A Data-driven ML Approach for Maximizing Performance in LLM-Adapter Serving

Ferran Agulló[1,2] Joan Oliveras[1,2] Chen Wang[3] Alberto Gutierrez-Torre[1]
Olivier Tardieu[3] Alaa Youssef[3] Jordi Torres[1,2] Josep Ll. Berral[1,2]
[1]Barcelona Supercomputing Center (BSC), Spain
[2]Universitat Politècnica de Catalunya - BarcelonaTech (UPC), Spain
[3]IBM Research, USA
{ferran.agullo, joan.oliveras, alberto.gutierrez, jordi.torres}@bsc.es
josep.ll.berral@upc.edu {chen.wang1, tardieu, asyousse}@ibm.com

Abstract

With the rapid adoption of Large Language Models (LLMs), LLM-adapters have become increasingly common, providing lightweight specialization of large-scale models. Serving hundreds or thousands of these adapters on a single GPU allows request aggregation, increasing throughput, but may also cause request starvation if GPU memory limits are exceeded. To address this issue, this study focuses on determining the joint configuration of concurrent and parallel adapters that maximizes GPU throughput without inducing starvation, given heterogeneous adapter and traffic properties. We propose a data-driven ML approach leveraging interpretable models to tackle this caching problem and introduce the first Digital Twin capable of reproducing an LLM-adapter serving system, enabling efficient training data generation. Experiments with the vLLM framework and LoRA adapters show that the Digital Twin reproduces throughput within 5.1% of real results, while the ML approach predicts optimal numbers of concurrent and parallel adapters with an error of at most 7.2% under heterogeneous, real-world workloads.

1 Introduction

With the rapid advancement and widespread adoption of Large Language Models (LLMs), the demand for LLM-adapters has grown significantly. While LLMs are large-scale models trained to achieve strong performance across diverse language tasks, adapters specialize this general knowledge to concrete applications through lightweight parameter additions [1, 2, 3, 4]. Their compact size enables serving systems to host hundreds or even thousands of adapters on a single GPU [5, 6], thereby increasing throughput by aggregating requests from many adapters. However, excessive concurrency can reach a critical threshold at which request starvation arises, as the system becomes unable to process incoming requests within available GPU memory limits. This saturation threshold is governed by the interaction of adapter size, request length, and adapter arrival rate, which together determine the degree of concurrency needed to reach the device limits. Request length and adapter size affect the per-request memory usage, while adapter size and arrival rate jointly determine the memory demand per adapter.

Moreover, GPU memory capacity is often insufficient to hold all adapter weights, and only those loaded into device memory can actively process requests. To address this, systems dynamically swap adapters between storage and GPU memory. Adapters resident on the GPU execute requests in parallel with other loaded adapters [7], while swapping enables concurrent execution across those that cannot fit. In some cases, limiting the number of simultaneously loaded adapters—even when memory is sufficient—can improve throughput by leaving more GPU memory available for request

ML For Systems workshop at Neural Information Processing Systems (NeurIPS 2025), San Diego.

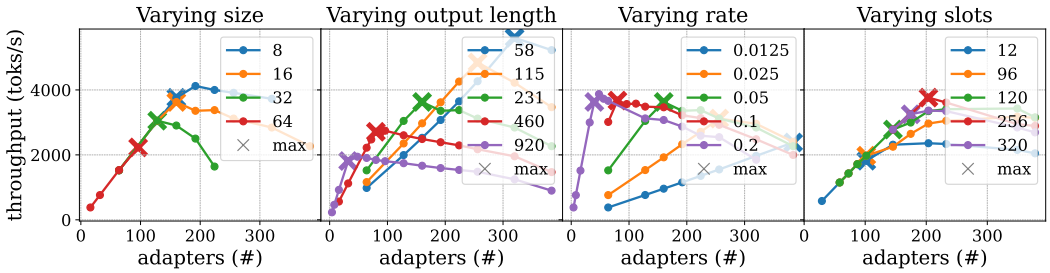


Figure 1: Throughput evolution with the number of concurrent adapters when varying adapter sizes, adapter request rates, request output lengths, and adapter slots. The results are shown for Llama-2-7B [11] and a public adapter [12], using by default rate 0.05 reqs/s, adapter size 8, 250 input tokens, and 231 output tokens. The *max* crosses denote the targeted optimal of the adapter caching problem.

processing. Conversely, loading too few may prevent loaded adapters from fully utilizing the available device memory. Thus, selecting the optimal number of adapters to process in parallel is a key server configuration parameter, that also impacts throughput and the saturation threshold.

Building on the above, this study addresses the following problem: **Given a set of adapters to serve with specified heterogeneous adapter sizes, arrival rates, and request lengths, determine the joint configuration of concurrent and parallel adapters that maximizes a GPU's achievable throughput without inducing request starvation**. We refer to this formulation as the adapter caching problem.

