# Data-Efficient Ranking of Recommendation Models

**Anonymous**[1]

[1]Anonymous Institution

**Abstract** Online learning is the cornerstone of applications where data becomes available in sequential order, and models continuously adapt to the shifting data distribution, e.g., in recommendation and advertising systems. Model training for such systems is remarkably expensive due to the massive amounts of data used. This cost multiplies during hyperparameter search, where developers run multiple training configurations. We aim to reduce the cost of this search process and execute it in two stages: **(1)** identifying the most promising configuration(s) and **(2)** training the most promising candidate configuration(s) to the best performance. As stage **(1)** traditionally incurs the dominant cost, we focus our efforts there, developing advanced data reduction and prediction strategies for efficient *identification* of the most promising configuration(s) from a pool of candidate configurations. Note that *identifying* the most promising configuration efficiently is different from *achieving* the best performance efficiently, and provides room for significant efficiency gain albeit with new tools and analyses. Our approach specifically overcomes challenges from online learning's sequential, non-stationary data that are not addressed by conventional "offline" hyperparameter optimization. Overall, we reduce the total hyperparameter search cost by up to 10 times.

## 1 Introduction

Online learning (Hoi et al., 2021) is crucial for applications like recommendation (Naumov et al., 2019) and advertising (McMahan et al., 2013) systems, which continuously train on sequential data to adapt to distribution shifts. Periodically, models are updated via an extensive and costly "hyperparameter search" to find better configurations (e.g., new architectures or hyperparameters). This involves training numerous candidate configurations on fixed historical data that mimics the sequential nature of live traffic. Our goal is to drastically reduce the cost of this search without compromising the quality of the final selected model.

Prior work on efficient hyperparameter search (Domhan et al., 2015; Gao et al., 2022) or data-efficient training (Paul et al., 2021) does not optimally address the unique aspects of our online learning setup described below:

**Opportunity**. Existing work often aims to train *all* configurations more efficiently while preserving their final performance (Jain et al., 2024). Instead, we adopt a two-stage approach: **(1)** efficiently *identify* the top configurations, and then **(2)** fully train *only these* promising candidates. The key insight is that stage (1) only needs to accurately predict the *ranking* of configurations, not their optimal performance. This allows for far more aggressive cost-saving measures in stage (1), which dominates the overall search cost.

**Challenge**. While two-stage search has been explored, prior work focuses on offline learning with stationary data. In contrast, online learning involves a single pass over non-stationary sequential data. This invalidates many existing techniques, like learning curve prediction methods that assume monotonically decreasing loss (Klein et al., 2017; Kadra et al., 2023), or data selection methods that require multiple passes to compute importance scores (Paul et al., 2021; Toneva et al., 2019).

These challenges and opportunities demand a new approach. We develop novel techniques that yield up to a 10x cost reduction on the Criteo pCTR dataset Criteo (2023).

Submitted to AutoML 2025 Non-Archival Track 

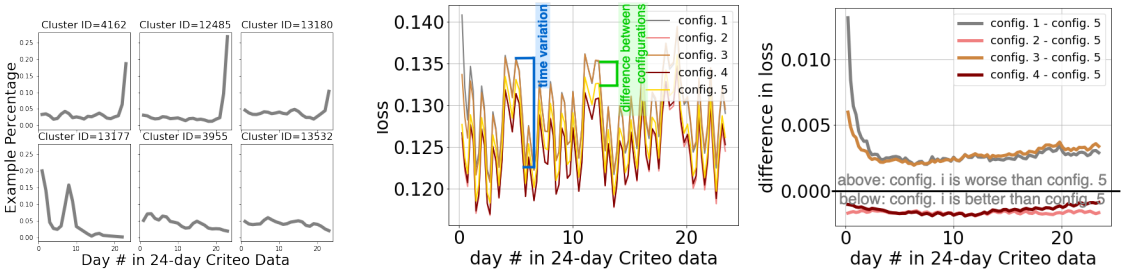


Figure 1: **(left)** Data distribution shifts over time, shown by changing cluster sizes on Criteo data. **(middle)** The loss for different configurations follows a strong time-varying pattern (e.g., blue variance) that dominates the performance difference between configurations (green line). **(right)** Taking the relative loss with respect to a reference configuration removes the common time variation, revealing the true performance differences.

