On a Foundation Model for Operating Systems

Divyanshu Saxena[*] **Nihal Sharma**[*] **Donghyun Kim**[*] **Rohit Dwivedula**[*] **Jiayi Chen**[*]
Chenxi Yang[*] **Sriram Ravula**[*] **Zichao Hu**[*] **Aditya Akella**[*] **Sebastian Angel**[†] **Joydeep Biswas**[*]
Swarat Chaudhuri[*] **Isil Dillig**[*] **Alex Dimakis**[*] **Daehyeok Kim**[*] **Chris Rossbach**[*] **Gang Wang**[‡]

Abstract

This paper lays down the research agenda for a domain-specific foundation model for operating systems (OSes). Our case for a foundation model revolves around the observations that several OS components such as CPU, memory, and network subsystems are interrelated and that OS traces offer the ideal dataset for a foundation model to grasp the intricacies of diverse OS components and their behavior in varying environments and workloads. We discuss a wide range of possibilities that then arise, from employing foundation models as policy agents to utilizing them as generators and predictors to assist traditional OS control algorithms. Our hope is that this paper spurs further research into OS foundation models and creating the next generation of operating systems for the evolving computing landscape.

1 Introduction

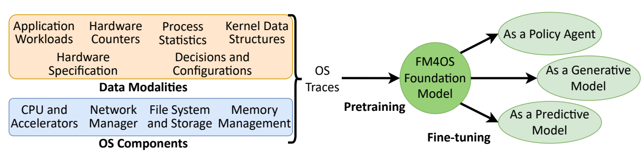

Figure 1: FM4OS: a foundation model for operating systems.

The Operating System (OS) is the central pillar of modern computing systems, overseeing hardware and software resources and enabling applications ranging from assistive robotics to cloud services. OSes serve vital tasks such as scheduling processes; managing CPU, network, and memory resources, and interfacing with devices. To make good decisions, OS policies must account for complex system dynamics such as hardware variances and environment responses, which is challenging for two reasons. First, OSes can be deployed atop a variety of hardware, and amidst diverse workloads and environments. Second, the OS does not have full visibility of the environment (e.g., network performance) or the workload (e.g., application request patterns), making the state space *uncertain*.

Conventional OS policies, reliant on manual algorithms or heuristics, lack adaptability across hardware, environments, and workloads, and often require manual tuning. Recent proposals for using machine learning (ML) models in OS components such as the network manager [1, 23], memory manager [26, 28, 40, 53] and CPU scheduler [9], while being good starting points in bringing data-driven decisions, are still far from ideal as they only optimize for individual components. Furthermore, they neither integrate well together nor generalize well for diverse environments.

Inspired by the recent successes of large unsupervised "foundation" models in NLP and vision tasks, we argue that it is time for the OS to eschew such task-specific solutions in favor of foundation models. Our insight is that OS traces consisting of hardware metrics, system event logs, and application arrivals and requests, can capture all the information on the workings of various OS components and the impact of their decisions on each other. Further, OS traces collected on diverse hardware and application workloads can also capture the intricate relationship between OS decisions, hardware

[*]The University of Texas at Austin
[†]University of Pennsylvania
[‡]University of Illinois at Urbana-Champaign

Machine Learning for Systems Workshop at 37th NeurIPS Conference, 2023, New Orleans, LA, USA.

features, and application workloads. We argue that a foundation model trained on such traces, FM4OS, is plausible and can be used for several downstream tasks (as shown in Figure 1).

2 Background

In this section, we first provide background on OS decision-making: what makes it difficult and why adaptive decisions are needed, and then we give a brief background on foundation models.

Desiderata for operating systems. Operating systems oversee hardware and software resources, including CPU, memory, storage, and network (Table 1 in appendix). In general, OS tasks can be considered sequential decision-making processes where past actions and states of the OS instruct the action at any time. However, these can be very complex because:

- OSes can be deployed on diverse hardware with differing performance profiles. Further, they can run different workloads (e.g., microservice [29] vs. ML workloads [50]) with varying objectives (e.g., prioritize power efficiency for robots vs. optimize performance for cloud servers).
- Access to fine-grained metrics (like the ones shown in Table 1: System State column) from hardware devices or the OS kernel, may be limited.
- *System dynamics*, i.e., the interplay of policies between OS components, also plays a role in decision-making because the actions of one component can impact the future states of other components. Capturing these intricate dynamics is difficult due to the myriad OS policy combinations.