While prior research has investigated the optimization of LLM-adapter serving through kernel-level enhancements [7], memory management techniques [5], and scheduling strategies [8], this specific adapter caching problem remains largely under-explored. The most closely related work, dLoRA [9], employs a greedy, heuristic-driven algorithm to proactively determine how many adapters to serve per GPU in a multi-device system. Advancing on this direction, our work makes two key contributions: (1) we introduce the usage of interpretable ML models to estimate the optimal joint configuration of the adapter caching problem, maximizing single-GPU throughput while preventing request starvation; and (2) we develop the first Digital Twin capable of accurately reproducing the behavior of an LLM-adapter serving system, enabling the timely generation of synthetic data required to train the ML models. Together, these two contributions constitute the so-called data-driven ML approach. We work with the widely adopted vLLM framework [10], in conjunction with LoRA adapters [1]. In vLLM, the maximum number of parallel adapters in GPU is statically defined at server startup as a fixed number of adapter slots; we adopt this terminology from this point onward.

2 Illustrating the adapter caching problem

Figure 1 provides a visual illustration of the adapter caching problem. It depicts throughput (y-axis) as a function of the number of concurrently served adapters (x-axis) across different homogenous scenarios. Each curve exhibits two distinct regimes: initially, throughput increases proportionally with the number of adapters, as the system can accommodate the additional requests. Beyond a certain threshold, however, throughput growth slows or declines, reflecting the system's inability to process excess requests in time, leading to request starvation. The number of concurrent adapters at the transition between these regimes, along with the adapter slots that achieve the best performance (rightmost figure), define the joint configuration targeted in the adapter caching problem for every scenario, which maximizes throughput while avoiding starvation. Practically, we identify this point as the highest measured throughput that remains above 90% of the total incoming token rate. As noted earlier and visible in the figure, this optimal point is highly dependent on the combination of adapter size, request length, and arrival rate, with even greater variation expected in scenarios with heterogeneous characteristics, which represents the real-world behaviour tested in this study.

3 Data-driven ML method

Rather than relying on heuristics, we directly employ ML models to solve the adapter caching problem. We evaluate fast, lightweight, and interpretable models that enable rapid, low-resource predictions suitable for production deployment, while producing outputs that are easily interpretable and allow extraction of simple decision rules. Specifically, we employ three types of linear models—standard linear regression (LinearRegression), Bayesian Ridge regression (BayesianRidge), and Partial Least Squares Regression (PLSRegression)—, as well as, three types of decision tree-based models: the default Random Forest Regressor (RFRegressor), RuleFitRegressor [13] and FIGSRegressor [14]. All models come from the Python libraries scikit-learn [15] and imodels [16]. The inputs for these models consist on the sizes and rates of the adapters to be served in the given scenario, encoded via the minimum, maximum, mean, and standard deviation of all values. Request lengths are fixed using a cleaned ShareGPT dataset version [17], retaining their heterogeneous input/output lengths. For each model type, we create two instances: one to predict the optimal number of concurrent adapters, and another to predict the optimal number of adapter slots. Ground truth for both outputs is obtained by systematically recreating the scenario defined by the inputs, in a manner analogous to Figure 1, evaluating a representative set of combinations of concurrent adapters and adapter slots, and selecting the one obtaining the highest throughput that does not induce request starvation.

Due to the significant resource and time consumption of LLM serving benchmarking in performing these kind of searches, we introduce the first Digital Twin (DT) of an LLM-adapter serving system, specifically targeting the tested vLLM framework. Unlike a traditional simulator, this Digital Twin emulates the system's state and behavior across key execution steps. Although operating offline, it reproduces scheduling, memory allocation, adapter loading, and model forward pass along a simulated timeline reflecting real-time execution. We use predictive performance models to estimate the latency of each step (expanded in Appendix C). These performance models are constructed from simplifications or modifications of prior work [18, 19, 20] and are formalized in Equation 1 (novelty discussed in Appendix D). For simplicity, the predictive models are specific to each LLM–hardware combination, meaning a separate model must be trained for every combination to be evaluated.

$$\begin{aligned}
&\text{Latency of the backbone model forward pass} = K_4 R_{running} + K_5 \\
&\text{Latency of the adapters forward pass overhead} = K_6 A_{running} + K_7 \\
&\text{Latency of the scheduler} = K_1 R_{running} + K_2 R_{waiting} + K_3 R_{waiting}(G/N) \\
&\text{Latency for loading adapters} = \text{simple dictionary created from benchmark data}
\end{aligned} \tag{1}$$

where all constants K_x are calibrated using benchmarking data with a non-linear least squares fitting (*curve_fit* method of SciPy python package [21]), and where $R_{running}$, $R_{waiting}$, $A_{running}$, G and N denote the number of running requests, waiting requests, parallel adapters, adapter slots, and concurrent adapters, respectively.

4 Results

Setup. We use Llama-3.1-8B-Instruct [22] and Qwen2.5-7B-Instruct [23] with LoRA adapters derived from two HuggingFace adapters [24, 25]. All experiments are conducted on a node equipped with a NVIDIA Hopper H100 (64GB HBM2), 128GB RAM memory, and 20 CPU cores.

