SOCRATES: Simulation Optimization with Correlated Replicas and Adaptive Trajectory Evaluations

Haoting Zhang[1,3] **Haoxian Chen**[2,3] **Donglin Zhan**[2] **Hanyang Zhao**[2]
Henry Lam[2] **Wenpin Tang**[2] **David Yao**[2] **Zeyu Zheng**[1]

[1]University of California, Berkeley [2]Columbia University [3]Amazon

{haoting_zhang, zyzheng}@berkeley.edu
{hc3136, donglin.zhan, hz2684, henry.lam, wt2319, yao}@columbia.edu

Abstract

The field of simulation optimization (SO) encompasses various methods developed to optimize complex, expensive-to-sample stochastic systems. Established methods include, but are not limited to, ranking-and-selection for finite alternatives and surrogate-based methods for continuous domains, with broad applications in engineering and operations management. The recent advent of large language models (LLMs) offers a new paradigm for exploiting system structure and automating the strategic selection and composition of these established SO methods into a tailored optimization procedure. This work introduces SOCRATES (Simulation Optimization with Correlated Replicas and Adaptive Trajectory Evaluations), a novel two-stage procedure that leverages LLMs to automate the design of tailored SO algorithms. The first stage constructs an ensemble of digital replicas of the real system. An LLM is employed to implement causal discovery from a textual description of the system, generating a structural 'skeleton' that guides the sample-efficient learning of the replicas. In the second stage, this replica ensemble is used as an inexpensive testbed to evaluate a set of baseline SO algorithms. An LLM then acts as a meta-optimizer, analyzing the performance trajectories of these algorithms to iteratively revise and compose a final, hybrid optimization schedule. This schedule is designed to be adaptive, with the ability to be updated during the final execution on the real system when the optimization performance deviates from expectations. By integrating LLM-driven reasoning with LLM-assisted trajectory-aware meta-optimization, SOCRATES creates an effective and sample-efficient solution for complex SO optimization problems.

1 Introduction

Stochastic systems can exhibit complex structures where system performance is not an analytical function of the decision variables but rather a surface that can only be evaluated through noisy samples. In many applications, the goal is to search the feasible set of decision variables in order to optimize system performance, which motivates the field of simulation optimization (SO) by regarding collecting system samples as simulations [26, 8, 12, 30]. For problems with a finite set of decisions, ranking-and-selection procedures are designed to efficiently allocate simulation budgets to identify the best decision with statistical confidence [29, 16, 22, 3]. When the decision variables are within in a continuous space, surrogate model-based algorithms are employed to approximate the system performance to optimize [4, 47, 17]; a leading representative is Bayesian optimization [14, 7, 9, 46], which employs a probabilistic model, such as a Gaussian process [10, 5, 35, 48, 25], to guide the optimization in a sample-efficient manner. Additionally, for highly non-structural and complex

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: MLxOR: Mathematical Foundations and Operational Integration of Machine Learning for Uncertainty-Aware Decision-Making.

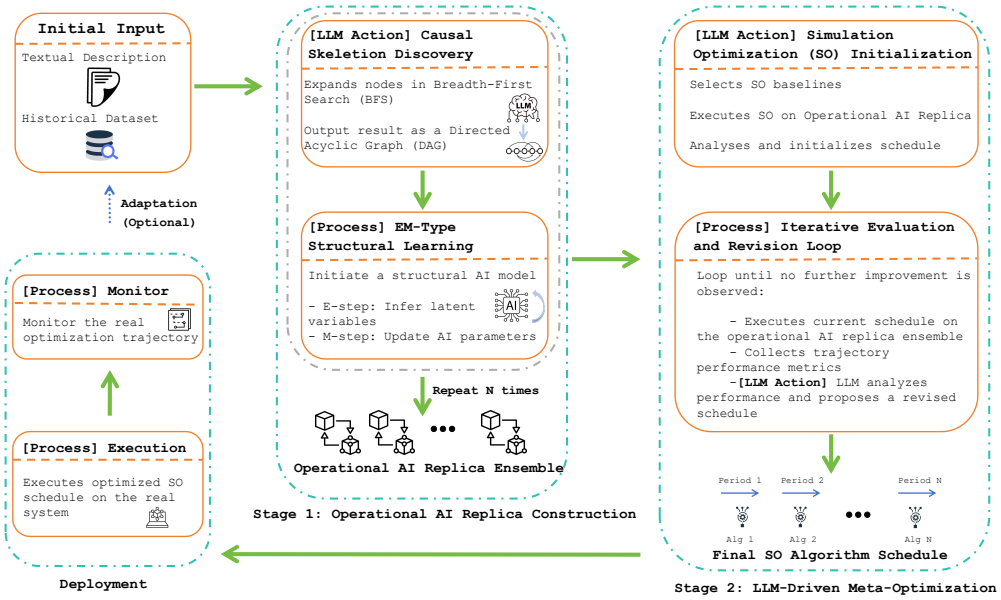

Figure 1: Pipeline of SOCRATES: Stage 1 learns an ensemble of operational AI replicas; Stage 2 employs these inexpensive replicas to evaluate and revise SO algorithms. The final SO algorithm is tailored to the real system before deployment, and an online adaptation is feasible when needed.

system performance, meta-heuristics methods provide a flexible and scalable framework [49, 18]. On the other hand, most SO methods remain general-purpose black-box procedures. Because samples are costly and noisy, there has been comparatively limited work on tailoring SO algorithms to specific problem instances or domains, leaving per-instance adaptation underexplored.

These open challenges create an opportunity for large language models (LLMs) to inject structure and prior knowledge into the SO pipeline. A primary frontier involves leveraging LLMs' extensive pre-training and semantic reasoning capabilities to translate natural language problem descriptions into optimization components. Research in this area has progressed from foundational benchmarks that parse problem statements into mathematical representations [32] to sophisticated multi-agent frameworks that decompose the modeling pipeline into specialized roles like formulation, planning, and code generation [40]. Concurrently, a significant research trend focuses on fine-tuning LLMs on domain-specific data to create specialized optimization assistants, enhancing both performance and data privacy [39, 20]. Beyond formulation and algorithm execution supported by LLMs, another paradigm employs LLMs as engines for methodological discovery, where program-synthesis and evolutionary loops produce verifiable new algorithms and heuristics [44]. Such approaches have yielded novel discoveries in mathematical sciences and improved heuristics for complex optimization problems by evolving semantic reasoning with executable implementations under automated evaluation [33, 27, 38].

Building on these opportunities, we propose *SOCRATES (Simulation Optimization with Correlated Replicas and Adaptive Trajectory Evaluations)*—an LLM-driven meta-optimizer [23] that evaluates and refines SO algorithms prior to deployment on complex, expensive-to-sample stochastic systems. Our procedure has two stages. First, we use an LLM's reasoning ability to infer a structural "skeleton" from the system's textual description, which guides data-efficient, structure-aware learning of an ensemble of system replicas (Sec. 2.1). This learning supports flexible selection of AI models and methods; we term the replicas *Operational AI Replicas (OARs)*. These OARs have an affinity to digital twins [42, 28], yet are purpose-built for evaluating and revising operations prior to deployment. In the second stage (Sec. 2.2), using the learned replicas as a testbed, we iteratively invoke LLMs as a meta-optimizer to analyze optimization-trajectory ensembles and select and refine a final SO algorithm. The resulting SO algorithm is tailored to the problem's structure before deployment on the expensive real system. Finally, the procedure is adaptive: during execution on the real system, if the observed trajectory deviates from expectation, the twin ensemble can be updated and the optimization strategy revised for the remaining budget. The entire pipeline is summarized in Figure 1.

2 Methodology

2.1 OAR Construction via LLM–Guided Causal Skeleton and EM Learning

Problem setting. We build an OAR from two inputs: (i) a textual specification $\mathcal{P}$ with system components, mechanism, constraints, etc. in natural language and (ii) a historical dataset $\mathcal{H} = \{(x_i, y_i)\}_{i=1}^N$ of controllable inputs x and observed objective y. A key challenge is the mismatch between *system complexity* and *data scarcity* (limited observations for expensive sampling). To address this, we 1) leverage LLMs to build a causal 'skeleton' that represents the system structure and 2) implement a expectation-maximization (EM)-type data-efficient learning procedure:

2.1.1 Step 1: Causal Skeleton Inference with LLMs

We first use LLMs to infer a *causal skeleton* $G = (V, E)$ over the system variables listed in the textual description $\mathcal{P}$. Here, V is the set of variables listed in $\mathcal{P}$, E denotes the set of edges that represent the causal relations between variables $v \in V$, and G is a directed acyclic graph (DAG) that summarizes the causal relations and mechanism of the system. To infer the DAG from the textual description $\mathcal{P}$, a breadth-first search (BFS) causal discovery procedure over variables V involves the following steps: **1. Initialization:** Prompt the LLM with all variable names/descriptions and ask it to select the set of *exogenous* variables (not caused by any others). These seed a BFS queue. **2. Expansion:** select a node u from the queue and ask the LLM to select all variables that are *caused by* u. **3. Insertion:** Add edges $(u \to v)$ that do not create cycles; push newly discovered children v to the queue. Repeat until the queue is empty; see [21].

To be more specific, for the variables, we let $V = X \cup Z \cup \{Y\}$ denote inputs, latent internal components, and the objective. In our implementation, the result of the initialization step is fixed to be X since they are controllable. Additionally, for the objective Y, we do not expand nodes since it is the final output. This BFS formulation reduces the number of LLM queries from $O(|V|^2)$ pairwise judgments to $O(|V|)$ expansions, while a per-edge cycle check guarantees a DAG. In addition, we include in the prompt a history of the inference, which accumulates previously discovered edges to improve consistency across turns of calling LLMs.