Thus, in the OS setting, there is an *inherent uncertainty and partial observability in the state*.

Existing methods: Prior research has proposed learned and data-driven approaches to address these challenges. Some have employed DNNs to learn policies for specific OS components [2, 9, 22, 30, 56, 57] while others have tackled state uncertainty by modeling OS tasks as MDPs [1, 31, 34, 48, 53]. Additionally, statistical and deep learning methods have been explored to generate realistic workloads [3, 20, 21, 27, 55] that can help inform conventional policies. However, these approaches remain *point solutions* that model individual OS components, leading to a diverse bag of policies, operating independently of others. Consequently, they fall short in optimizing end-to-end OS performance and decision-making. *Ideally, if we could learn how an OS task is impacted by other OS components, application workloads, and hardware specifications, we can devise methods to optimize OS decisions for desired objectives.* These existing approaches also struggle with generalization beyond their training distribution, as shown in prior research [17, 41]. *Therefore, we need techniques that generalize well to unseen inputs.*

Foundation models. This is a catch-all term for ML models trained on a large and diverse dataset to understand the general structure of the data and then fine-tuned (with much less data) for specific tasks. While they have been touted as useful in myriad settings [4, 54], these models have shown tremendous empirical success in sequence modeling problems in natural language [5, 10, 15], finance [51], computer vision [38], biomedical imaging [44] and climate modeling [32] to name a few. At the core of these successes is efficient use of the transformer architecture [45], which learns long-term (spatial and/or) temporal correlations between input sequences, and the principles of transfer learning [42], that enable learning for different tasks, domains, and modalities.

Several OS tasks also fall into this broad category of sequential modeling with the important caveat that the cadence with which decisions are made and the amount (time) and explicit form (states) of past observations vary widely between tasks (see Table 1 in Appendix B).

3 FM4OS: A Foundation Model for the OS

We propose the development of FM4OS, a foundation model that understands the "natural behavior" of the OS and can be fine-tuned for several classes of downstream tasks - all of which either replace or aid the existing policies in the OS. We begin by describing the data sources available that can be used to train such a model.

Data Sources. Today's OSes, along with associated monitoring and data collection infrastructures, provide data in several forms (as shown in Figure 1), including *logs from OS components, hardware metrics, and application workloads*. We elaborate on these sources in Appendix A.

We will use the term "OS trace" to refer to the union of the data corresponding to a single machine drawn from the sources above, represented as a single (time-annotated) sequence. Such traces can be collected from systems with varying hardware specs (CPU, Cache, RAM, NIC, file system, etc.) and under various deployments (cloud, robots, and edge). Below, we describe two OS tasks that operate on different parts of the system, both of which can be trained from the OS traces.

An example use case. Consider the OS scheduling task **SCH** and cache replacement task **CACHE** (descriptions as given in Table 1). As shown in the table, optimal decision-making for the **SCH** task requires process states, process completion times, hardware state, and process arrival workloads while the **CACHE** task requires cache size, state, and cache access workloads. All of these are captured in the OS traces. For **SCH**, the process states and completion times are captured in the logs, process arrival workloads and hardware aspects are captured by the application workloads and environment metadata, respectively. Similarly, for **CACHE**, the cache state is captured by the resource metrics, cache size by environment metadata, and cache access patterns by application workloads.

The OS traces also capture the relationships between the two tasks. For example, the process completion times would depend on the hardware specs of resources other than CPUs, such as the cache. This is because processes may access resources other than CPUs during their execution. For the same reason, OS decisions relating to **CACHE** would also impact the process completion times. Since our OS traces record features that cover the input space of both tasks **SCH** and **CACHE**, they can be used to train one model that can orchestrate both. This model can then be used for several downstream tasks, including: (i) directly making good-quality decisions for the **SCH** and **CACHE** tasks, (ii) predicting the completion time of a newly arrived process, or (iii) generating traces for **CACHE** tasks that can be used to improve conventional data-driven or ML-based algorithms.

When trained on diverse OS traces, the model learns how scheduler and cache behaviors relate to hardware and workloads, enabling generalization to predict program performance on new CPU specifications and cache sizes.