4.1 Modelling results

We evaluate the Digital Twin by comparing its simulated scenarios against real-system benchmark results, as summarized in Table 1. We average similarity results under high-rank scenarios—equally considering input adapters of ranks 8, 16, and 32—and medium- to low-rank scenarios, only considering ranks of 8 and 16. For adapter rates, we evaluate high-rate scenarios (0.2, 0.1, 0.05) and low-rate scenarios (0.025, 0.0125, 0.00625). For each scenario, we simulate one hour of serving and evaluate performance metrics across a range of served adapters (8–384) and adapter slots (8–384). Overall, the Digital Twin closely reproduces real-system behavior, particularly for throughput and Inter-Token Latency (ITL), with maximum errors of 5.1% and 9.6%, respectively. Time To First Token (TTFT) exhibits a higher average error of 17.9%. In terms of execution efficiency, the DT achieves up to a 90× speedup relative to the full one-hour being simulated, though there remains substantial room for

further improvement. Resource consumption is minimal: the DT runs without a GPU, utilizes at most a single CPU core, and requires approximately 200MB of RAM (expanded results in Appendix E).

	Digital Twin						ML model			
Model	Throu. (%)	ITL (%)	TTFT (%)	Time (s)	CPU (%)	Mem (%)	Type	Con. Adap. (%)	Adap. Slots (%)	Time (ms)
Llama	4.2	9.6	17.4	39.5	90.1	210	Linear	40.6	17.4	0.04
							Tree	0.1	6.7	0.13
							*Tree***	3.7	10.9	0.10
Qwen	5.1	9.1	17.9	41.4	90.7	211	Linear	38.6	23.6	0.03
							Tree	1.0	7.2	0.15
							*Tree***	1.0	12.4	0.09

Table 1: Final results comparing Digital Twin and ML models against benchmark values. Similarity is measured using the SMAPE metric (lower values indicate closer alignment). Execution time is reported for both approaches, and resource usage is also provided for the DT results. For the ML models, we report the best results from both Linear-based and Tree-based model types (expanded in Appendix F), along with results from the interpretability solution under *Tree***.

4.2 Predicting optimal joint configuration

We evaluate the ML model in identifying the joint configuration of the adapter caching problem. The evaluation focuses on scenarios with more highly heterogeneous adapter rates and sizes than the previous section, reflecting real-world conditions. Specifically, for adapter rates, we consider input adapters with rates from all combinations of three values drawn from the set [3.2, 1.6, 0.8, 0.4, 0.1, 0.05, 0.025, 0.0125, 0.00625, 0.003125]. Similarly, for adapter sizes, all possible three values are considered from the set [8, 16, 32]. For each size-rate mix, we evaluate all combinations from a range of concurrent adapter counts (8–384) and adapter slot counts (8–384) to establish the ground truth for these two ML outputs. Of these combinations, 99% are simulated using the DT to generate the training and validation dataset (87,198 runs), while only the remaining 1% is reserved for testing with the real system (882 runs)—given the resource- and time-intensive nature of LLM benchmarking. During training, we employ 5-fold cross-validation and perform hyperparameter optimization using the HalvingGridSearchCV method from scikit-learn [15]. The right portion of Table 1 summarizes the results on the testing set. Linear models fail to capture the complexity of the task, whereas tree-based models generate predictions closely aligned with actual values—achieving a maximum average error of 1.0% for the number of concurrent adapters in both models. Prediction error for adapter slots remains higher, with a maximum of 7.2%, suggesting room for further improvement. With an average inference time of at most 0.15 ms across all tree-based models, they are well-suited for production deployment. Additionally, for both ML outputs, we compress the best-performing tree-based model into a single, shallow tree that can be represented as fewer than 30 highly interpretable rules of the form "condition 1 AND condition 2 ... $\rightarrow$ RESULT". Despite this reduction, the tree maintains competitive performance (Table 1, row *Tree***) and provides a highly interpretable decision-making framework suitable for production deployment (expanded in Appendix G).

5 Discussion and Conclusion

We have presented a data-driven ML approach for determining the joint configuration of concurrent and parallel adapters that maximizes throughput while avoiding request starvation in single-GPU LLM-adapter serving systems. As shown in Table 1, the predicted configurations closely match the real benchmarked optimum, even in heterogeneous scenarios. Notably, simplified and highly interpretable versions of the ML models also achieve competitive performance. Supporting this approach, we developed the first Digital Twin capable of accurately replicating an LLM-adapter serving system, with potential applications beyond generating training data for this work.

Limitations: The key limitation of this study is that the proposed approach, as well as the conducted experiments, targeted only single-GPU configurations. Future work will explore multi-device and multi-node extensions to evaluate performance in realistic production settings and allow comparison with methods such as dLoRA [9]. In addition, because the approach predicts only the number of concurrent and parallel adapters rather than the allocation of specific adapters, it may perform suboptimally in scenarios with very few adapters that exhibit highly diverse characteristics.

6 Acknowledgments

This work has been partially financed by the EU-HORIZON MSCA programme under grant agreement EU-HORIZON MSCA GA.101086248. Also, it has been partially financed by Generalitat de Catalunya (AGAUR) under grant agreement 2021-SGR-00478, by Severo Ochoa Center of Excellence CEX-2021-001148-S-20-3, and by the Spanish Ministry of Science (MICINN), the Research State Agency (AEI) and European Regional Development Funds (ERDF/FEDER) under grant agreement PID2021-126248OB-I00, MCIN/AEI/10.13039/ 501100011033/ FEDER, UE.