2.1.2 Step 2: EM-Type Learning of Structural Mechanisms

Given the skeleton G, we learn node-level mechanisms with a differentiable structural model. Recall that $V = X \cup Z \cup \{Y\}$ denote inputs, latent internal components, and the objective. For each non-input node $j \in Z \cup \{Y\}$ with parents $\mathrm{pa}(j)$ in G, we let

$$\mathbf{s}_j \;=\; f_j\big(\mathbf{s}_{\mathrm{pa}(j)};\, \theta_j\big) \,+\, \boldsymbol{\varepsilon}_j, \quad \text{with} \quad Y \;=\; f_Y\big(\mathbf{s}_{\mathrm{pa}(Y)};\, \theta_Y\big) \,+\, \varepsilon_Y,$$

where $\mathbf{s}_j \in \mathbb{R}^{d_j}$ is the d_j-dimensional vector state of node j, and $\mathbf{s}_{\mathrm{pa}(j)}$ is the concatenated vector state of its parents. The objective Y remains a scalar. The map f_j is approximated by AI models (e.g., an MLP) with parameters θ_j that outputs a d_j-dimensional vector. We learn parameters $\theta = \{\theta_j\}$ and infer latent variables $Z = \{\mathbf{s}_z\}_{z \in Z}$ via an EM-type learning procedure that explicitly respects the causal skeleton G inferred by LLMs and accounts for the uncertainty within system components (latent variables). The learning is composed of alternating E/M steps as follows:

E-step (Latent Inference). With the parameters of the map θ fixed, we infer latent variables $\widehat{Z}$ by minimizing a loss that trades off end-to-end objective fit and consistency with the current mechanisms:

$$\mathcal{L}_E(\widehat{Z};\theta) \;=\; \frac{1}{N}\sum_{i=1}^{N} \underbrace{\big\|y_i - \widehat{y}(x_i, \widehat{Z}_i;\theta)\big\|_2^2}_{\text{objective fit}} \,+\, \lambda \cdot \sum_{j \in Z} \frac{1}{N}\sum_{i=1}^{N} \underbrace{\big\|\widehat{\mathbf{s}}_{j,i} - f_j(\mathbf{s}_{\mathrm{pa}(j),i};\theta_j)\big\|_2^2}_{\text{mechanism consistency}}.$$

Here $\widehat{y}$ is the forward pass through G, using input x_i and inferred latent variables $\widehat{Z}_i = \{\widehat{\mathbf{s}}_{j,i}, j \in Z\}$. The term mechanism consistency keeps the E-step from drifting far from the current mechanisms and empirically stabilizes training in low-data regimes.

M-step (Mechanism Update). With $\widehat{Z}$ fixed, we refit each component map $f_j(\,\cdot\,;\theta_j)$ by minimizing:

$$\mathcal{L}_M(\theta;\widehat{Z}) \;=\; \sum_{j \in Z} \frac{1}{N}\sum_{i=1}^{N} \big\|\widehat{\mathbf{s}}_{j,i} - f_j(\mathbf{s}_{\mathrm{pa}(j),i};\theta_j)\big\|_2^2 \;+\; \gamma \cdot \frac{1}{N}\sum_{i=1}^{N} \big\|y_i - \widehat{y}(x_i, \widehat{Z}_i;\theta)\big\|_2^2.$$

Lastly, to account for the uncertainty in LLM inference and EM-type learning procedure, we repeat the procedure multiple times independently to get a set of learned OARs. We then select top-K of them based on the accuracy, and then apply an ensemble method to decide a final OAR with an optimal ensemble weight, enhancing the resilience of the subsequent operations.

2.2 LLM-based Trajectory–Aware Meta Optimizer

Goal. Given a fixed OAR (Sec. 2.1) and a fixed evaluation budget T, our objective is to select and revise the SO algorithm that will be applied to optimizing the real system, i.e., we implement a meta optimization (optimizing the optimizer) procedure on an OAR instead of directly applying an optimization algorithm to the real system. Compared to classical meta–optimizers that rely solely on quantitative evaluations, our method leverages LLMs to reason jointly over *quantitative performances* (trajectory-based metrics) and *qualitative insights*, integrating the LLM's pre-trained knowledge of algorithmic strategies with its ability to reason semantically about the optimization process.

Let $\mathcal{A} = \{a_1, \ldots, a_m\}$ denote a set of baseline SO algorithms (e.g., variants of Bayesian optimization algorithms and meta–heuristics). Our procedure finally provides a schedule $\pi = ((a_j, T_j))_{j=1}^{J}$, where $a_j \in \mathcal{A}$ is executed for T_j budgets. That is, the final SO algorithm is composed of different SO baseline algorithms implemented in different periods, leveraging the strengths of each algorithm at different stages of the optimization. Specifically, the procedure is composed of:

1. Initialization: For the real system with the textual description $\mathcal{P}$ and the budget T, we use an LLM to select several baseline SO algorithms based on its knowledge base and a predefined selection logic. Each selected SO algorithm is then implemented on the OAR for the full budget to collect its entire optimization trajectory. Finally, the LLM decides an initial schedule $\pi^{(0)}$ by analyzing the collected trajectories, utilizing its semantic understanding and reasoning ability.

2. Iterative Revision: Starting from an initial schedule, we employ LLMs to automatically and iteratively revise it. In each iteration, we: 1) implement the current schedule on the OAR, 2) collect trajectories and the associated performance metrics, and 3) input the current schedule and its performance into an LLM to request a revision. The LLM also has access to the performance of baseline SO algorithms for reference. To facilitate directional and efficient revisions, we include the revision history in the LLM's input. If a newly proposed schedule does not outperform the previous one, we retain the earlier version but still append the new (inferior) schedule to the revision history. This allows the LLM to learn from both "successful" and "unsuccessful" attempts. The iterative revision process terminates when no further improvement is observed across successive iterations.

Additionally, we evaluate and select schedules using an *ensemble* of OARs (top-K models from Sec. 2.1). In this way, the final SO algorithm accounts for uncertainty arising from the OAR construction, thereby achieving resilience against discrepancies between the real system and the learned OAR. Furthermore, the ensemble enables us to construct a family of twins by assigning different ensemble weights. By regarding each OAR with a specific ensemble weight as a data point that provides a "loss" to minimize, the iterative revision procedure in our method can be cast as a learning epoch within the classical machine learning (ML) framework. In this manner, LLMs provide a "semantic gradient" to revise the input schedule of SO algorithms. That is, we integrate the iterative revision pipeline into a standardized ML framework that combines both the reasoning and semantic understanding of LLMs with the rigor of well-established ML training methodologies, ensuring both performance and generalizability.

In addition to the iterative revision on OARs, we implement an online *adaptation* procedure during the final execution on the real system. We monitor the real optimization trajectory and the performance metrics. If this trajectory exhibits unexpected behavior compared to the performance simulated on OARs, we update the optimal ensemble weight with the new real-system data and call LLMs to iteratively revise the schedule for the remaining budget. This allows the meta-optimizer to adapt to potential mismatches between the learned OAR and the real system, further improving the data-efficiency of the final optimization. The detailed procedure of SOCRATES, as well as the experimental results, is included in the appendix.

References

[1] M. J. Abdel-Rahman, Y. Alslman, D. Refai, A. Saleh, M. A. A. Loha, and M. Y. Hamed. Teaching llms to think mathematically: A critical study of decision-making via optimization. *arXiv preprint arXiv:2508.18091*, 2025.

[2] A. AhmadiTeshnizi, W. Gao, and M. Udell. Optimus: Scalable optimization modeling with (mi) lp solvers and large language models. *arXiv preprint arXiv:2402.10172*, 2024.

[3] Y. Bai, T. Balch, H. Chen, D. Dervovic, H. Lam, and S. Vyetrenko. Calibrating non-identifiable high-dimensional simulation models: A framework via eligibility set. *ACM Transactions on Modeling and Computer Simulation*, 2025.

[4] R. R. Barton. Simulation optimization using metamodels. In *Proceedings of the 2009 winter simulation conference (WSC)*, pages 230–238. IEEE, 2009.

[5] R. R. Barton, B. L. Nelson, and W. Xie. Quantifying input uncertainty via simulation confidence intervals. *INFORMS journal on computing*, 26(1):74–87, 2014.

[6] B. Biller, J. Yi, and S. Biller. A practitioner's guide to digital twin development. In *Tutorials in Operations Research: Advancing the Frontiers of OR/MS: From Methodologies to Applications*, pages 198–227. INFORMS, 2023.

[7] S. Cakmak, R. Astudillo Marban, P. Frazier, and E. Zhou. Bayesian optimization of risk measures. *Advances in Neural Information Processing Systems*, 33:20130–20141, 2020.

[8] H. Chen, Z. Huang, H. Lam, H. Qian, and H. Zhang. Learning prediction intervals for regression: Generalization and calibration. In *International Conference on Artificial Intelligence and Statistics*, pages 820–828. PMLR, 2021.

[9] H. Chen and H. Lam. Pseudo-bayesian optimization. *arXiv preprint arXiv:2310.09766*, 2023.

[10] X. Chen, B. E. Ankenman, and B. L. Nelson. Enhancing stochastic kriging metamodels with gradient estimators. *Operations Research*, 61(2):512–528, 2013.

[11] T. Dai, J. Wong, Y. Jiang, C. Wang, C. Gokmen, R. Zhang, J. Wu, and L. Fei-Fei. Automated creation of digital cousins for robust policy learning. *arXiv preprint arXiv:2410.07408*, 2024.

[12] D. J. Eckman, S. G. Henderson, and S. Shashaani. Simopt: A testbed for simulation-optimization experiments. *INFORMS Journal on Computing*, 35(2):495–508, 2023.

[13] H. Fang, B. Han, N. Erickson, X. Zhang, S. Zhou, A. Dagar, J. Zhang, A. C. Turkmen, C. Hu, and H. Rangwala. Mlzero: A multi-agent system for end-to-end machine learning automation. *arXiv preprint arXiv:2505.13941*, 2025.

[14] P. I. Frazier. Bayesian optimization. In *Recent advances in optimization and modeling of contemporary problems*, pages 255–278. Informs, 2018.