Our contributions can be summarized as follows:

**(1)** We introduce novel **data reduction strategies** (e.g., performance-based stopping, subsampling) for online learning. To enable aggressive reduction, we complement them with **prediction strategies** that estimate full-data performance from data-reduced runs, ensuring we can accurately rank the top configurations.

**(2)** Our prediction strategies enable dynamic resource management. We use a performance-based stopping mechanism to terminate non-promising runs early, reallocating resources to more promising candidates.

**(3)** Our key contribution is the **stratification of predictions via data slicing**. By predicting performance on distinct data clusters and aggregating these sliced predictions, we turn a hard overall prediction problem into a set of more tractable sub-problems, boosting effectiveness.

## 2 Problem Setup and Preliminaries

Let a model be $f(x, \theta, \omega)$, where $x \in \mathcal{X}$ is input, $y \in \mathcal{Y}$ is the label, $\theta \in \Theta$ are trainable parameters, and $\omega \in \Omega$ is a configuration (e.g., architecture, hyperparameters). In online learning, we process a sequence of examples $(x_1, y_1), \ldots, (x_T, y_T)$. The model parameters $\theta_t$ at time $t$ depend only on data seen up to $t$. We evaluate performance using a metric $m$ (e.g., log loss; lower is better), averaged over a final evaluation window $\mathcal{W}_{\text{eval}} = [T - \Delta, T]$. We denote this final performance for a configuration $\omega$ as $\overline{m}(\omega)$.

**Our Goal: Ranking Configurations.** Our goal is to efficiently find a ranking $r$ of configurations that is close to the true ranking $r^*$, which is based on $\overline{m}(\omega)$ values from full-data training. We measure efficiency by the cost ratio $C = \frac{\text{cost of obtaining } r}{\text{cost of obtaining } r^*}$.

**Ranking Metric.** While there are several metrics suitable for ranking (see Appendix C), we focus on a metric that measures how well the best configurations are identified. To this end, our primary ranking metric is **Regret@k**, which measures the average performance loss from choosing the top-$k$ configurations from our predicted ranking $r$ instead of the true top-$k$ from $r^*$: $\texttt{regret}\,@k(r) = \frac{1}{k}\sum_{i=1}^{k} \max\left(0, \overline{m}(r(i)) - \overline{m}(r^*(i))\right)$. A lower $\texttt{regret}\,@k$ indicates a better ranking of top configurations.

### 2.1 Time Variation in Online Training Data

Online learning data is non-stationary. Figure 1(left) shows how the distribution of data clusters in the Criteo dataset shifts over the 24-day period. This distribution shift causes large fluctuations in the loss during training, as shown in Figure 1(middle). This time variation in a single model's loss is often much larger than the actual performance difference between two distinct configurations, making the ranking task difficult. However, the time variation pattern is remarkably consistent across all configurations, regardless of their architecture or hyperparameters. This suggests the

pattern is inherent to the data's "hardness" over time. We can exploit this by analyzing the *relative* performance between configurations. As seen in Figure 1(right), plotting the loss difference against a reference model cancels out the common time-varying noise, making the true performance differences much clearer. Our methods build upon this insight.

## 3 Method

In this section, we introduce our data reduction and prediction strategies.

### 3.1 Data Reduction Strategies

**One-Shot Early Stopping.** The simplest strategy is to stop all training runs at an early time $t_{\text{stop}} < T$ and rank configurations based on their performance up to that point. The relative cost is $C \approx \frac{t_{\text{stop}}}{T}$.

**Performance-Based Stopping.** Since we only care about the top configurations, we can stop non-promising runs even earlier and let promising ones continue. This adaptive strategy allocates resources more effectively. At predefined checkpoints $t_{\text{stop}} \in \mathcal{T}_{\text{stop}}$, we predict the final performance of all active configurations, permanently stop a fraction $\rho$ of the worst-performing ones, and continue training the rest. This is a generalization of successive halving (Kumar et al., 2018). The full procedure is in Algorithm 1 in the appendix. More details can be found in Appendices B and C.4.