Foundation Model for the OS. The OS traces used above not only encode information for **SCH** and **CACHE** tasks *but also that corresponding to the decision-making in several other OS components*, e.g., I/O prefetching, packet scheduling, congestion control policies, etc. Referring to Table 1, we make two observations to support this. Firstly, several of the OS tasks have shared state space components. For example, both **PREFETCH** and **CACHE** tasks need the cache state, both **PREFETCH** and **PAGE** tasks require process instructions, etc. Secondly, these tasks are not entirely independent, as shown in the above **SCH** task example, where the process completion times (needed for **SCH**) depend on the policies adopted in the **CACHE** task. This inter-dependence of one component on others is a widely seen and natural phenomenon in the operating system. Using OS traces collected across many machines, one can therefore build a foundation model – FM4OS, that knows the 'natural behavior' of the OS. A prospective pretaining regime for FM4OS is discussed in Appendix C.

4 Downstream Tasks for FM4OS

We are now ready to discuss the fine-tuning of FM4OS. We present key downstream tasks and categorize them into three broad use cases: as a policy agent, a generative model, and a predictive model. We discuss these individually below and highlight challenges unique to the OS setting that require novel research on training and using foundation models, in Appendix D.

4.1 FM4OS as a Policy Agent

As discussed in §2, several OS tasks can be modeled as a sequential decision-making process where the state of the OS evolves according to the actions a policy makes. Prior works [1, 9, 22, 34] have used handcrafted features based on heuristics in order to model complex system dynamics.

The key challenge for any solution addressing multiple tasks in the OS is the diversity in state and action spaces of tasks and the different lengths of temporal history deemed relevant for each task (see Table 1). Foundation models have been shown to solve precisely this issue of varying lengths of temporal history due to their ability to summarize inputs of arbitrary lengths in a common representation space. Further, they have also shown evidence of being capable of handling multi-modal input data [43], which suits the various forms of information captured in OS traces (see §3). By engineering the size of these representations for pre-training and specifying the objective during fine-tuning, we expect that FM4OS can be used to suggest optimal actions.

Making low-level decisions: By pre-training FM4OS over OS traces, we expect it to understand the semantic space for OS decision-making. Then, we can use FM4OS to take low-level actions for OS tasks, such as setting the congestion window for the **CC** task and choosing processes for the **SCH** task. Fine-tuning to make these decisions requires historical traces labeled with optimal actions.

Policy selection: Current inference times for transformer-based models do not match the pace at which some OS tasks require actions (every few ns). Accelerating inference [13, 52], especially for operation in the OS [19] is an ongoing research area. In the meantime, FM4OS can address the relatively simpler task of selecting from existing policies (over longer time frames) instead of specifying actions explicitly. For each task, there exist policies optimized for specific environments and workloads. For instance, for the **CACHE** task, Least Recently Used (LRU) policy is favored when access patterns follow locality trends, while Least Frequently Used (LFU) policy is more suitable for random accesses with consistent popular requests [37, 49].

4.2 FM4OS as a Generative Model

Content generated by ML models offers new opportunities for OSes, similar to benefits observed from using generative models in other domains [3, 38]. Synthetically generated data can help add diversity to existing training data used by data-driven solutions, help with the availability and sharing of proprietary or confidential data, and testing models under settings that occur infrequently in practice.

Generating traces: The lack of (diverse) training data is a major hurdle in most data-driven and learned approaches for OS tasks. Even if such data were available, storing and maintaining such a large corpus of data collected under different hardware configurations and workloads be challenging. For example, for the **CACHE** task, traces are needed for different memory specifications and for different types of workloads (small objects, large objects, mixed sizes, etc.). By training FM4OS using auto-regressive tasks like Next Token Prediction, we could train the model to learn to generate OS traces that can be used in a variety of ways.

Fine-tuning it with specially designed prompts could lead to traces that adhere to specific constraints (e.g., setting hardware configurations, limiting network bandwidth, etc). These can be used to supplement the training data collected on specific configurations. Further, the foundation model can also be fine-tuned to obfuscate confidential information from the traces while keeping the important relationships of the traces intact (prior works [27, 55] show feasibility of such obfuscation in network traces). Another opportunity that we identify here is that FM4OS can also be used to generate *pathological corner cases*. Specifically, we posit that by appropriately querying the foundation model, we can use it to generate *pathological corner cases* that would have been otherwise difficult to get.

4.3 FM4OS as a Predictive Model

Foundation models have been shown to exhibit good performance on downstream prediction tasks [5, 32]. In the OS setting, we can use FM4OS as an encoder of the state, and then use linear probing to predict various things about the system's response, future utilization. This can lead to efficient placement, scheduling, performance, and anomaly detection.