[15] X. He, K. Zhao, and X. Chu. Automl: A survey of the state-of-the-art. *Knowledge-based systems*, 212:106622, 2021.

[16] L. J. Hong, W. Fan, and J. Luo. Review on ranking and selection: A new perspective. *Frontiers of Engineering Management*, 8(3):321–343, 2021.

[17] L. J. Hong and X. Zhang. Surrogate-based simulation optimization. In *Tutorials in Operations Research: Emerging Optimization Methods and Modeling Techniques with Applications*, pages 287–311. INFORMS, 2021.

[18] J. Hu, E. Zhou, and Q. Fan. Model-based annealing random search with stochastic averaging. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 24(4):1–23, 2014.

[19] X. Huang, Q. Shen, Y. Hu, A. Gao, and B. Wang. Llms for mathematical modeling: Towards bridging the gap between natural and mathematical languages. *arXiv preprint arXiv:2405.13144*, 2024.

[20] C. Jiang, X. Shu, H. Qian, X. Lu, J. Zhou, A. Zhou, and Y. Yu. LLMOPT: Learning to define and solve general optimization problems from scratch. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[21] T. Jiralerspong, X. Chen, Y. More, V. Shah, and Y. Bengio. Efficient causal graph discovery using large language models. *arXiv preprint arXiv:2402.01207*, 2024.

[22] G. Keslin, B. L. Nelson, B. Pagnoncelli, M. Plumlee, and H. Rahimian. Ranking and contextual selection. *Operations Research*, 2024.

[23] P. Krus and J. Andersson. Optimizing optimization for design optimization. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 37009, pages 951–960, 2003.

[24] J. Kudela and L. Dobrovsky. Performance comparison of surrogate-assisted evolutionary algorithms on computational fluid dynamics problems. In *International Conference on Parallel Problem Solving from Nature*, pages 303–321. Springer, 2024.

[25] D. Li, W. Tang, and S. Banerjee. Inference for gaussian processes with matérn covariogram on compact riemannian manifolds. *Journal of machine learning research*, 24(101):1–26, 2023.

[26] H. Lin, H. Chen, K. M. Choromanski, T. Zhang, and C. Laroche. Demystifying orthogonal monte carlo and beyond. *Advances in Neural Information Processing Systems*, 33:8030–8041, 2020.

[27] F. Liu, T. Xialiang, M. Yuan, X. Lin, F. Luo, Z. Wang, Z. Lu, and Q. Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 32201–32223. PMLR, 2024.

[28] A. Matta and G. Lugaresi. Digital twins: Features, models, and services. In *2023 Winter Simulation Conference (WSC)*, pages 46–60. IEEE, 2023.

[29] E. C. Ni, D. F. Ciocan, S. G. Henderson, and S. R. Hunter. Efficient ranking and selection in parallel computing environments. *Operations Research*, 65(3):821–836, 2017.

[30] Y. Peng, C.-H. Chen, and M. C. Fu. A review of simulation optimization with connection to artificial intelligence. *Fundamental Research*, 2025.

[31] G. Prockl, Y. Bouras, and T. Jensen. Supply chain & digital twins: Mature enough? In *The 58th Hawaii International Conference on System Sciences. HICSS 2025*, pages 4129–4138. Hawaii International Conference on System Sciences (HICSS), 2025.

[32] R. Ramamonjison, T. Yu, R. Li, H. Li, G. Carenini, B. Ghaddar, S. He, M. Mostajabdaveh, A. Banitalebi-Dehkordi, Z. Zhou, and Y. Zhang. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In M. Ciccone, G. Stolovitzky, and J. Albrecht, editors, *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pages 189–203. PMLR, 28 Nov–09 Dec 2022.

[33] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. R. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi, P. Kohli, and A. Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

[34] D. Roth and V. Srikumar. Integer linear programming formulations in natural language processing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Tutorial Abstracts*, 2017.

[35] I. O. Ryzhov. On the convergence rates of expected improvement methods. *Operations Research*, 64(6):1515–1528, 2016.

[36] M. L. Santoni, E. Raponi, R. D. Leone, and C. Doerr. Comparison of high-dimensional bayesian optimization algorithms on bbob. *ACM Transactions on Evolutionary Learning*, 4(3):1–33, 2024.

[37] J. Srai and E. Settanni. Supply chain digital twins: Opportunities and challenges beyond the hype. 2019.

[38] A. Surina, A. Mansouri, L. Quaedvlieg, A. Seddas, M. Viazovska, E. Abbe, and C. Gulcehre. Algorithm discovery with llms: Evolutionary search meets reinforcement learning, 2025.

[39] Z. Tang, C. Huang, X. Zheng, S. Hu, Z. Wang, D. Ge, and B. Wang. Orlm: Training large language models for optimization modeling, 2024.

[40] R. Thind, Y. Sun, L. Liang, and H. Yang. Optimai: Optimization from natural language using llm-powered ai agents, 2025.

[41] V. Volz and B. Naujoks. On benchmarking surrogate-assisted evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1603–1605, 2019.

[42] K. Wang, W. Xie, B. Wang, J. Pei, W. Wu, M. Baker, and Q. Zhou. Simulation-based digital twin development for blockchain enabled end-to-end industrial hemp supply chain risk management. In *2020 Winter Simulation Conference (WSC)*, pages 3200–3211. IEEE, 2020.

[43] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[44] X. Wu, S. hao Wu, J. Wu, L. Feng, and K. C. Tan. Evolutionary computation in the era of large language model: Survey and roadmap, 2024.

[45] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2023.

[46] D. Zhan, H. Zhang, R. Righter, Z. Zheng, and J. Anderson. Collaborative bayesian optimization via wasserstein barycenters. *arXiv preprint arXiv:2504.10770*, 2025.

[47] H. Zhang, J. He, D. Zhan, and Z. Zheng. Neural network-assisted simulation optimization with covariates. In *2021 Winter Simulation Conference (WSC)*, pages 1–12. IEEE, 2021.

[48] L. Zhang, W. Tang, and S. Banerjee. Bayesian geostatistics using predictive stacking. *arXiv preprint arXiv:2304.12414*, 2023.

[49] E. Zhou and X. Chen. Sequential monte carlo simulated annealing. *Journal of Global Optimization*, 55(1):101–124, 2013.

A Detailed Methodology

Here we include the detailed procedure of SOCRATES as a complement of Sec. 2.

A.1 Operational AI Replica Construction

During the first stage of SOCRATES, we construct an ensemble of operational AI replicas (OARs), which leverage LLMs to 1) approximate the system under different "what-if" scenarios, and 2) provide a cost-effective "testbed" to evaluate and revise simulation optimization (SO) algorithms before execution on the expensive real system. The construction of OARs takes two types of input:

1. A textual specification $\mathcal{P}$ with system components, mechanism, constraints, etc., in natural language;
2. A historical input/output (I/O) dataset $\mathcal{H} = \{(x_i, y_i)\}_{i=1}^{N}$ of controllable inputs x and observed objective y.

With these two sources of input, the construction of OARs begins with the causal relation discovery of the system components to *qualify* the structure of the system. The inferred causal relations serve as the "skeleton" of OAR, and we then implement an expectation-maximization (EM) procedure to *quantify* the dependencies between components in OAR.

A.1.1 Causal Relation Inference

Let $G = (V, E)$ denote the directed acyclic graph (DAG) to be inferred from a textual description $\mathcal{P}$ of the variables. We partition the variables as $V = X \cup Z \cup \{Y\}$, where X are system inputs (the decision variables for SO), Z are latent internal components, and Y is the system input (the objective value for SO). The implementation employs the breadth–first search (BFS) procedure: 1) Initialization, 2) Expansion, and 3) Insertion with cycle checks:

1. **Initialization:** The procedure begins with deciding the set of variables $V = X \cup Z \cup \{Y\}$:
   - A set V of variable names, each with a short natural-language description, extracted by an LLM from the textual description $\mathcal{P}$.
   - The designated objective $Y \in V$ (treated as a sink: no outgoing edges).
   - A subset $X \subseteq V$ of exogenous variables. When provided, edges **into** X are disallowed so that X remains exogenous, i.e., $x \in X$ is not dependent on any other $v' \in V$.

   After deciding the variable set V, we initialize the state of DAG:
   - A directed graph G initialized with node set V and empty edge set E.
   - A list that tracks the variables (nodes of the graph) that have been visited to ensure each node is expanded at most once. We also record a *queue* that decides the order of nodes to be expanded, which is initialized with the set of exogenous variables X.
   - A causal discovery history recording the instructions summarized by an LLM from the textual description $\mathcal{P}$, the variable catalog $\{\langle v \rangle : \text{description}(v)\}_{v \in V}$, and, after each turn, the decided causal relation (the inserted edge of the graph) $(u \to v)$; this history is fed back into subsequent prompts input to LLM to improve consistency across turns.
2. **Expansion (LLM query at a node):** We decide the expansion node by node. While the queue is non-empty, select a node u from it (first-in-first-out). The prompt into an LLM to ask for expansion contains:
   (a) The list of exogenous variables.
   (b) A causal discovery history recording the general instruction, the variable description, and the currently discovered causal relationships to date. The history is dynamically compacted by an LLM when the length exceeds the pre-specified threshold.
   (c) The instruction for the current node u for expansion: *"Select ALL variables that are caused by $\langle u \rangle$."*

   The output of LLM is restricted (through the prompt) to return a set of variable names from V, which is then used as the proposed children of u.

3. **Insertion with checks:** For each proposed child v of u, the implementation executes the following admissibility checks before inserting the edge $(u \to v)$:
   (a) Sink constraint: if $u = Y$, reject (enforces that Y has no outgoing edges).
   (b) Exogeneity constraint: if $v \in X$, reject (keeps X exogenous).
   (c) Acyclicity: tentatively add $(u \to v)$; if this creates a directed cycle, revert and reject.

