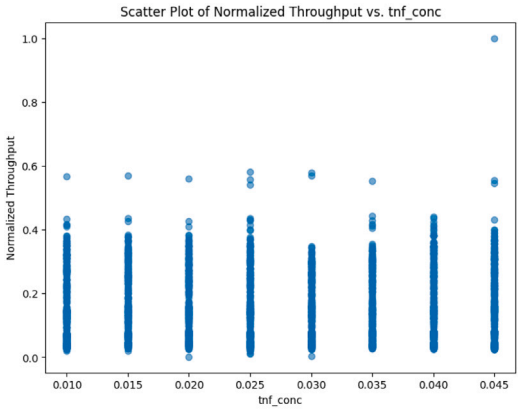
(d) Number of Simulations vs Time (Hours), cap = 25% $|SS|$.

Fig. 11. Effectiveness of RATS+, under core capacity constraints of $cap = 25\%|SS|$, in terms of simulation time and aggregate utility, based on its proximity to "Optimal", RATS trained from scratch and RATS trained under a reduced simulation budget. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



(a) Scatter Plot of TNF Frequency (tnf_freq)

(b) Scatter Plot of TNF Duration (tnf_dur)



(c) Scatter Plot of TNF Concentration (tnf_conc)

Fig. A.12. Scatter plots of normalized throughput vs. TNF parameters.

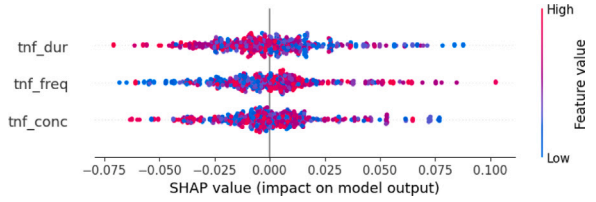


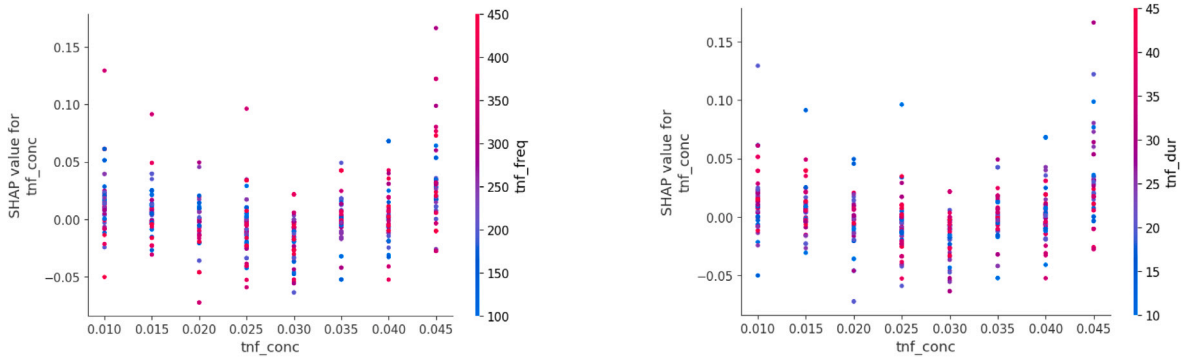
Fig. A.13. SHAP summary plot showing the impact of the features tnf_freq, tnf_dur, and tnf_conc on the model's output. A blue colored SHAP value for an individual sample indicates low feature value, while a red colored one indicates high feature value. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Recall, that in our experimental evaluation (Section 7.3), we also perform a comparative analysis between RATS and a Tabular Q-Learning approach, i.e., a white-box, non-linear model. Since RATS outperforms the Tabular Q-Learning alternative, we have already confirmed that white-box, non-linear models are also not able to capture the complex patterns in the data. Nonetheless, this does not mean that such patterns do not exist or that they cannot be learned. It means that our simulation execution data contains complex feature interactions, latent factors and non-obvious patterns that only a model of higher capacity can capture.