System response prediction: Understanding how the environment of the OS evolves with application-level decisions made by the OS are crucial to improve decision quality. For example, predicting the time to completion of a process would allow the kernel to reorder its CPU work queue based on completion times leading to an optimal schedule for minimum waiting time of jobs. Since FM4OS is pre-trained to understand precisely the needed semantic relationships between OS subsystems, it can be used to closely predict system responses.

Application behavior prediction: Predicting the behavior of an application can help the OS prepare in advance for additional resources the application might need and minimize competition for shared hardware. For example, if the OS can predict that an application's execution will be memory-intensive in the near future based on its recent memory allocation calls and nature of inputs received, it can both provision more memory for the application and avoid scheduling another memory-intensive application on the same node.

Anomaly Detection: Using the state encoding of the OS or any of its components, and given a trace, one can ask if the current state is normal, or if there is some anomaly or failure issue. Such predictions can be used to identify and kill anomalous applications, thereby improving the security of the OS kernel.

5 Summary

In conclusion, we argue that the OS decision-making tasks provide a rich arena for a domain-specific foundation model to be built for the OS. We discuss the shortcomings of existing methods of data-driven decision-making and posit that rich OS traces can provide the necessary data to train such a foundation model, FM4OS, which can understand the 'natural behavior' of the OS. We then provide a systematic analysis of the various ways in which FM4OS can be used and the various key challenges that remain open research questions.

References

[1] Soheil Abbasloo, Chen-Yu Yen, and H. Jonathan Chao. Classic meets modern: A pragmatic learning-based congestion control for the internet. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 632–647, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379557. doi: 10.1145/3387514.3405892. URL https://doi.org/10.1145/3387514.3405892.

[2] Ibrahim Umit Akgun, Ali Selman Aydin, Aadil Shaikh, Lukas Velikov, and Erez Zadok. A machine learning framework to improve storage system performance. In *Proceedings of the 13th ACM Workshop on Hot Topics in Storage and File Systems*, pages 94–102, 2021. URL https://dl.acm.org/doi/10.1145/3465332.3470875.

[3] Shane Bergsma, Timothy Zeyl, Arik Senderovich, and J. Christopher Beck. Generating complex, realistic cloud workloads using recurrent neural networks. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, page 376–391, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450387095. doi: 10.1145/3477132.3483590. URL https://doi.org/10.1145/3477132.3483590.

[4] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models, 2022. URL https://arxiv.org/abs/2108.07258.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[6] Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650. USENIX Association, August 2021. ISBN 978-1-939133-24-3. URL https://www.usenix.org/conference/usenixsecurity21/presentation/carlini-extracting.

[7] Nicolas Carlini, Jamie Hayes, Milad Nasr, Matthew Jagielski, Vikash Sehwag, Florian Tramer, Borja Balle, Daphne Ippolito, and Eric Wallace. Extracting training data from diffusion models. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5253–5270, 2023.

[8] Jiayi Chen, Nihal Sharma, Tarannum Khan, Shu Liu, Brian Chang, Aditya Akella, Sanjay Shakkottai, and Ramesh Sitaraman. Darwin: Flexible learning-based cdn caching. In *Proceedings of the 2023 ACM SIGCOMM 2023 Conference*, 2023. URL https://dl.acm.org/doi/pdf/10.1145/3603269.3604863.

[9] Jingde Chen, Subho S. Banerjee, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. Machine learning for load balancing in the linux kernel. In *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems*, APSys '20, page 67–74, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380690. doi: 10.1145/3409963.3410492. URL https://doi.org/10.1145/3409963.3410492.

[10] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways, 2022. URL https://arxiv.org/abs/2204.02311.

[11] Jürgen Cito, Isil Dillig, Seohyun Kim, Vijayaraghavan Murali, and Satish Chandra. Explaining mispredictions of machine learning models using rule induction. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, page 716–727, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385626. doi: 10.1145/3468264.3468614. URL https://doi.org/10.1145/3468264.3468614.

[12] Jürgen Cito, Isil Dillig, Vijayaraghavan Murali, and Satish Chandra. Counterfactual explanations for models of code. In *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 125–134, 2022. doi: 10.1145/3510457.3513081.

[13] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[15] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: An embodied multimodal language model, 2023. URL https://arxiv.org/abs/2303.03378.

[16] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, page 265–284, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3540327312. doi: 10.1007/11681878_14. URL https://doi.org/10.1007/11681878_14.