   If $(u \to v)$ is accepted, then v is appended to the queue if it has not yet been visited or enqueued. The causal discovery history is updated as well. This per-edge cycle check guarantees that the final inferred causal 'skeleton' G is DAG. In most scenarios, the proposed children of u pass the insertion since all the requirements (e.g, acyclicity) and the information (X, Y, and previously-visited nodes, etc.) are provided in the prompt to LLM. On the other hand, LLMs cannot always strictly follow all the requirements, especially when the causal discovery history is compacted, and therefore these admissibility checks are necessary.

The BFS procedure terminates when the queue is empty. The complexity in LLM calls is $O(|V|)$ expansions since each node is expanded at most once. Additionally, we consider the scenarios when only the I/O historical data is available. Since I/O is pre-fixed during the causal relation discovery in our implementation, the I/O data is not utilized, i.e., the causal relation discovery is based on the domain knowledge provided in the textual description $\mathcal{P}$ and the knowledge base of LLMs attained in extensive pre-training. In some scenarios when the data of intermediate variables Z is available, the Expansion step augments the prompt to LLMs by providing the statistics in data (e.g. Pearson correlations) between the current node u and other variables. This strategy has the potential to provide additional instructions that can help the LLM when domain text alone is ambiguous.

A.1.2 EM-type Learning

Given the LLM-inferred causal skeleton $G = (V, E)$, we learn an OAR by attaching a learnable mechanism $f_j(\cdot; \theta_j)$ to each non-input node $j \in Z \cup Y$, with inputs given by the states of its parents $\mathrm{pa}(j)$ in G. Written in vector form,

$$\mathbf{s}_j = f_j\left(\mathbf{s}_{\mathrm{pa}(j)}; \theta_j\right) + \boldsymbol{\varepsilon}_j, \quad Y = f_Y\left(\mathbf{s}_{\mathrm{pa}(Y)}; \theta_Y\right) + \varepsilon_Y.$$

The collection $f(\cdot; \theta) = f_j(\cdot; \theta_j)_{j \in Z \cup Y}$ defines the OAR. The *model family for each f_j is flexible*: small MLPs, gradient-boosted trees, monotone nets, or other light parametric learners can be selected per node based on variable type, smoothness, and known monotonicities. When a variable arrives in a raw, non-numeric modality (categorical/text/ordinal, etc.), we introduce learned or library embeddings e_j to map raw inputs into numeric states used by f_j. In practice, this *per-node model selection* can be automated by an AutoML layer [15], optionally seeded by LLM guidance; recent systems such as MLZero [13] demonstrate promising LLM-augmented AutoML that we can leverage to pick reasonable f_j families and embeddings without changing the EM loop.

We learn parameters θ and latent node variables Z by alternating: (i) an *E-step* that infers per-sample latent states $\widehat{Z}$ with the mechanisms held fixed, balancing end-to-end objective fit and per-node consistency; and (ii) an *M-step* that refits the mechanisms to the inferred latents, with a small end-to-end term to keep global performance aligned. Before the first E/M iteration, we perform a Stage-0 end-to-end fit over the historical set that ignores explicit latents (i.e., forward only through G), which supplies an informed starting point $\theta^{(0)}$ and an initial set of latent states via a forward pass.

After we have set the model parameter θ, the learning procedure of OAR is iterated by E, M steps, where we optimize latent variables $\widehat{Z}$ and the model parameters θ to minimize the loss functions respectively:

$$\mathcal{L}_E(\widehat{Z}; \theta) = \frac{1}{N} \sum^{N} \left\| y_i - \widehat{y}\left(x_i, \widehat{Z}_i; \theta\right) \right\|_2^2 + \lambda \sum_{j \in Z} \frac{1}{N} \sum_{i=1}^{N} \left\| \widehat{\mathbf{s}}_{j,i} - f_j\left(\mathbf{s}_{\mathrm{pa}(j),i}; \theta_j\right) \right\|_2^2,$$

$$\mathcal{L}_M(\theta; \widehat{Z}) = \sum_{j \in Z} \frac{1}{N} \sum_{i=1}^{N} \left\| \widehat{\mathbf{s}}_{j,i} - f_j\left(\mathbf{s}_{\mathrm{pa}(j),i}; \theta_j\right) \right\|_2^2 + \gamma \frac{1}{N} \sum_{i=1}^{N} \left\| y_i - \widehat{y}\left(x_i, \widehat{Z}_i; \theta\right) \right\|_2^2.$$

Here $\lambda, \gamma > 0$ trade local mechanism fidelity and global predictive performance. Additionally, since EM procedure is sensitive to the "starting point," we therefore implement a multi-start learning procedure and select the best model based on a leave-out test set. The detailed procedure of learning OAR is included in Alg. 1.

Algorithm 1 Structure-Aware OAR Learning: Stage-0 + EM with Multi-Start and Validation

Require: Historical data $\mathcal{H} = \{(x_i, y_i)\}_{i=1}^{N}$; inferred causal skeleton $G = (V, E)$; train/val/test split ratios; EM rounds R; number of starts S.

Ensure: A learned OAR $f(\cdot\,; \theta^\star)$.

Split $\mathcal{H}$ into $\mathcal{H}_{\text{train}}, \mathcal{H}_{\text{val}}, \mathcal{H}_{\text{test}}$ (fixed once per input).

for $s = 1$ **to** S **do** ▷ Multi-start restarts to reduce local minima

  Per-node model/embedding selection:
  For each $j \in Z \cup \{Y\}$, choose a candidate family $f_j \in \mathcal{F}_j$ and (if needed) encoder e_j via rules or AutoML.

  Initialize $\theta^{(0,s)}$ (random or AutoML warm start); fix topology by G.

  Stage-0 end-to-end fit:
  Train θ on $\mathcal{H}_{\text{train}}$ to minimize:

$$\frac{1}{|\mathcal{H}_{\text{train}}|} \sum_{(x,y)\in\mathcal{H}_{\text{train}}} |y - \hat{y}(x;\theta)|^2$$

  with early stopping on $\mathcal{H}_{\text{val}}$; set $\theta^{(0,s)} \leftarrow$ best checkpoint.

  Initialize latents $\hat{Z}^{(0,s)}$:
  Forward pass with $\theta^{(0,s)}$ on $\mathcal{H}_{\text{train}}$;
  For observed internals, clamp $\hat{\mathbf{z}}_j \leftarrow e_j(\text{raw})$.

  for $t = 1$ **to** R **do** ▷ EM rounds

    E-step: With $\theta^{(t-1,s)}$ fixed, update $\hat{Z}$ on $\mathcal{H}_{\text{train}}$ by K_E gradient steps to reduce $\mathcal{L}_E(\hat{Z}; \theta^{(t-1,s)})$.
    Clamp known internals; project to feasible bounds if needed;
    warm-start from $\hat{Z}^{(t-1,s)}$.

    M-step: With $\hat{Z}^{(t,s)}$ fixed, refit θ on $\mathcal{H}_{\text{train}}$ to reduce $\mathcal{L}_M(\theta; \hat{Z}^{(t,s)})$ with early stopping on $\mathcal{H}_{\text{val}}$;
    set $\theta^{(t,s)} \leftarrow$ best checkpoint this round.

    Validation snapshot: Compute

$$\text{MSE}_{\text{val}}^{(t,s)} = \frac{1}{|\mathcal{H}_{\text{val}}|} \sum_{(x,y)\in\mathcal{H}_{\text{val}}} |y - \hat{y}(x, \hat{Z}^{(t,s)}; \theta^{(t,s)})|^2$$

    Keep the best $(\theta, \hat{Z})$ seen so far within this start.

    if no improvement for τ rounds **then break**

    end if

  end for

  Per-start test score: Evaluate the best checkpoint for start s on $\mathcal{H}_{\text{test}}$; record $\text{MSE}_{\text{test}}^{(s)}$.

end for

Select $s^\star = \arg\min_s \text{MSE}_{\text{test}}^{(s)}$ and return the corresponding OAR $f(\cdot\,; \theta^\star)$.

Why structure-aware EM vs. pure end-to-end. Compared with a single unstructured end-to-end model (e.g, an MLP from X to Y), the proposed learning is:

1. **Data-efficient.** The LLM-guided skeleton prunes the hypothesis space; the EM splits learning into local maps constrained by G, which substantially lowers sample complexity in scarce-data regimes.

2. **Interpretable.** Each f_j is a named, low-dimensional mechanism attached to a node; latent variables $\widehat{Z}$ are explicit and can be inspected, constrained, or partially instrumented. Instead, an end-to-end model is a black box, of which the performance is difficult to explain.

3. **Flexible to partial observability.** Whenever some internal components are measured (fully or sporadically), the E-step can clamp or supervise those states, and the M-step can fit f_j directly without re-architecting the entire model.

To account for the uncertainty in LLM-based causal relation discovery and EM-type learning procedure, we implement the OAR construction J times independently (repeating Alg. 1 from scratch), and get a set of learned OARs:

$$\{(\mathrm{OAR}_1, \mathrm{MSE}_1), (\mathrm{OAR}_2, \mathrm{MSE}_2), \ldots, (\mathrm{OAR}_J, \mathrm{MSE}_J)\},$$

where MSE_j represents the fidelity of OAR_j as in Alg. 1. Then we select a top-K of OARs with the highest fidelity, i.e.,

$$\mathcal{K}^\star = \{k_{(1)}, \ldots, k_{(K)}\} \subset \{1, \ldots, J\}, \quad \mathrm{MSE}_{k_{(1)}} \leq \cdots \leq \mathrm{MSE}_{k_{(K)}},$$

and construct an ensemble:

$$\mathrm{OAR}^\star = \sum_{k \in \mathcal{K}^*} w_k^\star \, \mathrm{OAR}_k \, .$$

Here the optimal ensemble weight is decided by $w_k^\star = \frac{\frac{1}{\mathrm{MSE}_k + \epsilon}}{\sum_{k' \in \mathcal{K}^\star} \frac{1}{\mathrm{MSE}_{k'} + \epsilon}}$ with a fixed constant $\epsilon > 0$ to numerically stabilize the weight. In this manner, the ensemble of OARs exhibits the resilience against the uncertainty involved in the construction procedure, especially hallucinations from LLMs.