To validate this claim, we analyzed the discriminatory power of the input parameters by computing the Permutation Feature Importance (PFI) [41], as well as the Shapley values (SHAP) on our trained Gaussian Process Regressor model. To stress test the GPR, we use 128 simulations sampled uniformly at random. All other GPR hyperparameters remained the same used in our experimental evaluation (Section 7). This builds a less accurate model compared to having the acquisition function drive the GPR training process. Table A.7 includes the computed Permutation Feature Importance values for our Gaussian Process Regressor. Permutation Feature Importance is a model inspection technique that measures the contribution of each feature to a fitted model's performance. This technique involves randomly shuffling the values of a single feature and observing the resulting degradation of the model's score. Based on this, Table A.7 shows that all $\{tnf_freq, tnf_dur, tnf_conc\}$ features have comparable and high discriminatory value in the resulted normalized throughput. Therefore, none of the features stands out as significantly more important or irrelevant to the analysis. All features are equally and highly important.

Similar conclusions can be drawn upon examining Shapley values, which capture the marginal contribution of each TNF parameter to the model's output. Fig. A.13 shows the impact of each feature in the model output. The spread of points indicates the range of impact a feature can have, being an indicator of the discriminatory power of the feature. It can easily be observed that all features have similar discriminatory power. The color gradient shows that high values of tnf_freq (red) generally increase the prediction (positive SHAP values), while both high and low values of tnf_dur and tnf_conc evenly contribute based on the coloring in the prediction spectrum.

tnf_conc has more values concentrated around zero. This is expected because, by recalling that tnf_conc quantifies the drug dosage, very limited value ranges of tnf_conc correspond to impactful treatments. For instance, certain low or certain high tnf_conc values will either have no impact on the simulated tumor or kill every, even healthy, cell. This is further validated upon examining the SHAP dependency plots of tnf_conc with tnf_freq and tnf_dur. Fig. A.14 illustrates that in both cases the effect of tnf_conc is not highly dependent on tnf_freq



(a) SHAP values for 'tnf_conc' with 'tnf_freq' interaction

(b) SHAP values for 'tnf_conc' with 'tnf_dur' interaction

Fig. A.14. SHAP dependence plots for 'tnf_conc' with different interaction features ('tnf_freq' and 'tnf_dur').

and tnf_dur. This is due to the fact that, irrespectively of the frequency of the drug application (tnf_freq) and the duration the drug is applied (tnf_dur), if the drug dosage is too high or too low it will lead to either low throughput due to the large number of killed cells or to always high throughput, due to the unaffected number of living or proliferated cells.

In all, our analysis in this appendix shows that while some parameters can have slightly stronger influences on the outcome of the model with respect to the target, throughput, variable; the inherent complexity of the simulated biological processes require treating it as a black-box to more effectively learn these relationships.

Data availability

Data will be made available on request.

References

[1] M. Ponce-de Leon, A. Montagud, V. Noël, et al., Physiboss 2.0: a sustainable integration of stochastic boolean and agent-based modelling frameworks, Npj Syst. Biol. Appl. 9 (2023) 54, http://dx.doi.org/10.1038/s41540-023-00314-4.

[2] Y.A. Fouad, C. Aanei, Revisiting the hallmarks of cancer, Am J Cancer Res. 7 (5) (2017) 1016–1036.

[3] S.M. Shaffer, M.C. Dunagin, S.R. Torborg, et al., Rare cell variability and drug-induced reprogramming as a mode of cancer drug resistance, Nature 546 (7658) (2017) 431–435.

[4] N. Giatrakos, N. Katzouris, A. Deligiannakis, et al., Interactive extreme: Scale analytics towards battling cancer, IEEE Technol. Soc. Mag. 38 (2) (2019) 54–61.

[5] J. Li, Q. Yin, H. Wu, Chapter five - structural basis of signal transduction in the tnf receptor superfamily, in: 119 of Advances in Immunology, Academic Press, 2013, pp. 135–153, http://dx.doi.org/10.1016/B978-0-12-407707-2.00005-9.

[6] P. Bloomingdale, V.A. Nguyên, J. Niu, D.E. Mager, Boolean network modeling in systems pharmacology, J. Pharmacokinet. Pharmacodyn. 45 (2018) 159–180.