References

[1] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "Lora: Low-rank adaptation of large language models." *ICLR*, vol. 1, no. 2, p. 3, 2022.

[2] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for nlp," in *International conference on machine learning*. PMLR, 2019, pp. 2790–2799.

[3] H. Liu, D. Tam, M. Muqeeth, J. Mohta, T. Huang, M. Bansal, and C. A. Raffel, "Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 1950–1965, 2022.

[4] D. Guo, A. Rush, and Y. Kim, "Parameter-efficient transfer learning with diff pruning," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, C. Zong, F. Xia, W. Li, and R. Navigli, Eds. Online: Association for Computational Linguistics, Aug. 2021, pp. 4884–4896.

[5] Y. Sheng, S. Cao, D. Li, C. Hooper, N. Lee, S. Yang, C. Chou, B. Zhu, L. Zheng, K. Keutzer *et al.*, "Slora: Scalable serving of thousands of lora adapters," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 296–311, 2024.

[6] R. Brüel-Gabrielsson, J. Zhu, O. Bhardwaj, L. Choshen, K. Greenewald, M. Yurochkin, and J. Solomon, "Compress then serve: Serving thousands of lora adapters with little overhead," *arXiv preprint arXiv:2407.00066*, 2024.

[7] L. Chen, Z. Ye, Y. Wu, D. Zhuo, L. Ceze, and A. Krishnamurthy, "Punica: Multi-tenant lora serving," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 1–13, 2024.

[8] N. Iliakopoulou, J. Stojkovic, C. Alverti, T. Xu, H. Franke, and J. Torrellas, "Chameleon: Adaptive caching and scheduling for many-adapter llm inference environments," *arXiv preprint arXiv:2411.17741*, 2024.

[9] B. Wu, R. Zhu, Z. Zhang, P. Sun, X. Liu, and X. Jin, "dLoRA: Dynamically orchestrating requests and adapters for LoRA LLM serving," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024, pp. 911–927.

[10] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 611–626.

[11] H. Touvron *et al.*, "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: https://arxiv.org/abs/2307.09288

[12] yard1, "Sql lora for llama-2-7b," 2024. [Online]. Available: https://huggingface.co/yard1/llama-2-7b-sql-lora-test

[13] J. H. Friedman and B. E. Popescu, "Predictive learning via rule ensembles," *The Annals of Applied Statistics*, vol. 2, no. 3, pp. 916–954, 2008.

[14] Y. S. Tan, C. Singh, K. Nasseri, A. Agarwal, J. Duncan, O. Ronen, M. Epland, A. Kornblith, and B. Yu, "Fast interpretable greedy-tree sums (figs)," *ArXivorg*, 2023.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[16] C. Singh, K. Nasseri, Y. S. Tan, T. Tang, and B. Yu, "imodels: a python package for fitting interpretable models," p. 3192, 2021.

[17] anon8231489123, "Clean sharegpt dataset," 2023. [Online]. Available: https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered

[18] H. Zhang, Y. Tang, A. Khandelwal, and I. Stoica, "SHEPHERD: Serving DNNs in the wild," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 787–808. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/zhang-hong

[19] H. Shen, L. Chen, Y. Jin, L. Zhao, B. Kong, M. Philipose, A. Krishnamurthy, and R. Sundaram, "Nexus: a gpu cluster engine for accelerating dnn-based video analysis," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ser. SOSP '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 322–337.

[20] S. Li, H. Lu, T. Wu, M. Yu, Q. Weng, X. Chen, Y. Shan, B. Yuan, and W. Wang, "Caraserve: Cpu-assisted and rank-aware lora serving for generative llm inference," *arXiv preprint arXiv:2401.11240*, 2024.

[21] P. Virtanen *et al.*, "Scipy 1.0: Fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, 2020.

[22] A. Grattafiori *et al.*, "The llama 3 herd of models," 2024. [Online]. Available: https://arxiv.org/abs/2407.21783

[23] A. Yang *et al.*, "Qwen2.5 technical report," 2025. [Online]. Available: https://arxiv.org/abs/2412.15115

[24] Wengwengwhale, "Finance lora adapter for llama-3.1-8b instruct," 2024. [Online]. Available: https://huggingface.co/Wengwengwhale/llama-3.1-8B-Instruct-Finance-lora-adapter

[25] zjudai, "Medical lora for qwen2.5-7b-instruc," 2025. [Online]. Available: https://huggingface.co/zjudai/flowertune-medical-lora-qwen2.5-7b-instruct

[26] P. G. Recasens, Y. Zhu, C. Wang, E. K. Lee, O. Tardieu, A. Youssef, J. Torres, and J. L. Berral, "Towards pareto optimal throughput in small language model serving," in *Proceedings of the 4th Workshop on Machine Learning and Systems*, 2024, pp. 144–152.

[27] P. G. Recasens, F. Agullo, Y. Zhu, C. Wang, E. K. Lee, O. Tardieu, J. Torres, and J. L. Berral, "Mind the memory gap: Unveiling gpu bottlenecks in large-batch llm inference," in *2025 IEEE 18th International Conference on Cloud Computing (CLOUD)*, 2025, pp. 277–287.