In addition to resilience, the ensemble of OARs brings two additional advantages: First, by re-weighting the ensemble weights around the optimal weight $w^\star$, we have access to a *family* of OARs, which allows for generating sufficient "data points" for the subsequent meta-optimization procedure. Second, the ensemble of OARs shares similar spirits with Bayesian model averaging (BMA). In this manner, when the I/O dataset is updated with new observations, the optimal ensemble weight of OARs can be updated efficiently instead of re-constructing the OARs from scratch.

A.2 Trajectory-Aware Meta Optimizer

At the second stage of SOCRATES, we leverage LLMs to evaluate and revise SO algorithms, based on the ensembles of OARs learned at the first stage. Both the evaluation and revision are supported by the semantic reasoning of LLMs. In this manner, the update of SO algorithms proposed by LLMs can be regarded as "semantic gradient [45]," and we then cast the second stage into a rigorous learning framework to guarantee the generalizability of the final SO algorithm.

A.2.1 Dataset Construction for Meta-Optimization via OAR Ensembles

We begin with constructing a "dataset" for meta-optimization. Recall that, $\{\mathrm{OAR}_k, \ \mathrm{MSE}_k\}_{k \in \mathcal{K}^\star}$ denote the top-K OARs. Let $\boldsymbol{w}^\star = (w_1^\star, w_2^\star, \ldots, w_K^\star)^\top \in \Delta^{K-1}$ be the optimal ensemble weight on the probability simplex

$$w_k^\star \ \propto \ \frac{1}{\mathrm{MSE}_k + \epsilon}, \qquad \sum_{k \in \mathcal{K}^\star} w_k^\star = 1, \quad w_k^\star \geq 0,$$

with a small $\epsilon > 0$ for numerical stability. An OAR ensemble parameterized by any $\boldsymbol{w} \in \Delta^{K-1}$ is the linear mixture

$$\widetilde{\mathrm{OAR}}(\,\cdot\,; \boldsymbol{w}) \ := \ \sum_{k \in \mathcal{K}^\star} w_k \ \mathrm{OAR}_k(\,\cdot\,),$$

which we regard as one "data point" for meta-optimization. To facilitate a rigorous learning framework, we construct a data with M data points. Below we specify: (i) how to *sample* many such ensembles $\boldsymbol{w}^{(m)}$ around $\boldsymbol{w}^\star$; (ii) how to assign each a scalar *importance weight* $\tilde{\omega}_m$; and (iii) how to obtain a weighted, stratified *train/validation/test* split.

1. **Sampling ensemble weights near $\boldsymbol{w}^\star$:** We draw M weight K-dimensional vectors $\{\boldsymbol{w}^{(m)}\}_{m=1}^M$ from a mixture of L Dirichlet distributions centered at $\boldsymbol{w}^\star$ with varying concentration ("radius") to balance *local* exploration near $\boldsymbol{w}^\star$ and *global* coverage of the simplex:

$$\boldsymbol{w}^{(m)} \sim q(\boldsymbol{w}) \ := \ \textstyle\sum_{\ell=1}^L \pi_\ell \, \mathrm{Dir}\big(\boldsymbol{\alpha}^{(\ell)}\big), \qquad \boldsymbol{\alpha}^{(\ell)} \ := \ \tau_\ell \, \big(\, (1 - \delta K) \, \boldsymbol{w}^\star + \delta \, \mathbf{1} \, \big),$$
$$\tau_1 < \tau_2 < \cdots < \tau_L, \quad \textstyle\sum_\ell \pi_\ell = 1, \quad \delta \in (0, 1/K),$$

where τ_ℓ controls concentration around $\boldsymbol{w}^\star$ (large τ_ℓ yields tight samples near $\boldsymbol{w}^\star$; small τ_ℓ explores farther), and a tiny floor δ prevents zero Dirichlet parameters when some $w_k^\star = 0$. A simple and effective choice is a geometric ladder $\tau_\ell = \tau_{\min} \, \rho^{\,\ell - 1}$ with $\rho > 1$ and uniform $\pi_\ell = 1/L$.

2. **Defining importance weights:** Each sampled ensemble $\boldsymbol{w}^{(m)}$ receives an importance weight $\tilde{\omega}_m$ considering two aspects: (i) *closeness* to the best ensemble $\boldsymbol{w}^\star$, and (ii) *fidelity* implied by the base-OAR MSEs. We quantify closeness by the KL divergence on the simplex and approximate ensemble fidelity using MSE:

$$D_m \;:=\; D_{\mathrm{KL}}\big(\boldsymbol{w}^{(m)} \,\|\, \boldsymbol{w}^\star\big), \qquad \widehat{\mathrm{MSE}}\big(\boldsymbol{w}^{(m)}\big) \;:=\; \sum_{k\in\mathcal{K}^\star} w_k^{(m)}\,\mathrm{MSE}_k.$$

We set the importance weight:

$$\zeta_m \;:=\; \exp\Big(-\alpha\, D_m\Big) \;\cdot\; \Big(\widehat{\mathrm{MSE}}(\boldsymbol{w}^{(m)}) + \epsilon\Big)^{-\beta},$$

with hyperparameters $\alpha, \beta > 0$, and then normalize the weight as:

$$\tilde{\omega}_m \;:=\; \frac{\zeta_m}{\sum_{j=1}^{M} \zeta_j}.$$

The dataset is the weighted set of ensemble OARs

$$\mathcal{D} \;:=\; \Big\{\, \big(\widetilde{\mathrm{OAR}}^{(m)},\, \boldsymbol{w}^{(m)},\, \tilde{\omega}_m\big) \,\Big\}_{m=1}^{M}, \qquad \widetilde{\mathrm{OAR}}^{(m)}(\,\cdot\,) \;:=\; \sum_{k\in\mathcal{K}^\star} w_k^{(m)}\;\mathrm{OAR}_k(\,\cdot\,).$$

In subsequent meta-optimization, SO algorithms are run on subsets of $\mathcal{D}$ to generate trajectories; the performance of algorithms is aggregated using the weights $\tilde{\omega}_m$, while the full trajectories are retained to drive LLM-based revisions.

3. **Weighted, stratified train/validation/test split.** We partition $\mathcal{D}$ into $(\mathcal{D}_{\mathrm{train}}, \mathcal{D}_{\mathrm{val}}, \mathcal{D}_{\mathrm{test}})$ with proportions $(\rho_{\mathrm{tr}}, \rho_{\mathrm{val}}, \rho_{\mathrm{te}})$ (e.g., $0.70/0.15/0.15$), ensuring both (i) coverage across distances to $\boldsymbol{w}^\star$ and (ii) respect for the importance weights.

   (a) **Distance stratification.** Compute distances D_m and define S strata by weighted quantiles of $\{D_m\}_{m=1}^{M}$ under weights $\tilde{\omega}_m$; i.e., pick cutpoints $0 = q_0 < q_1 < \cdots < q_S = 1$ and choose thresholds d_s such that

   $$\sum_{m:\, D_m \le d_s} \tilde{\omega}_m \;=\; q_s \sum_{m=1}^{M} \tilde{\omega}_m.$$

   Let $\mathcal{I}_s = \{m :\; d_{s-1} < D_m \le d_s\}$ be stratum s.

   (b) **Per-stratum targets.** For each stratum s, compute total weight $W_s = \sum_{m\in\mathcal{I}_s} \tilde{\omega}_m$ and set target weights

   $$T_s^{\mathrm{tr}} = \rho_{\mathrm{tr}}\, W_s, \qquad T_s^{\mathrm{val}} = \rho_{\mathrm{val}}\, W_s, \qquad T_s^{\mathrm{te}} = \rho_{\mathrm{te}}\, W_s.$$

   (c) **Weighted selection (without replacement).** Within each $\mathcal{I}_s$, sample indices into the train set sequentially *without replacement* with probabilities proportional to $\tilde{\omega}_m$ until the accumulated weight $\sum_{m\in\mathcal{D}_{\mathrm{train}}\cap\mathcal{I}_s} \tilde{\omega}_m$ first exceeds T_s^{tr} (greedy acceptance with stochastic tie-breaking). Repeat on the remaining pool for validation until exceeding T_s^{val}; assign the rest in $\mathcal{I}_s$ to test. Optionally, to reduce variance, enforce a minimal count per stratum for each split (e.g., at least 2 items) by adjusting the last few assignments with the smallest absolute deviation from the targets.

   This procedure yields $(\mathcal{D}_{\mathrm{train}}, \mathcal{D}_{\mathrm{val}}, \mathcal{D}_{\mathrm{test}})$ that (i) preserve weighted coverage across a continuum of closeness to $\boldsymbol{w}^\star$ (near, mid, far shells), (ii) concentrate effort according to $\tilde{\omega}_m$, and (iii) avoid leakage by sampling without replacement. The train split is then used to drive iterative, trajectory-based algorithm revision; the validation split is held out to decide when a revision overfits the training ensemble pool; the test split is kept fixed for final selection among candidate revised algorithms.

The construction of the dataset is flexible with different strategies of sampling ensemble weights, defining importance weights, and dataset split. The above serves as an example for illustration.