[7] E. Streviniotis, N. Giatrakos, Y. Kotidis, T. Ntiniakou, M. Ponce de Leon, Optimizing resource allocation for tumor simulations over HPC infrastructures, in: 10th IEEE International Conference on Data Science and Advanced Analytics, DSAA, Thessaloniki, Greece, IEEE, 2023, pp. 1–10, http://dx.doi.org/10.1109/DSAA60987.2023.10302484.

[8] Openmpi software documentation, 2025, https://www.open-mpi.org/doc/v4.0/man3/MPI_Allreduce.3.php. (Accessed 02 March 2025).

[9] A. Ghaffarizadeh, R. Heiland, S.H. Friedman, S.M. Mumenthaler, P. Macklin, Physicell: An open source physics-based cell simulator for 3-d multicellular systems, PLoS Comput. Biol. 14 (2) (2018) 1–31, http://dx.doi.org/10.1371/journal.pcbi.1005991.

[10] G. Stoll, B. Caron, E. Viara, A. Dugourd, A. Zinovyev, A. Naldi, G. Kroemer, E. Barillot, L. Calzone, S. MaBoS, 2.0: an environment for stochastic boolean modeling, Bioinform. 33 (14) (2017) 2226–2228, http://dx.doi.org/10.1093/bioinformatics/btx123, arXiv:https://academic.oup.com/bioinformatics/article-pdf/33/14/2226/50314917/bioinformatics_33_14_2226.pdf.

[11] A. Ghaffarizadeh, S.H. Friedman, P. Macklin, BioFVM:.an. efficient, Parallelized diffusive transport solver for 3-D biological simulations, Bioinform. 32 (8) (2015) 1256–1258, http://dx.doi.org/10.1093/bioinformatics/btv730, arXiv:https://academic.oup.com/bioinformatics/article-pdf/32/8/1256/49018460/bioinformatics_32_8_1256.pdf.

[12] Regular access call – technical guidelines - prace-ri.eu, 2023, URL https://prace-ri.eu/wp-content/uploads/Technical_Guidelines-Regular_Access.pdf. (Accessed 12 May 2023).

[13] K. DudziÅski, S. Walukiewicz, Exact methods for the knapsack problem and its generalizations, European J. Oper. Res. 28 (1) (1987) 3–21, http://dx.doi.org/10.1016/0377-2217(87)90165-2.

[14] D.R. Jones, M. Schonlau, W.J. Welch, Efficient global optimization of expensive black-box functions, J. Global Optim. 13 (4) (1998) 455–492.

[15] E. Brochu, V.M. Cora, N. de Freitas, A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning, 2010, arXiv:1012.2599.

[16] C.E. Rasmussen, C.K.I. Williams, Gaussian processes for machine learning, in: Adaptive Computation and Machine Learning Series, MIT Press, 2005.

[17] J. Wang, An intuitive tutorial to Gaussian processes regression, 2020, arXiv:2009.10862. CoRR abs/2009.10862, URL https://arxiv.org/abs/2009.10862.

[18] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 1345–1359, http://dx.doi.org/10.1109/TKDE.2009.191.

[19] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, Q. He, A comprehensive survey on transfer learning, 2020, arXiv:1911.02685.

[20] N. Giatrakos, E. Kougioumtzi, A. Kontaxakis, A. Deligiannakis, Y. Kotidis, Easyflinkcep: Big event data analytics for everyone, in: CIKM, 2021.

[21] C. Akasiadis, M.P. de Leon, A. Montagud, E. Michelioudakis, A. Atsidakou, E. Alevizos, A. Artikis, A. Valencia, G. Paliouras, Parallel model exploration for tumor treatment simulations, Comput. Intell. 38 (4) (2022) 1379–1401.

[22] B.K. Lind, P. Mavroidis, S. Hyödynmaa, C. Kappas, Optimization of the dose level for a given treatment plan to maximize the complication-free tumor cure., Acta Oncol. 38 (6) (1999) 787–798.

[23] F.S. Lobato, V.S. Machado, V. Steffen, Determination of an optimal control strategy for drug administration in tumor treatment using multi-objective optimization differential evolution, Comput. Methods Programs Biomed. 131 (2016) 51–61.

[24] R. Mohan, G.S. Mageras, B. Baldwin, L.J. Brewster, G.J. Kutcher, S.A. Leibel, C. Burman, C.C. Ling, Z. Fuks, Clinically relevant optimization of 3-d conformal treatments, Med. Phys. 19 (4) (1992) 933–944.