[28] J. Cho, M. Kim, H. Choi, G. Heo, and J. Park, "Llmservingsim: A hw/sw co-simulation infrastructure for llm inference serving at scale," in *2024 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2024, pp. 15–29.

[29] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for Transformer-Based generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 521–538.

A Main LLM-adapter serving overheads

We describe the three main overheads that we encountered when working with adapters, providing more insights into the variations of the targeted optimal configuration. In order to compare the impact of different input/output lengths, we create three synthetic datasets by repeating a single request sampled from the clean ShareGPT dataset at the 25th percentile, mean, and 75th percentile of input/output length: *SmallRequest* (23/27 tokens), *MediumRequest* (250/231), and *LargeRequest* (423/358). We use Llama-2-7B and Llama-2-13B models with LoRA adapters also based on a publicly available adapter [12].

A.1 Increased memory usage

The increased usage of GPU memory for saving adapters' weights reduces the available space for requests' KV cache values, limiting the batch size, and consequently the throughput. In this way, as depicted in left part of Figure 2, as the number of loaded adapters increases, the maximum achievable throughput also decreases in a somehow exponential pattern. This decrease occurs earlier and happens to be more pronounced with larger models and adapter sizes in accordance with their higher memory usage. Figure 2 also shows the corresponding impact in batch size, which decreases linearly with the number of loaded adapters. This linear trend contrasts with the exponential decline observed in throughput, a difference that is linked to a broader characteristic of LLM serving not just adapter serving. As reported in several works, increasing the batch size beyond a certain point leads to diminishing returns in throughput—a phenomenon known as the throughput plateau [26, 27]—which corresponds to the leftmost points of each line in the figure.

Insight. Each loaded adapter affects maximum throughput, depending on model, request length, and adapter size—but this impact fades at large batch sizes due to the throughput plateau.

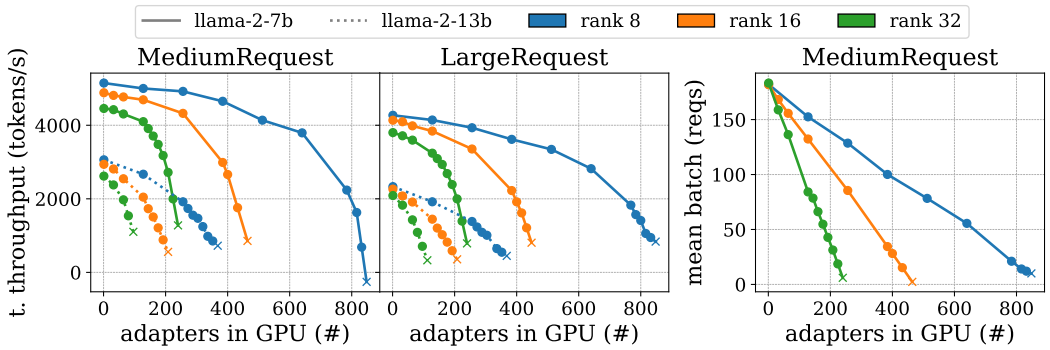


Figure 2: Maximum throughput (left) and batch size (right) evolution as the number of loaded adapters increases, shown for both models, varying adapter sizes/ranks and two datasets. Crosses indicate when no space is left for loading more adapters.

A.2 Increased computational workload

Building upon the analysis by Li et al. [20], Figure 3 illustrates the impact of increasing the number of unique adapters in the batch on both throughput and ITL. Adapter weights introduce overhead to activation computation and GPU cache transfers. The most significant slowdown occurs when moving from zero to one adapter, as this introduces a sequential computation step that cannot be parallelized across requests. Beyond this point, the overhead does not scale in proportion to the number of adapters. While throughput degradation is relatively consistent across datasets, smaller requests can be more affected due to their higher batch sizes, which allow more adapters per batch.

Insight. Serving more unique adapters significantly reduces throughput, bounded by the maximum batch size.

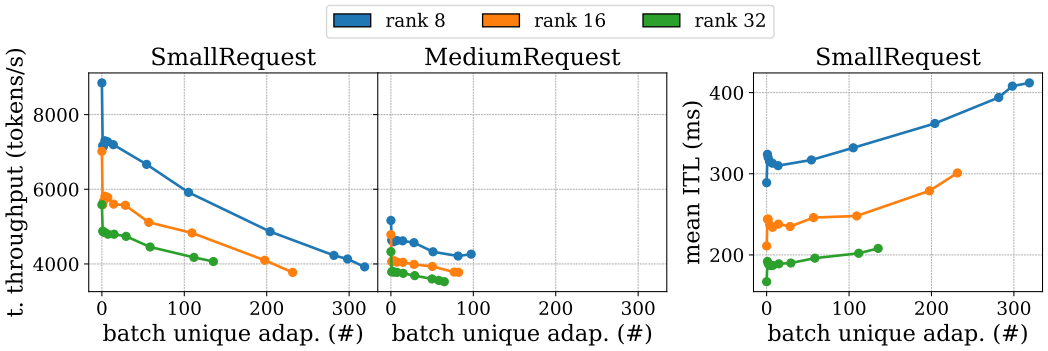


Figure 3: Maximum throughput (left) and ITL (right) as the number of unique adapters in the batch increases, shown for Llama-2-7B, varying adapter sizes/ranks, and two datasets. Lines terminate at the point where the batch size can no longer be increased.