**Data Sub-Sampling.** Orthogonal to stopping strategies, we can reduce cost by training on a subset of data, for example by uniform sub-sampling or by sub-sampling the majority class (e.g., negative labels).

### 3.2 Prediction Strategies

Data reduction requires us to predict the final performance $\overline{m}$ from partial training data up to $t_{\text{stop}}$. We denote this prediction as $\hat{m}_{t_{\text{stop}}}$.

**Constant Prediction.** The baseline prediction is to assume the final performance is the same as the most recently observed performance: $\hat{m}^{\text{constant}}_{t_{\text{stop}}} = \overline{m}_{[t_{\text{stop}}-\Delta, t_{\text{stop}}]}$.

**Trajectory Prediction.** We can make more sophisticated predictions by fitting a parameterized law $f(D)$, where $D$ is the fraction of data seen, to the observed performance trajectory. For example, we use an inverse power law $f(D) := E + A/D^{\alpha}$. We then extrapolate to predict the final performance, $\hat{m}^{\text{trajectory}}_{t_{\text{stop}}} = f(1)$. To handle the time variation noise discussed in Section 2.1, we fit the laws not on absolute performance values, but on the pairwise performance differences between configurations. This is done by jointly optimizing the law parameters for all configurations to minimize the error on these relative performance curves, which yields more stable and accurate predictions.

**Stratified Prediction.** Our most advanced strategy addresses the fact that different data segments exhibit different distribution shifts (Figure 1(left)). Instead of a single, aggregate prediction, we slice the data into clusters and make separate predictions for each slice. Let $\mathcal{S} = \{S^{(l)}\}_{l=1}^{L}$ be a partition of the data into $L$ slices (e.g., based on feature clusters). We predict the performance on each slice, $\hat{m}^{(l)}_{t_{\text{stop}}}$, using constant or trajectory prediction on the data seen so far within that slice. The final prediction is a weighted average of these slice-level predictions, where weights are determined by the prevalence of each slice in the final evaluation window:

$$\hat{m}^{\text{stratified}}_{t_{\text{stop}}} = \frac{1}{\Delta + 1} \sum_{S^{(l)} \in \mathcal{S}} \left[ \hat{m}^{(l)}_{t_{\text{stop}}} \cdot \sum_{t=T-\Delta}^{T} \mathbb{1}\{x_t \in S^{(l)}\} \right]. \tag{1}$$

This approach transforms a difficult global prediction problem into a series of simpler, more robust local predictions. More details can be found in Appendix C.6.

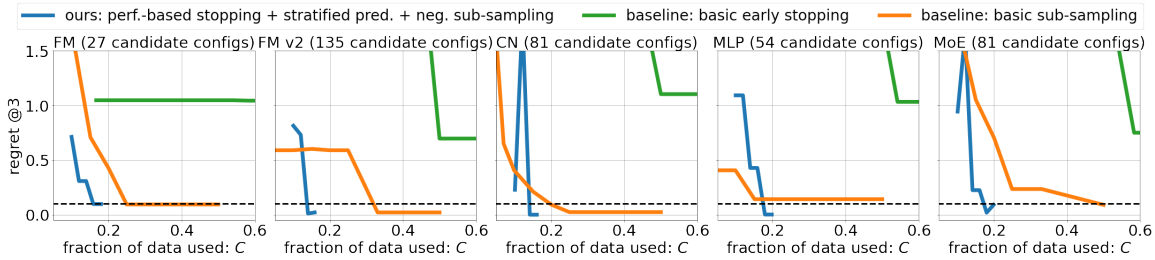


Figure 2: Our proposed method (performance-based stopping with stratified prediction on 50 % negative-subsampled data) compared to baselines. Our method achieves a `regret` @3 well below the 0.1% target with up to 10x less data (cost $C = 0.1$), significantly outperforming basic early stopping and sub-sampling.

## 4 Experimental Results

**Setup**. We conduct experiments on the Criteo 1TB pCTR dataset (Criteo, 2023), which spans 24 days of chronologically ordered data. We evaluate several architectures: Factorization Machines (FM), Cross Networks (CN), MLP, and Mixture of Experts (MoE), each with a wide range of hyperparameter configurations. Our primary metric is normalized `regret` @3 over the last 3 days of data. As baselines, we use simple uniform sub-sampling and one-shot early stopping with constant prediction since there no existing methods suitable for our setup. For full experimental details, including hyperparameter ranges, clustering methods, and additional metrics, please see Appendix C.