[25] N. Jagiella, D. Rickert, F.J. Theis, J. Hasenauer, Parallelization and high-performance computing enables automated statistical inference of multi-scale models., Cell Syst. 4 (2) (2017) 194–206, e9.

[26] D. Tsesmelis, A. Simitsis, Database optimizers in the era of learning, in: IEEE ICDE, 2022.

[27] V. Stavropoulos, E. Alevizos, N. Giatrakos, A. Artikis, Optimizing complex event forecasting, in: DEBS, 2022.

[28] O. Poppe, T. Amuneke, D. Banda, et al., Seagull: An infrastructure for load prediction and optimized resource allocation, Proc. VLDB Endow. 14 (2) (2020) 154–162.

[29] F. Di Natale, H. Bhatia, T.S. Carpenter, C. Neale, S. Kokkila-Schumacher, T. Oppelstrup, L. Stanton, X. Zhang, S. Sundram, T.R.W. Scogland, G. Dharuman, M.P. Surh, Y. Yang, C. Misale, L. Schneidenbach, C. Costa, C. Kim, B. D'Amora, S. Gnanakaran, D.V. Nissley, F. Streitz, F.C. Lightstone, P.-T. Bremer, J.N. Glosli, H.I. Ingólfsson, A massively parallel infrastructure for adaptive multiscale simulations: Modeling ras initiation pathway for cancer, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '19, Association for Computing Machinery, New York, NY, USA, 2019, http://dx.doi.org/10.1145/3295500.3356197.

[30] J. Zhang, Y. Liu, K. Zhou, G. Li, Z. Xiao, B. Cheng, J. Xing, Y. Wang, T. Cheng, L. Liu, M. Ran, Z. Li, An end-to-end automatic cloud database tuning system using deep reinforcement learning, in: SIGMOD, 2019.

[31] M. Kunjir, S. Babu, Black or white? how to develop an autotuner for memory-based analytics, in: D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, H.Q. Ngo (Eds.), SIGMOD, 2020.

[32] I.H. Sarker, Algorithms, real-world applications and research directions, SN Comput. Sci. 2 (3) (2021) http://dx.doi.org/10.1007/s42979-021-00592-x.
[33] D. Khurana, A. Koli, K. Khatter, S. Singh, Natural language processing: State of the art, current trends and challenges, Multimedia Tools Appl. 82 (2022) http://dx.doi.org/10.1007/s11042-022-13428-4.
[34] J. Chai, H. Zeng, A. Li, E. Ngai, Deep learning in computer vision: A critical review of emerging techniques and application scenarios, Mach. Learn. Appl. 6 (2021) 100134, http://dx.doi.org/10.1016/j.mlwa.2021.100134.
[35] D. Roy, M. Dutta, A systematic review and research perspective on recommender systems, J. Big Data 9 (2022) 1–36, URL https://api.semanticscholar.org/CorpusID:248508374.
[36] L. Koumakis, Deep learning models in genomics; are we there yet? Comput. Struct. Biotechnol. J. 18 (2020) http://dx.doi.org/10.1016/j.csbj.2020.06.017.
[37] T. Bai, Y. Li, Y. Shen, X. Zhang, W. Zhang, B. Cui, Transfer learning for bayesian optimization: A survey, 2023, arXiv:2302.05927.
[38] D.H. Ahn, N. Bass, A. Chu, J. Garlick, M. Grondona, S. Herbein, H.I. Ingólfsson, J. Koning, T. Patki, T.R. Scogland, B. Springmeyer, M. Taufer, Flux: Overcoming scheduling challenges for exascale workflows, Future Gener. Comput. Syst. 110 (2020) 202–213, http://dx.doi.org/10.1016/j.future.2020.04.006, URL https://www.sciencedirect.com/science/article/pii/S0167739X19317169.
[39] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, second ed., MIT Press, 2018, URL http://incompleteideas.net/book/the-book-2nd.html.
[40] H. v. Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16, AAAI Press, 2016, pp. 2094–2100.
[41] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32, http://dx.doi.org/10.1023/A:1010933404324.