A.2.2 Learning Framework of LLM-Driven SO Evaluation & Revision

Given the OAR-ensemble dataset $\mathcal{D} = \{(\widetilde{\mathrm{OAR}}^{(m)}, \boldsymbol{w}^{(m)}, \tilde{\omega}_m)\}_{m=1}^{M}$ from Sec. A.2.1, a fixed SO budget B, and a library $\mathcal{A}$ of baseline SO algorithms, the goal is to learn a *schedule* (piecewise execution plan)

$$\pi \;=\; ((a_j,\, T_j))_{j=1}^{J}\,, \qquad a_j \in \mathcal{A},\;\; T_j \in \mathbb{N},\;\; \sum_{j=1}^{J} T_j = B,$$

that will be deployed on the real system. The learning proceeds in *epochs*. Within an epoch, an LLM proposes *trajectory-aware revisions* of π using (i) the full SO trajectories generated on OAR ensembles in the training split and (ii) a textual revision history accumulated *within the current epoch* (to learn from both successful and unsuccessful attempts). Revisions are accepted only if they improve a weighted scalar score computed from per-trajectory metrics. After several intra-epoch revisions, the candidate schedule is validated on the validation split to detect overfitting; epochs with a large train–validation gap are rejected. The test split is fixed for the final schedule selection across multiple outer runs (distinct train/validation partitions and initializations). We summarize the learning framework in Alg. 2, and more details are included as follows:

Setup and notation. Let $\mathcal{D}_{\mathrm{train}}, \mathcal{D}_{\mathrm{val}}, \mathcal{D}_{\mathrm{test}}$ be the weighted splits of $\mathcal{D}$. For minimization (the maximization case follows by sign change), executing a schedule π on an ensemble $\widetilde{\mathrm{OAR}}^{(m)}$ yields a trajectory up to time T

$$\mathsf{Traj}^{(m)}(\pi) \;=\; \Big\{\Big(x_t^{(m)},\, y_t^{(m)}\Big)\Big\}_{t=1}^{T}\,, \tag{1}$$

where $x_t^{(m)}$ is the decision variable and $y_t^{(m)}$ is the objective value.The LLM revision operator is an abstract map

$$\widehat{\pi} \;=\; \mathsf{Rev}_{\mathrm{LLM}}\Big(\pi,\; \{\mathsf{Traj}^{(m)}(\pi)\}_{m\in\mathcal{I}},\; \mathcal{H}\Big)\,,$$

which takes the current schedule π, a set of training trajectories indexed by $\mathcal{I} \subseteq \{1, \ldots, M\}$, and a textual history $\mathcal{H}$ maintained *within the epoch*, and returns a candidate schedule $\widehat{\pi}$. Acceptance uses a scalar score $S(\pi \mid \mathcal{D}')$ on a split $\mathcal{D}'$ (see Alg. 3), with revision rule

$$\pi \;\leftarrow\; \begin{cases} \widehat{\pi}, & \text{if } S(\widehat{\pi} \mid \mathcal{D}_{\mathrm{train}}) \;\geq\; S(\pi \mid \mathcal{D}_{\mathrm{train}}), \\ \pi, & \text{otherwise,} \end{cases} \qquad \text{and} \quad \mathcal{H} \;\leftarrow\; \mathcal{H} \;\cup\; \text{“(schedule, metrics, score)”},$$

i.e., the schedule updates when the performance is better, while the history that includes the proposed schedule, the trajectory-based metrics, and the associated scalar score, is appended for every revision. At the end of T_{rev} intra-epoch revisions, we compute $S(\pi \mid \mathcal{D}_{\mathrm{val}})$ and accept the epoch if $\big|S(\pi \mid \mathcal{D}_{\mathrm{train}}) - S(\pi \mid \mathcal{D}_{\mathrm{val}})\big| \leq \varepsilon_{\mathrm{gap}}$; otherwise we revert to the previous epoch's π. Revision histories are *not* carried across epochs to prevent unbounded context growth and to encourage exploration of revision strategies, while the performances of the baseline algorithms are always included in the history for LLMs to refer to.

Trajectory metrics and score (minimization). For each OAR ensemble m and schedule π, let the full trajectory be as in (1), and define the best–so–far sequence $b_t^{(m)} := \min_{1\leq s\leq t} y_s^{(m)}$. To compare across ensembles without relying on baseline quantiles, we normalize per trajectory using a one-time "heavy" reference optimum $\widehat{y}_\star^{(m)}$ (obtained once per ensemble with a large budget, e.g., 10×–20× the standard budget). To avoid degeneracy when the initial gap is tiny, we adopt a robust range:

$$\rho^{(m)} \;:=\; q_{0.9}\big(\{y_t^{(m)}\}_{t=1}^{T}\big) - q_{0.1}\big(\{y_t^{(m)}\}_{t=1}^{T}\big), \qquad \Delta_\star^{(m)} \;:=\; \max\Big\{\big|b_1^{(m)} - \widehat{y}_\star^{(m)}\big|,\; \rho^{(m)}\Big\} + \varepsilon,$$

where q_α denotes the empirical α-quantile on the trajectory and $\varepsilon > 0$ is small.

We compute the following metrics (each in $[0, 1]$ by construction). We use $[z]_+ := \max\{z, 0\}$.

- **Final improvement.** Normalized net improvement (clipped):

$$I_{\mathrm{final}}^{(m)} \;:=\; \min\left\{1,\; \frac{\big(b_1^{(m)} - b_T^{(m)}\big)_+}{\Delta_\star^{(m)}}\right\}.$$

- **Any-time area under the improvement curve.** Rewards earlier gains by averaging the normalized best–so–far improvement over time:

$$\mathrm{AUC}_{\mathrm{any}}^{(m)} \;:=\; \frac{1}{T-1}\sum_{t=2}^{T}\min\left\{1,\; \frac{\big(b_1^{(m)} - b_t^{(m)}\big)_+}{\Delta_\star^{(m)}}\right\}.$$

- **Monotonicity on the raw trajectory (tolerance $\xi \geq 0$).** Fraction of non-worsening raw steps:

$$\mu_{\mathrm{raw}}^{(m)} \;:=\; \frac{1}{T-1}\sum_{t=2}^{T}\mathbb{I}\Big\{\, y_t^{(m)} \leq y_{t-1}^{(m)} + \xi \,\Big\}.$$

- **Stability via overshoot above the incumbent.** Penalizes oscillation/backtracking relative to the incumbent; larger is better:

$$\sigma_{\mathrm{osc}}^{(m)} \;:=\; 1 \;-\; \min\left\{1,\; \frac{1}{(T-1)\,\Delta_\star^{(m)}}\sum_{t=2}^{T}\big(y_t^{(m)} - b_{t-1}^{(m)}\big)_+\right\}.$$

Stacking the P metrics into $\mathbf{u}^{(m)}(\pi) \in [0,1]^P$ ($P = 4$ with the order above), the per-ensemble scalar score is a convex combination

$$s_m(\pi) \;=\; \sum_{p=1}^{P}\lambda_p\, u_p^{(m)}(\pi), \qquad \lambda_p \geq 0,\; \sum_{p=1}^{P}\lambda_p = 1. \tag{2}$$

The weights λ_p are *user-specified* to emphasize final performance vs. stability, or *estimated offline from pairwise preferences* (e.g., Bradley–Terry/logistic regression) collected by querying an LLM on trajectory plots and metric tables.

Finally, the schedule score on a split $\mathcal{D}' \subseteq \mathcal{D}$ (train/val/test) is the importance-weighted mean

$$S(\pi \mid \mathcal{D}') \;=\; \frac{\sum_{(\widetilde{\mathrm{OAR}}^{(m)},\boldsymbol{w}^{(m)},\tilde{\omega}_m)\in\mathcal{D}'}\tilde{\omega}_m\, s_m(\pi)}{\sum_{(\widetilde{\mathrm{OAR}}^{(m)},\boldsymbol{w}^{(m)},\tilde{\omega}_m)\in\mathcal{D}'}\tilde{\omega}_m}.$$

Why trajectories and why a schedule of different SO algorithms. Different SO algorithm families exhibit complementary and phase-dependent behavior. For instance, a meta-heuristic may first be deployed for a global exploration epoch to identify multiple promising regions in a given landscape. The decision to switch to a different algorithm is guided by the optimization trajectory, which reveals phase-dependent behaviors that would be obscured by a single final metric. When the algorithm's progress stalls or its search becomes narrow, the process transitions to another one (e.g., Bayesian optimization (BO) for sample-efficient refinement) [41, 36, 24]. Even within the BO class, different choices of surrogate models or acquisition functions exhibit varying performances across different phases. These phase transitions are reflected in *trajectory features* which are invisible to a single end-of-run metric. By exposing the full trajectories and their metrics to the LLM, the revision operator can (i) detect regime changes (e.g., stalling monotonicity, rising oscillation, feasibility collapse), (ii) switch to an algorithm class whose inductive bias fits the current regime (e.g., BO for fine-grained exploitation after early exploration, or a restart-capable heuristic when BO's uncertainty collapses prematurely), and (iii) allocate remaining budget across phases accordingly. Thus, *trajectory* and *schedule* mutually justify each other: trajectories provide the signals that inform switching, while a schedule integrates complementary strengths across phases to outperform a standalone baseline algorithm.

B Experiments

B.1 Multi-SKU Single-Echelon Warehouse and Base-Stock Optimization

We consider a system in supply chain management. To be more specific, the system is a single-echelon warehouse that manages multiple products day by day. Each product has a fixed replenishment lead time, a holding cost for items kept on the shelf, and a backorder penalty for unfilled demand. Daily

Algorithm 2 Learning Framework of an SO Schedule on OAR Ensembles

Require: Problem description $\mathcal{P}$; OAR ensemble dataset $\mathcal{D} = \{(\widetilde{\mathrm{OAR}}_m, \tilde{w}_m)\}_{m=1}^{M}$ with importance weights $\tilde{w}_m > 0$; fixed test split $\mathcal{D}_{\text{test}} \subset \mathcal{D}$; number of outer runs R; epochs per run E; intra-epoch revision steps T_{rev}; generalization threshold $\varepsilon_{\text{gap}} > 0$; budget B; the function $\textsc{ScoreAndLog}(\pi, \mathcal{D})$ defined in Alg. 3.

Ensure: Final SO schedule $\pi^\star$ selected by performance on $\mathcal{D}_{\text{test}}$.

Fix $\mathcal{D}_{\text{test}}$ once. Let $\mathcal{D}_{\setminus\text{test}} \leftarrow \mathcal{D} \setminus \mathcal{D}_{\text{test}}$.
for $r = 1$ **to** R **do** ▷ Outer runs
  Partition $\mathcal{D}_{\setminus\text{test}}$ into disjoint $\mathcal{D}_{\text{train}}^{(r)}$ and $\mathcal{D}_{\text{val}}^{(r)}$, accounting for importance weights.