[17] Tomer Eliyahu, Yafim Kazak, Guy Katz, and Michael Schapira. Verifying learning-augmented systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 305–318, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383837. doi: 10.1145/3452296.3472936. URL https://doi.org/10.1145/3452296.3472936.

[18] Todd Evans, William L. Barth, James C. Browne, Robert L. DeLeon, Thomas R. Furlani, Steven M. Gallo, Matthew D. Jones, and Abani K. Patra. Comprehensive resource use monitoring for hpc systems with tacc stats. In *2014 First International Workshop on HPC User Support Tools*, pages 13–21, 2014. doi: 10.1109/HUST.2014.7.

[19] Henrique Fingler, Isha Tarte, Hangchen Yu, Ariel Szekely, Bodun Hu, Aditya Akella, and Christopher J. Rossbach. Towards a machine learning-assisted kernel with lake. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS 2023, page 846–861, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450399166. doi: 10.1145/3575693.3575697. URL https://doi.org/10.1145/3575693.3575697.

[20] Jiajun Gong, Wuqi Zhang, Charles Zhang, and Tao Wang. Surakav: Generating realistic traces for a strong website fingerprinting defense. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1558–1573, 2022. doi: 10.1109/SP46214.2022.9833722. URL https://ieeexplore.ieee.org/document/9833722.

[21] Raúl Gracia-Tinedo, Danny Harnik, Dalit Naor, Dmitry Sotnikov, Sivan Toledo, and Aviad Zuck. SDGen: Mimicking datasets for content generation in storage benchmarks. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pages 317–330, Santa Clara, CA, February 2015. USENIX Association. ISBN 978-1-931971-201. URL https://www.usenix.org/conference/fast15/technical-sessions/presentation/gracia-tinedo.

[22] Mingzhe Hao, Levent Toksoz, Nanqinqin Li, Edward Edberg Halim, Henry Hoffmann, and Haryadi S Gunawi. {LinnOS}: Predictability on unpredictable flash storage with a light neural network. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 173–190, 2020. URL https://www.usenix.org/conference/osdi20/presentation/hao.

[23] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*, pages 3050–3059. PMLR, 2019. URL https://proceedings.mlr.press/v97/jay19a.html.

[24] Hwajung Kim and Heon Y. Yeom. Lpr: Learning-based page replacement scheme for scientific applications. In *Proceedings of the 23rd International Middleware Conference Industrial Track*, Middleware Industrial Track '22, page 36–42, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450399173. doi: 10.1145/3564695.3564777. URL https://doi.org/10.1145/3564695.3564777.

[25] Seyeon Kim, Kyungmin Bin, Sangtae Ha, Kyunghan Lee, and Song Chong. Ztt: Learning-based dvfs with zero thermal throttling for mobile devices. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '21, page 41–53, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384438. doi: 10.1145/3458864.3468161. URL https://doi.org/10.1145/3458864.3468161.

[26] Yu Liang, Riwei Pan, Tianyu Ren, Yufei Cui, Rachata Ausavarungnirun, Xianzhang Chen, Changlong Li, Tei-Wei Kuo, and Chun Jason Xue. {CacheSifter}: Sifting cache files for boosted mobile performance and lifetime. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*, pages 445–459, 2022. URL https://www.usenix.org/conference/fast22/presentation/liang.

[27] Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. Using gans for sharing networked time series data: Challenges, initial promise, and open questions. In *Proceedings of the ACM Internet Measurement Conference*, pages 464–483, 2020. URL https://dl.acm.org/doi/10.1145/3419394.3423643.

[28] Evan Liu, Milad Hashemi, Kevin Swersky, Parthasarathy Ranganathan, and Junwhan Ahn. An imitation learning approach for cache replacement. In *International Conference on Machine Learning*, pages 6237–6247. PMLR, 2020. URL https://dl.acm.org/doi/abs/10.5555/3524938.3525517.

[29] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. Characterizing microservice dependency and performance: Alibaba trace analysis. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '21, page 412–426, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450386388. doi: 10.1145/3472883.3487003. URL https://doi.org/10.1145/3472883.3487003.

[30] Martin Maas, David G. Andersen, Michael Isard, Mohammad Mahdi Javanmard, Kathryn S. McKinley, and Colin Raffel. Learning-based memory allocation for c++ server workloads. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 541–556, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371025. doi: 10.1145/3373376.3378525. URL https://doi.org/10.1145/3373376.3378525.