A.3 Loading time

Building upon the analysis by Iliakopoulou et al. [8], Figure 4 presents the loading time relative to request latency, distinguishing between loading from disk or CPU memory. Larger adapters incur greater overhead, and loading from disk is, on average, 70% slower than loading from CPU memory. Furthermore, the request length significantly influences the relative impact: for small requests, loading from CPU introduces a latency overhead of 7–16%, depending on adapter size, whereas for longer requests, this overhead drops below 2%. Since longer requests require more computation time, the fixed cost of loading becomes comparatively smaller.

Insight. Loading overhead is significant only for short requests and can be largely mitigated by preloading adapters into CPU memory.

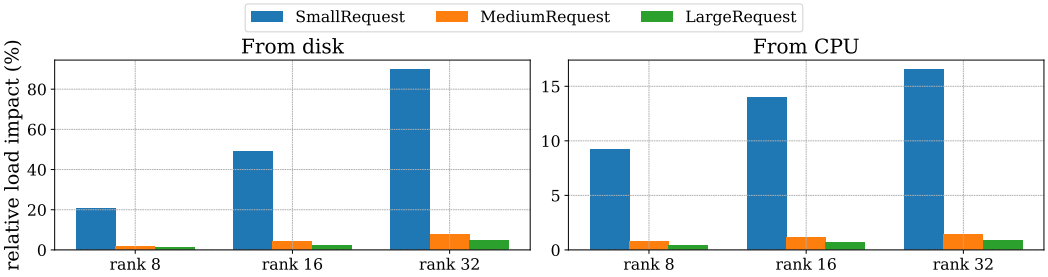


Figure 4: Loading times for varying adapter sizes, shown relative to request latency across the three datasets for Llama-2-7B, and storage type. Request latency is computed as $TPOT * (output_tokens - 1)$ where TPOT is the time per output token.

B S-LoRA case

We include a brief analysis using the S-LoRA framework [5] to demonstrate that our problem is not specific to vLLM and can be applied to other frameworks. Figure 5 presents the optimal configuration of concurrent adapters in S-LoRA across different arrival rates. Notably, the decline in throughput as the rate decreases is remarkably modest compared to vLLM, highlighting the relevance of the S-LoRA design. Nevertheless, we still can perceive a 15-20% decrease in the maximum throughput. Our approach could be handy in identifying these throughput variations and determining the number of adapters at which this maximum is achieved.

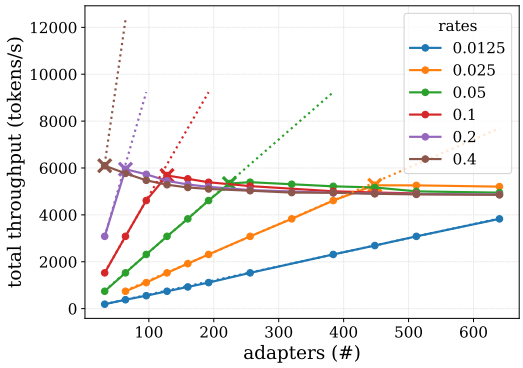


Figure 5: Optimal concurrent adapters when working with S-LoRA (mark with crosses) with varying adapter rates in Llama-2-7B, with rank 32, 250 input tokens and 231 output tokens.

C Digital Twin extended description

As introduced, the proposed Digital Twin (DT) is an offline emulator that replicates an online LLM-adapter serving system that serves requests from multiple adapters of the same backbone model. Specifically, we try to replicate the widely used vLLM [10] framework. As LLMServingSim [28], we mirror the "infinite" loop over the running batch of modern LLM serving systems (online batching) [29], enabling the accurate estimation of key metrics. Figure 6 illustrates the overall behavior of this loop, the different components involved, and their interactions. The DT is build on modular components, each responsible for simulating a specific aspect of the system, that rely on predictive performance models—described in Subsection C.1—to predict the time required on the different tasks. Each iteration of the loop follows the same sequence of actions. First, new request arrivals are collected based on the current simulation time and the workload characteristics, and forwarded to the scheduler. The scheduler manages the running batch, removing finished requests and adding new ones. As vLLM, we replicate a First Come First Served (FCFS) policy with greedy allocation of KV cache and chunked prefill. Once the running batch is updated by the scheduler, it is sent to the adapter cache and model components, which replicate the loading and unloading of adapters with a Least Recently Used (LRU) policy and the model forward pass, respectively.

To run the DT, the expected workload and adapter characteristics—adapter rates and sizes—and key server configuration parameters—mainly the number of adapter slots—are given. In addition, the DT only needs the expected average and standard deviation of the input and output lengths.