  (Baseline reference) Query an LLM with $\mathcal{D}$ to select baseline algorithms $\mathcal{A}^{(r)}$.
  For each $a \in \mathcal{A}^{(r)}$ and each $(\widetilde{\mathrm{OAR}}_m, \tilde{w}_m) \in \mathcal{D}_{\text{train}}^{(r)}$, run a for budget B, compute trajectory metrics, and record a per-ensemble, natural-language reference $\mathcal{R}^{(0)}$.

  (Initialization) Obtain a schedule $\pi^{(0)}$ from an LLM using $\mathcal{D}$ and $\mathcal{R}^{(0)}$; set $\pi_{\text{acc}} \leftarrow \pi^{(0)}$.
  for $e = 1$ **to** E **do** ▷ Learning epochs
    $\mathcal{H}^{(e)} \leftarrow \mathcal{R}^{(0)}$; $\pi^{(e,0)} \leftarrow \pi_{\text{acc}}$.
    $S_{\text{train}}^{(e,0)}$, $\{\mathbf{u}^{(m)}(\pi^{(e,0)})\}$, $\mathsf{Traj}^{(e,0)} \leftarrow \textsc{ScoreAndLog}(\pi^{(e,0)}, \mathcal{D}_{\text{train}}^{(r)})$.
    Append to $\mathcal{H}^{(e)}$ a concise record of $\pi^{(e,0)}$, per-ensemble metrics, and $S_{\text{train}}^{(e,0)}$.
    for $t = 1$ **to** T_{rev} **do** ▷ Trajectory-guided revisions
      LLM proposal: $\widehat{\pi}^{(e,t)} \leftarrow \mathsf{Rev}_{\text{LLM}}\big(\pi^{(e,t-1)}, \mathsf{Traj}^{(e,t-1)}, \mathcal{H}^{(e)}\big)$.
      $S_{\text{train}}^{\text{cand}}$, $\{\mathbf{u}^{(m)}(\widehat{\pi}^{(e,t)})\}$, $\mathsf{Traj}^{\text{cand}} \leftarrow \textsc{ScoreAndLog}(\widehat{\pi}^{(e,t)}, \mathcal{D}_{\text{train}}^{(r)})$.
      Append to $\mathcal{H}^{(e)}$ a record of the candidate schedule, per-ensemble metrics, and $S_{\text{train}}^{\text{cand}}$.
      if $S_{\text{train}}^{\text{cand}} > S_{\text{train}}^{(e,t-1)}$ **then** ▷ Improvement
        $\pi^{(e,t)} \leftarrow \widehat{\pi}^{(e,t)}$; $S_{\text{train}}^{(e,t)} \leftarrow S_{\text{train}}^{\text{cand}}$; $\mathsf{Traj}^{(e,t)} \leftarrow \mathsf{Traj}^{\text{cand}}$
      else
        $\pi^{(e,t)} \leftarrow \pi^{(e,t-1)}$; $S_{\text{train}}^{(e,t)} \leftarrow S_{\text{train}}^{(e,t-1)}$; $\mathsf{Traj}^{(e,t)} \leftarrow \mathsf{Traj}^{(e,t-1)}$
      end if
    end for
    Validation: $S_{\text{val}}^{(e)}$, $_$, $_ \leftarrow \textsc{ScoreAndLog}(\pi^{(e,T_{\text{rev}})}, \mathcal{D}_{\text{val}}^{(r)})$; $S_{\text{train}}^{(e)} \leftarrow S_{\text{train}}^{(e,T_{\text{rev}})}$.
    if $\left|S_{\text{train}}^{(e)} - S_{\text{val}}^{(e)}\right| \le \varepsilon_{\text{gap}}$ **then** ▷ Accept epoch
      $\pi_{\text{acc}} \leftarrow \pi^{(e,T_{\text{rev}})}$
    end if
  end for
  $\pi_{\text{final}}^{(r)} \leftarrow \pi_{\text{acc}}$; $S_{\text{test}}^{(r)}$, $_$, $_ \leftarrow \textsc{ScoreAndLog}(\pi_{\text{final}}^{(r)}, \mathcal{D}_{\text{test}})$.
end for
Select $r^\star \in \arg\max_r S_{\text{test}}^{(r)}$ and **return** $\pi^\star = \pi_{\text{final}}^{(r^\star)}$.

demand follows a Poisson process with a weekly seasonal pattern. The warehouse runs a base-stock (order-up-to) policy: at the end of each day it calculates the inventory position (on-hand plus pipeline orders minus any backlog) and orders enough to raise that position to a preset target. Orders arrive after the product's lead time. When serving customers each day, the warehouse first clears any backlog and then serves new demand. Costs are recorded each day as holding plus backorder penalties, but only after an initial warm-up period used to stabilize the system.

Objective. The optimization task is to choose an integer order-up-to target for each product, within practical lower and upper bounds, to minimize the expected average operating cost over the planning horizon. Capacity can be handled either as a hard limit on the sum of all targets or as a soft constraint by adding an increasing penalty (for example, quadratic) when the total target exceeds a specified capacity. Under the chosen targets, the base-stock policy and fixed lead times fully determine material flows and, in turn, the expected costs under the seasonal, stochastic demand.

Baseline SO algorithms. We benchmark three BO variants and two population-based heuristics on the common bounded design space for the base-stock vector $\mathbf{S}$. BO employs a Gaussian process

Algorithm 3 ScoreAndLog: evaluate a schedule and return score, per-ensemble metrics, and trajectories

function SCOREANDLOG(π, $\mathcal{D}'$) $\triangleright\ \mathcal{D}' \subseteq \mathcal{D}$
  for all $(\widetilde{\mathrm{OAR}}_m,\ \tilde{w}_m) \in \mathcal{D}'$ **do**
    Run π on $\widetilde{\mathrm{OAR}}_m$ for budget B; log $\mathcal{T}^{(m)}(\pi) = \{(x_t^{(m)}, y_t^{(m)})\}_{t=1}^{B}$.
    Compute trajectory metrics $\mathbf{u}^{(m)}(\pi) \in [0,1]^P$ (Sec. A.2.2); set $s_m \leftarrow \sum_{p=1}^{P} \lambda_p\, u_p^{(m)}(\pi)$.
  end for
  $S\,(\pi \mid \mathcal{D}') \leftarrow \dfrac{\sum_{(\widetilde{\mathrm{OAR}}_m,\tilde{w}_m)\in\mathcal{D}'} \tilde{w}_m\, s_m}{\sum_{(\widetilde{\mathrm{OAR}}_m,\tilde{w}_m)\in\mathcal{D}'} \tilde{w}_m}$ $\triangleright$ Calculate the score
  return S, $\{\mathbf{u}^{(m)}(\pi)\}_{m\in\mathcal{D}'}$, and $\{\mathcal{T}^{(m)}(\pi)\}_{m\in\mathcal{D}'}$
end function

Table 1: Average cost (lower is better) with one–standard-deviation error across 5 runs; each method uses a budget of 100 evaluations.

Method	Mean $\pm$ Std.	Notes
Single algorithms		
BO-EI	28.27 ± 1.16	
BO-UCB	28.50 ± 0.89	
BO-PI	28.20 ± 1.22	Best single
GA	28.68 ± 1.79	
PSO	43.97 ± 19.90	Highest variance
Hybrid schedules		
BO-EI(50) $\rightarrow$ GA(50)	26.52 ± 0.85	Best overall
BO-UCB(25) $\rightarrow$ PSO(25) $\rightarrow$ BO-PI(50)	27.44 ± 1.34	
GA(20) $\rightarrow$ BO-EI(40) $\rightarrow$ BO-PI(40)	27.58 ± 1.21	

surrogate using a Matérn kernel. We employ (i) expected improvement (EI), (ii) upper confidence bound (UCB), and (iii) probability of improvement (PI) as the acquisition functions. Regarding the heuristics algorithm, the Genetic Algorithm (GA) uses a population of 10 with tournament selection, uniform crossover (rate 0.7) and per-coordinate Gaussian mutation (rate 0.2; step size 0.1 of the variable range); all offspring are clipped to the bounds. The Particle Swarm Optimizer (PSO) maintains 5 particles, asynchronous one-particle updates, and bound clipping. All baselines consume the same evaluation budget and serve both as standalone solvers and as components in hybrid schedules.

Experimental results. All methods were evaluated under budgets of 100 evaluations across 5 independent runs. We exhibit the results including all 5 baseline SO algorithms and the top 3 proposed schedule of algorithms (see Table 1). Among single baseline SO algorithms, **BO-PI** achieved the lowest mean cost (28.20 ± 1.22), narrowly ahead of **BO-EI** and **BO-UCB**; **GA** was moderately worse on average, and **PSO** exhibited the largest variance and the worst mean under this budget.

The schedule of hybrid SO algorithms improved both performance (lower optimal costs) and robustness (lower variance of the optimal costs): the schedule **BO-EI(50)$\rightarrow$GA(50)** delivered the best overall performance with mean cost 26.52 ± 0.85, a **6.0%** reduction relative to the best single method. Qualitatively, the schedules showed faster early decrease in cost and tighter dispersion by exploiting the information from the previous iterations: evaluations accumulated by the first stage seed the second (e.g., GA's population inherits high-quality designs), combining BO's model-based search with GA's refinement. Overall, BO variants are strong standalone algorithms for this problem class, while hybrid schedules yield lower final cost and reduced variability under fixed evaluation budgets. We also include the optimization trajectories (best-so-far plots) of the experiments in Figure 2.