[31] Hongzi Mao, Parimarjan Negi, Akshay Narayan, Hanrui Wang, Jiacheng Yang, Haonan Wang, Ryan Marcus, ravichandra addanki, Mehrdad Khani Shirkoohi, Songtao He, Vikram Nathan, Frank Cangialosi, Shaileshh Venkatakrishnan, Wei-Hung Weng, Song Han, Tim Kraska, and Dr.Mohammad Alizadeh. Park: An open platform for learning-augmented computer systems. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/f69e505b08403ad2298b9f262659929a-Paper.pdf.

[32] Tung Nguyen, Johannes Brandstetter, Ashish Kapoor, Jayesh K. Gupta, and Aditya Grover. Climax: A foundation model for weather and climate. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 25904–25938. PMLR, 2023. URL https://proceedings.mlr.press/v202/nguyen23a.html.

[33] Liang Niu, Shujaat Mirza, Zayd Maradni, and Christina Pöpper. CodexLeaks: Privacy leaks from code generation language models in GitHub copilot. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 2133–2150, Anaheim, CA, August 2023. USENIX Association. ISBN 978-1-939133-37-3. URL https://www.usenix.org/conference/usenixsecurity23/presentation/niu.

[34] Haoran Qiu, Subho S. Banerjee, Saurabh Jha, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. FIRM: An intelligent fine-grained resource management framework for SLO-Oriented microservices. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 805–825. USENIX Association, November 2020. ISBN 978-1-939133-19-9. URL https://www.usenix.org/conference/osdi20/presentation/qiu.

[35] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[36] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. *CoRR*, abs/1602.04938, 2016. URL http://arxiv.org/abs/1602.04938.

[37] Liana V Rodriguez, Farzana Yusuf, Steven Lyons, Eysler Paz, Raju Rangaswami, Jason Liu, Ming Zhao, and Giri Narasimhan. Learning cache replacement with {CACHEUS}. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 341–354, 2021.

[38] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022. URL https://openaccess.thecvf.com/content/CVPR2022/papers/Rombach_High-Resolution_Image_Synthesis_With_Latent_Diffusion_Models_CVPR_2022_paper.pdf.

[39] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. Serverless in the wild: Characterizing and optimizing the serverless workload at

a large cloud provider. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 205–218. USENIX Association, July 2020. ISBN 978-1-939133-14-4. URL https://www.usenix.org/conference/atc20/presentation/shahrad.

[40] Zhenyu Song, Daniel S Berger, Kai Li, Anees Shaikh, Wyatt Lloyd, Soudeh Ghorbani, Changhoon Kim, Aditya Akella, Arvind Krishnamurthy, Emmett Witchel, et al. Learning relaxed belady for content distribution network caching. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 529–544, 2020. URL https://www.usenix.org/conference/nsdi20/presentation/song.

[41] Rohan Taori, Achal Dave, Vaishaal Shankar, Nicholas Carlini, Benjamin Recht, and Ludwig Schmidt. Measuring robustness to natural distribution shifts in image classification, 2020.

[42] Sebastian Thrun and Lorien Pratt, editors. *Learning to Learn*. Kluwer Academic Publishers, USA, 1998. ISBN 0792380479.

[43] Maria Tsimpoukelli, Jacob Menick, Serkan Cabi, S. M. Ali Eslami, Oriol Vinyals, and Felix Hill. Multimodal few-shot learning with frozen language models, 2021.

[44] Tao Tu, Shekoofeh Azizi, Danny Driess, Mike Schaekermann, Mohamed Amin, Pi-Chuan Chang, Andrew Carroll, Chuck Lau, Ryutaro Tanno, Ira Ktena, Basil Mustafa, Aakanksha Chowdhery, Yun Liu, Simon Kornblith, David Fleet, Philip Mansfield, Sushant Prakash, Renee Wong, Sunny Virmani, Christopher Semturs, S Sara Mahdavi, Bradley Green, Ewa Dominowska, Blaise Aguera y Arcas, Joelle Barral, Dale Webster, Greg S. Corrado, Yossi Matias, Karan Singhal, Pete Florence, Alan Karthikesalingam, and Vivek Natarajan. Towards generalist biomedical ai, 2023. URL https://arxiv.org/abs/2307.14334.

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[46] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5045–5054. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/verma18a.html.

[47] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015. URL https://dl.acm.org/doi/10.1145/2741948.2741964.