C.1 Predictive performance models

Among the five performance models shown in Figure 6, Mem_{max} works an auxiliary estimator that was not included in the main text. It estimates the available GPU memory for storing KV values based on the number of adapter slots and adapter sizes, covering the memory overhead discussed in Appendix A. This estimator is built directly from retrieving the values obtained in that section. The remaining four predictive models predict the latency of each component of the loop, as previously seen in Equation 1. While Lat_{load} is also derived from direct benchmarking of the loading process across various sizes—coming from Appendix A results—, the other three use simple linear models fitted to benchmarking data. Lat_{sched}, representing the scheduler time, is estimated based on the number of running requests ($R_{running}$), waiting requests ($R_{waiting}$), and an interaction term involving $R_{waiting}$ and the ratio of adapter slots (G) to concurrent adapters (N), reflecting the behavior of the original scheduling algorithm. Lat_{model}, the latency of the base model, is estimated solely from the number of running requests, following prior analyses [18, 19]. Finally, $Lat_{adapters}$, corresponding to the computational overhead discussed in Appendix A, is modelled as an overhead and estimated based on the number of parallel adapters ($A_{running}$), in accordance with our results.

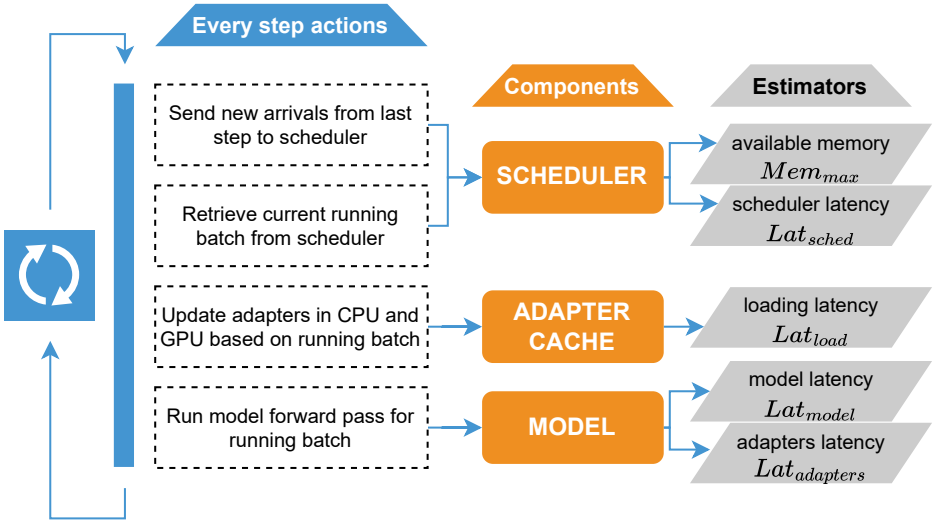


Figure 6: Digital Twin behavior and architecture.

D Novelty in Digital Twin predictive performance models

As outlined in the main text, the predictive performance models in Equation 1 can be viewed as either simplifications or modifications of prior work. Since this characterization is somewhat broad, we provide a more detailed explanation here. For the latency of the backbone LLM, our approach does not introduce much novelty: prior studies have employed linear models with respect to the number of concurrent requests (batch size) for generic AI models [18, 19], which is consistent with our benchmark results. For the latency overhead introduced by adapters, Li et al. [20] estimate it based on the sum or maximum of the adapter ranks present in the batch. However, their formulation did not align with our results, leading us instead to model the latency overhead as a function of the number of parallel adapters, which has a way higher impact than the rank in our results. Finally, regarding scheduler time, to the best of our knowledge we are the first to propose such a model for LLM serving. That said, this formulation is currently limited to vLLM, and further evaluation across other frameworks is required to assess its generality.

E Digital Twin extra results

Figure 7 visually illustrates the differences between the Digital Twin and real system results for one of the evaluated combinations. Consistent with the table results, the DT accurately estimates throughput and ITL, while TTFT shows larger deviations. Additionally, the figure highlights that ITL estimation worses with a higher number of adapter slots, and TTFT becomes less accurate under higher arrival rates.

F ML model extra results

Table 2 expands the results of the right portion of Table 1 for all tested model types.

G ML model interpretability claim

We simplify the best-performing model, the RFRegressor, by reducing it to a single decision tree (parameter *n_estimators*) with a maximum depth of six (parameter *max_depth*) and enforcing a minimum of ten samples per leaf (parameter *min_samples_leaf*). This reduction yields a highly interpretable model that retains competitive performance, as shown in Table 2 under the *Tree*** row. An example of such a tree, trained to predict the number of adapter slots, is presented in