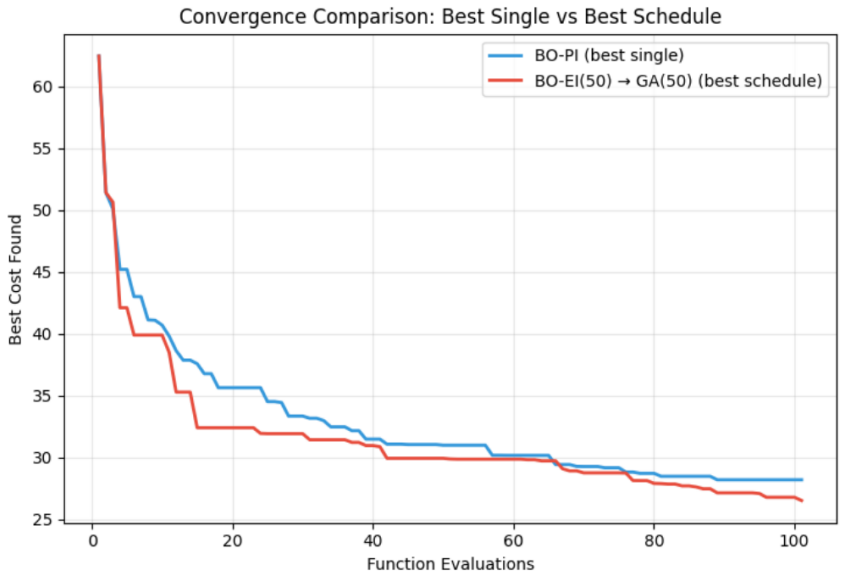

Figure 2: The best-so-far (minimal) values of the best baseline (BO-PI) and the best schedule (BO-EI(50) → GA(50)).

B.2 Multi-Server Queuing Network and Resource–Routing Optimization

We consider a system in service operations. More specifically, the system is a multi-server queuing network with several service stations operating in discrete time. Customers arrive according to a Poisson process and are initially admitted to an entry station; upon service completion, a customer either exits the system or, with a prescribed probability, is routed to another station according to a row-stochastic routing matrix. Each station is staffed by an integer number of parallel servers drawn from a fixed system-wide pool, and its effective service rate can be boosted via a continuous "service-rate multiplier." Unserved customers wait in first-come queues. At each period, we accrue (after a warm-up phase) holding cost proportional to total queue length, operating cost per active server, and resource cost for service-rate enhancements; additional penalties discourage excessive congestion and load imbalance across stations.

Objective. The optimization task is to choose (i) an integer server allocation at each station subject to a total-server budget, (ii) continuous service-rate multipliers within prescribed bounds, and (iii) routing probabilities whose rows normalize to one, so as to minimize the expected average operating cost over the horizon. Under any candidate configuration, the induced stochastic dynamics (arrivals, services, and routing) fully determine customer flows and thus the resulting holding, server, resource, and penalty costs.

The experimental results are summarized in Figure 3, which deliver similar results as in Sec. B.1, and therefore support the superiority of SO schedules (as well as SOCRATES) over standalone algorithms.

C Related Work & Comparison

We discuss some related work of our proposed SOCRATES, and compare our method with existing literature.

C.1 Digital Twins

Our method is connected to and benefits from digital twins (DTs). DT has emerged as a powerful paradigm, creating high-fidelity virtual replicas of physical assets, processes, and systems to enable advanced simulation, prediction, and optimization. By establishing a dynamic, data-driven virtual replica of a real-world system, a DT serves as a sophisticated platform for applying operations research

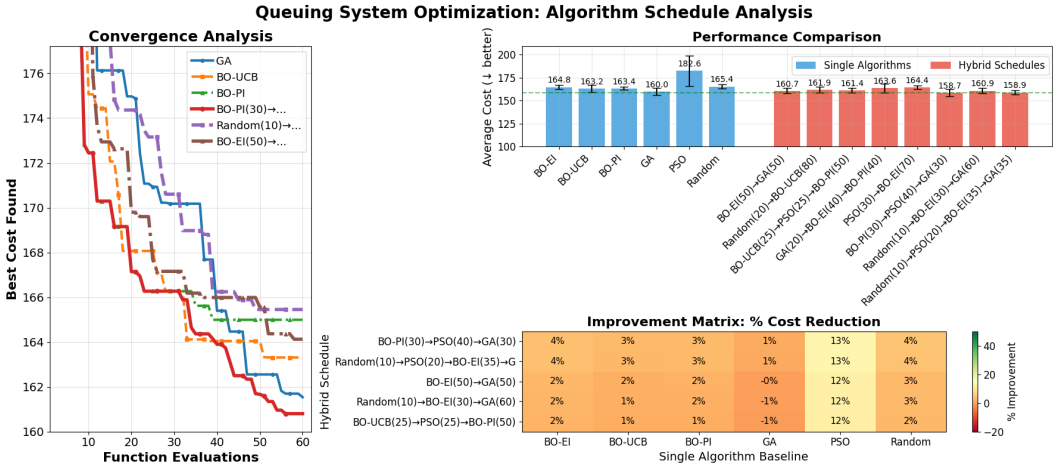

Figure 3: The experiments for the multi-server queue.

(OR) methodologies to understand past events, predict future outcomes, and prescribe actions to solve complex problems before they occur [6]. This capability is particularly transformative for managing large-scale stochastic systems, with a prominent application being the Supply Chain Digital Twin (SCDT) [37, 31]. An SCDT models the supply network as a spatio-temporal dynamic system, using real-time data to enable unprecedented visibility and agility. This allows for the application of OR techniques to pressing supply chain challenges, such as using high-fidelity simulations for resilience testing and risk management, and employing real-time optimization to dynamically control inventory, logistics, and production in response to disruptions.

To better distinguish our proposed digital replicas from the current implementation of DTs, we refer to them as Operational AI Replicas (OARs), emphasizing that they are purpose-built for operational evaluations (i.e., evaluating and revising optimization algorithms) and are constructed using AI techniques. There are mainly three differences between conventional DTs and our OARs:

- **Purpose-Built for Algorithm Development:** OARs are specifically designed as an "algorithm workbench" for the offline development, evaluation, and revision of optimization algorithms. In contrast, the primary focus of conventional DTs is often on creating a high-fidelity, real-time replica of a physical system for live monitoring and control.
- **Automated and Cost-Effective Construction:** Our OARs are built through a highly automated process using LLMs and an EM-type learning procedure, which is significantly less expensive than traditional methods. This differs from many DTs that require extensive, resource-intensive efforts from human experts to manually model physical systems.
- **Scalability and Computational Efficiency:** By design, OARs are computationally efficient and scalable, which is essential for their task of running numerous simulations to evaluate algorithms. This is a key advantage, as many high-fidelity DTs can be resource-intensive, demanding substantial computational power to maintain real-time synchronization with their physical counterparts.

Compared to DT, our proposed OARs are more analogous to digital cousins (DCs) proposed in [11]. The defining characteristic of a DC is the deliberate relaxation of the "exact replica" requirement that defines a DT. A DC does not aim to model every detail of its real-world counterpart. Instead, it focuses on preserving higher-level, functionally important properties. The goal is not perfect imitation but the creation of a distribution of similar environments to train more robust and generalizable policies. While OARs are conceptually similar to DCs, the primary difference lies in the domain of abstraction: DCs are geometric and visual abstractions. They are built from an RGB image and a 3D asset library, resulting in a visually and physically interactive simulation. OARs are causal and analytical abstractions. They are built from textual descriptions and I/O data, resulting in computational efficient functions/mappings that compose stochastic systems.

C.2 LLMs for Optimization Modeling

The intersection LLMs and mathematical optimization is a rapidly advancing field aimed at addressing a challenge in real-life application of optimization: the specialized expertise required to translate real-world problems into formal mathematical models. This emerging synergy leverages LLMs not as direct solvers, but as "optimization modeling copilots" that can interpret natural language descriptions, formulate mathematical models, and generate executable code for traditional high-performance solvers. The primary goal is to democratize access to powerful optimization tools, enabling domain experts without formal optimization modeling training to tackle complex decision-making challenges [1].

A core challenge is the translation of ambiguous natural language into the precise structure of a mathematical programming [34]. Early efforts, such as the NL4Opt competition [32], formalized this task by breaking it down into identifying key components (decision variables, objectives, and constraints), and generating a logical representation. To improve reliability, leading frameworks have adopted structured intermediate representations as a crucial bridge between language and solver code. These representations, such as a formal mathematical model in LaTeX, force the LLM to create a clear and verifiable specification before generating code. The accuracy of this formulation is further enhanced by sophisticated prompt engineering techniques, most notably Chain-of-Thought (CoT) prompting [43], which guides the LLM through a step-by-step reasoning process.

As the field has matured, several specialized architectural frameworks have been developed, each with a distinct philosophy. ORLM [39] represents a data-centric approach, utilizing a semi-automated data synthesis engine called OR-Instruct to create vast, diverse datasets for fine-tuning smaller, open-source models. This has enabled models with 7-8 billion parameters to achieve state-of-the-art performance, demonstrating the power of domain-specific training while addressing cost and privacy concerns. In comparison, OptiMUS [2] employs a modular, multi-agent architecture to handle large-scale, complex problems. It decomposes the modeling process into tasks for specialized agents (Formulator, Programmer, Evaluator) and uses a "connection graph" to manage context, thereby overcoming the input length limitations of LLMs. Additionally, LLMOPT [20] is a unified framework focused on improving "optimization generalization." It integrates its universal five-element formulation with multi-stage learning, including model alignment to reduce hallucinations and an automated self-correction loop to refine solutions.

The final stage of the pipeline involves code generation and execution, where the "solver-in-the-loop" paradigm has become critical. These proposed LLM frameworks not only generate Python code for solvers like Gurobi but also execute it, interpret feedback (such as runtime errors), and autonomously debug their own code or even the underlying mathematical model. The efficacy of these systems is measured using specialized benchmarks like NL4OPT [32], MAMO [19], and the industrially-focused IndustryOR [39]. Performance metrics such as solving accuracy and execution rate consistently show that these specialized, fine-tuned frameworks outperform general-purpose models like GPT-4.