[48] Giuseppe Vietri, Liana V. Rodriguez, Wendy A. Martinez, Steven Lyons, Jason Liu, Raju Rangaswami, Ming Zhao, and Giri Narasimhan. Driving cache replacement with ML-based LeCaR. In *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*, Boston, MA, July 2018. USENIX Association. URL https://www.usenix.org/conference/hotstorage18/presentation/vietri.

[49] Giuseppe Vietri, Liana V Rodriguez, Wendy A Martinez, Steven Lyons, Jason Liu, Raju Rangaswami, Ming Zhao, and Giri Narasimhan. Driving cache replacement with {ML-based}{LeCaR}. In *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*, 2018.

[50] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 945–960, Renton, WA, April 2022. USENIX Association. ISBN 978-1-939133-27-4. URL https://www.usenix.org/conference/nsdi22/presentation/weng.

[51] Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. Bloomberggpt: A large language model for finance, 2023. URL https://arxiv.org/abs/2303.17564.

[52] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2246–2251, Online, July 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.acl-main.204.

[53] Juncheng Yang, Ziming Mao, Yao Yue, and KV Rashmi. {GL-Cache}: Group-level learning for efficient and high-performance caching. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*, pages 115–134, 2023. URL https://www.usenix.org/conference/fast23/presentation/yang-juncheng.

[54] Sherry Yang, Ofir Nachum, Yilun Du, Jason Wei, Pieter Abbeel, and Dale Schuurmans. Foundation models for decision making: Problems, methods, and opportunities, 2023. URL https://arxiv.org/abs/2303.04129.

[55] Yucheng Yin, Zinan Lin, Minhao Jin, Giulia Fanti, and Vyas Sekar. Practical gan-based synthetic ip header trace generation using netshare. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 458–472, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394208. doi: 10.1145/3544216.3544251. URL https://doi.org/10.1145/3544216.3544251.

[56] Yanqi Zhang, Weizhe Hua, Zhuangzhuang Zhou, G. Edward Suh, and Christina Delimitrou. Sinan: Ml-based and qos-aware resource management for cloud microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, page 167–181, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383172. doi: 10.1145/3445814.3446693. URL https://doi.org/10.1145/3445814.3446693.

[57] Yiying Zhang and Yutong Huang. "learned": Operating systems. 53(1):40–45, jul 2019. ISSN 0163-5980. doi: 10.1145/3352020.3352027. URL https://doi.org/10.1145/3352020.3352027.

A Data Sources for OS Traces

Below we list the various data sources that can be used to train FM4OS:

- *Action logs from OS components:* Kernel logs such as `dmesg` in Linux/MacOS and event logs in Windows, capture kernel debugging data, hardware events (e.g., network link status), and system events like interrupts, process restarts. These logs capture the actions taken by the OS components and the system state used by OS tasks (System state and Actions columns in Table 1).
- *Resource metrics and hardware counters:* Hardware drivers record several quantities relating to the resource's state at a pre-configured frequency. These include CPU, memory, and disk bandwidth utilization, NIC queue length, and hardware counters.
- *Application workloads:* Workload traces from productions [29, 39, 47], public infrastructures [18] and synthetically generated ones offer detailed application-level information, such as application type, request arrival rates, statistics of resource usage during execution.

B Decision Making Tasks in Operating Systems

Table 1 shows the various components in the OS and a *representative subset* of the tasks for these components. It also shows the relevant system and environment states, action spaces, and the objectives of these tasks. Each task description is also accompanied by an acronym that we use in the paper to refer to the particular task, e.g. **SCH** for the CPU scheduling task.

C Pretraining Methodology for FM4OS

We envision FM4OS to be **pre-trained** using self-supervised methods on a large corpus of OS traces. This pre-trained model would capture temporal relationships in the sequence of inputs it accepts and build an understanding of the system dynamics of the OS. Existing literature (particularly in natural language) has proposed several pre-training tasks that can be used to develop this basic understanding. Notable among these are the Next Token Prediction [35], Masked Language Modeling, Next Sentence Prediction [14]; each with their own pros and cons. While it seems intuitive to employ an auto-regressive model, pre-trained with next token prediction to build FM4OS, the optimal pre-training task is an open and interesting question in itself.

Table 1: Various decision-making components in the OS.