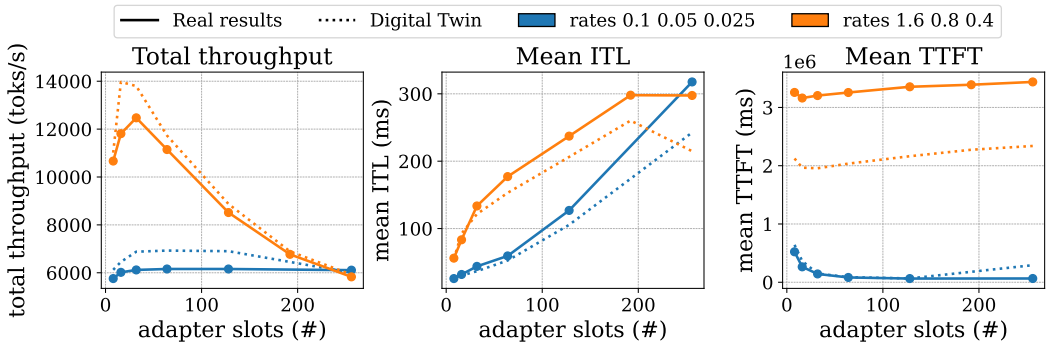
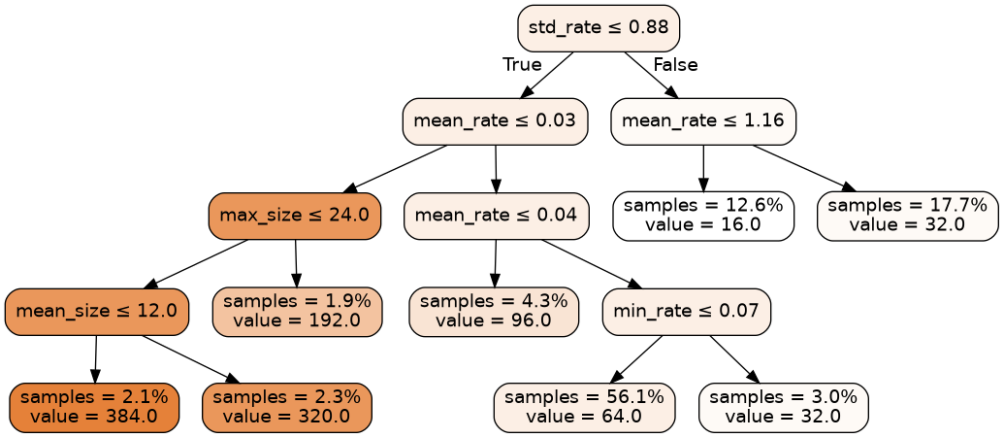


Figure 7: Comparison between DT and real results for throughput, ITL and TTFT for varying adapter slots and rates when serving 256 concurrent adapters of ranks 8 and 16 on Qwen-2.5-7B model.

Model	Estimator	Concurrent adapters (%)	Adapter slots (%)	Time (ms)
Llama-3.1-8B	LinearRegression	40.61	19.53	0.04
	BayesianRidge	40.59	17.54	0.25
	PLSRegression	40.61	17.38	0.04
	RFRegressor	**0.05**	**6.73**	0.13
	RuleFitRegressor	6.36	14.89	1.80
	FIGSRegressor	0.14	14.06	**0.03**
	*Tree***	3.70	10.86	**0.10**
Qwen-2.5-7B	LinearRegression	38.59	23.70	**0.03**
	BayesianRidge	38.57	24.57	**0.03**
	PLSRegression	38.56	23.62	**0.03**
	RFRegressor	**1.01**	**7.15**	0.15
	RuleFitRegressor	1.88	11.76	2.03
	FIGSRegressor	1.11	14.97	**0.03**
	*Tree***	**1.01**	12.41	0.09

Table 2: Expanded results of the ML model for predicting the optimal joint configuration (concurrent adapters and adapter slots). All values represent the SMAPE difference from the benchmark maximum values, except time (reported in milliseconds), which corresponds to the summed inference time across the two output predictions.

Figure 8, which also illustrates the set of simple rules in the form "condition 1 AND condition 2 ... $\rightarrow$ RESULT" that can be extracted from the estimator. Equivalent models were generated for the concurrent adapters output, requiring no more than 30 rules. These rule-based representations provide an interpretable decision source for production systems while offering a lightweight, efficient, and practical solution for real-world deployments.



(a) Simplified RFRegressor

1) $\widetilde{rate} \leq 0.88$ & $\overline{rate} \leq 0.03$ & $max(size) \leq 24.$ & $\overline{size} \leq 12. \rightarrow 384$
2) $\widetilde{rate} \leq 0.88$ & $\overline{rate} \leq 0.03$ & $max(size) \leq 24.$ & $\overline{size} > 12. \rightarrow 320$
3) $\widetilde{rate} \leq 0.88$ & $\overline{rate} \leq 0.03$ & $max(size) > 24. \rightarrow 192$
4) $\widetilde{rate} \leq 0.88$ & $\overline{rate} > 0.03$ & $\overline{rate} \leq 0.04 \rightarrow 96$
5) $\widetilde{rate} \leq 0.88$ & $\overline{rate} > 0.03$ & $\overline{rate} > 0.04$ & $min(rate) \leq 0.08 \rightarrow 64$
6) $\widetilde{rate} \leq 0.88$ & $\overline{rate} > 0.03$ & $\overline{rate} > 0.04$ & $min(rate) > 0.08 \rightarrow 32$
7) $\widetilde{rate} > 0.88$ & $\overline{rate} \leq 1.16 \rightarrow 16$
8) $\widetilde{rate} > 0.88$ & $\overline{rate} > 1.16 \rightarrow 32$

(b) Extracted rules

Figure 8: (Top) Simplified RFRegressor used to predict the number of adapter slots for the Qwen-2.5-7B model. (Bottom) The same model expressed as a set of eight rules,, where $\overline{x}$ and $\widetilde{x}$ denote the mean and standard deviation of x, respectively.