Components		Description and Acronym	System State	Environment Info	Actions	Objectives
CPU	Scheduling	**[SCH]:** Choose next process to run and which CPU to run it on [9]	Process state (niceness, priority, execution time), Hardware state (CPU, RAM spec, etc.)	Arrival pattern and type of processes (computation-heavy vs. I/O-heavy)	Process to core assignment	Job completion time, fairness
	Voltage and frequency scaling	**[DVFS]:** Choosing CPU frequencies dynamically to reduce power consumptions [25]	CPU frequency buckets, Hardware spec of the CPU	Process workloads, and process instructions	Choose CPU frequency	CPU performance, power, temperature
Memory Subsystem	Page Allocation	**[ALLOC]:** What page size to use (e.g., huge pages vs normal pages) and how to allocate memory [30]	Page table size, Hardware spec (amount of memory, type, etc.)	Memory access patterns of running processes	Page Size, Allocation mechanism	Latency of memory accesses
	Page Replacement	**[PAGE]:** Choose a page in the physical memory to replace with another page in virtual memory [24]	Physical memory state, Hardware spec (amount of memory, type, etc.)	Program instructions, historical data of page faults for the processes	Choose the page to replace	Number of page faults
Network Subsystem	Packet Scheduling	**[NETQUEUE]:** Order packets to send/process from the NIC queues	Queuing delays, NIC spec	Application type (video streaming, analytics, etc.)	Packet drop rate	Throughput and delay
	Congestion Control	**[CC]:** Set congestion window, pacing rate for the connection [1, 23]	Network throughput, delay and packet loss; NIC spec	Application type (video streaming, analytics, etc.)	Congestion window, pacing rate	Throughput and delay
Storage Subsystem	I/O scheduling	**[IOSCH]:** Deciding in which order I/O requests should be submitted to storage devices [22]	I/O metadata (block offset, size), queue state, historical I/O latencies	Application type information (e.g., database, file system, etc.)	Order of I/O requests	I/O latencies
	Prefetching	**[PREFETCH]:** Predict which segments of memory to prefetch [2]	Cache size and state, Cache and PCIe spec	Process workloads and process instructions	Choose segment to prefetch	Throughput of future reads
	Cache replacement	**[CACHE]:** Decide whether and which object to replace in the cache with the new object [8, 40, 53]	Cache size and state (occupied, address, last access)	Cache workloads (object sizes, frequency of access, etc.)	Choose a set of objects to evict/admit	Cache hit ratio

D Open Challenges for FM4OS

D.1 Challenges in using FM4OS as a Policy Agent

End-to-end application performance depends on collective decisions made by OS components. Using foundation models as policy agents brings two unique challenges: *composability* of actions from various policy agents and end-to-end *explanability* of their decisions. The former arises because decisions of one policy can affect the future states of other agents (as with the **CACHE** and **SCH** example discussed in §2). Independently fine-tuned components in the OS may result in suboptimal OS-wide decisions, that may affect both individual application and system-wide guarantees (e.g. fairness and starvation-freedom). One possible approach here is to jointly fine-tune components (to ensure concerted decisions) as well as to develop techniques that provide component-wise guarantees (on performance, e.g., bounds on tail request completion times, or correctness, e.g., safety and liveness properties [17]), and *formally guaranteed composability* of these actions to provide global invariants for the entire OS.

Regarding the latter, ideally, we desire human users to understand the OS at some level to audit or debug it. However, learned decisions from a black-box model may easily obscure the understanding of overall behavior. We envision the use of approaches that describe what each learned policy did in a given execution (similar to LIME [36]), what could have happened had a learned policy made a different decision, and also produce human-comprehensible 'summaries' in the form of rules [11, 12], or programs [46] of what the module will do ahead of time.

D.2 Challenges in using FM4OS as a Generative Model

As with any generative model, quantifying the quality of synthetic samples is a key challenge. At the very least, these traces should maintain certain relationships between variables (e.g., total network transmissions should be less than network bandwidth). They must also capture desired properties that are difficult to obtain otherwise, such as 'realism', i.e., a specific sequence of requests in a generated trace can actually arise in practice — this is an avenue for future research. Further, the generated traces should also be diverse to be useful. For example, for a cache replacement algorithm, we would want traces with diverse and realistic combinations of small and large object arrivals to effectively stress-test the algorithm [8]. Another challenge is with leakage and memorization of sensitive data. As shown in previous works [7, 33], carefully designed prompts can extract memorized training data with sensitive information. Thus, integrating techniques such as filtering the memorized data [6] and ideas from say, differential privacy [16], into FM4OS are necessary.