**Establishing an Acceptable Regret Level**. To interpret `regret` @$k$, we first normalize it by the performance of a reference model. We then measure the inherent performance variability by training the same configuration with 8 different random seeds. This revealed a typical performance variation of about 0.1%. We therefore set an acceptable "performance loss" from misranking, our target for normalized `regret` @$k$, to be 0.1%, as this is on par with noise from training randomness. This target is shown as a black dashed line in our result figures.

**Main Result**. Our most advanced method, combining **performance-based stopping**, **negative sub-sampling** (at 50%), and **stratified trajectory prediction**, dramatically reduces the cost of hyperparameter search. Figure 2 shows that across all architectures, this approach identifies the top-3 configurations with a `regret` @3 well below our 0.1% target, while using as little as 10% of the original data (a 10x cost reduction). It consistently and significantly outperforms the baselines.

**Ablation Studies**.. We find that **performance-based stopping consistently outperforms one-shot early stopping**, requiring significantly less data to reach the target regret regardless of the prediction strategy used (see Appendix Figure 4). Furthermore, advanced prediction strategies are crucial. Both **trajectory and stratified predictions significantly outperform constant prediction**. Our novel **stratified prediction** provides an additional 10% data reduction over standard trajectory prediction, highlighting the benefit of handling data heterogeneity via slicing (see Appendix Figure 6 for a detailed comparison).

## 5 Conclusion

We introduced a suite of data reduction and prediction strategies tailored for the costly task of hyperparameter search in online learning systems. Our approach of combining performance-based stopping with advanced prediction methods allows for aggressive but intelligent data reduction. The key innovation, stratified prediction, effectively handles the non-stationarity of online data by breaking the prediction problem into more manageable slices. Together, these methods achieve up to a 10-fold reduction in the computational cost of hyperparameter search, enabling faster and more efficient development of large-scale recommendation and advertising models.

## References


Alain, G., Lamb, A., Sankar, C., Courville, A., and Bengio, Y. (2015). Variance reduction in SGD by distributed importance sampling. *arXiv preprint arXiv:1511.06481*.

Anil, R., Gadanho, S., Huang, D., Jacob, N., Li, Z., Lin, D., Phillips, T., Pop, C., Regan, K., Shamir, G. I., Shivanna, R., and Yan, Q. (2022). On the factory floor: ML engineering for industrial-scale ads recommendation models. In *Proceedings of the 5th Workshop on Online Recommender Systems and User Modeling*.

Bedi, A. S., Sarma, P., and Rajawat, K. (2018). Tracking moving agents via inexact online gradient descent algorithm. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):202–217.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(1):281–305.

Chandrashekaran, A. and Lane, I. R. (2017). Speeding up hyper-parameter optimization by extrapolation of learning curves using previous builds. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 477–492. Springer.

Coleman, B., Kang, W.-C., Fahrbach, M., Wang, R., Hong, L., Chi, E., and Cheng, D. (2024). Unified Embedding: Battle-tested feature representations for web-scale ML systems. *Advances in Neural Information Processing Systems*, 36.

Coleman, C., Yeh, C., Mussmann, S., Mirzasoleiman, B., Bailis, P., Liang, P., Leskovec, J., and Zaharia, M. (2019). Selection via proxy: Efficient data selection for deep learning. *arXiv preprint arXiv:1906.11829*.

Covington, P., Adams, J., and Sargin, E. (2016). Deep neural networks for YouTube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 191–198.

Criteo (2023). Criteo pctr data - 1tb. `https://ailab.criteo.com/download-criteo-1tb-click-logs-dataset/`.

Domhan, T., Springenberg, J. T., and Hutter, F. (2015). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, volume 15, pages 3460–8.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21.

Fahrbach, M., Javanmard, A., Mirrokni, V., and Worah, P. (2023). Learning rate schedules in the presence of distribution shift. In *International Conference on Machine Learning*, pages 9523–9546. PMLR.

Feurer, M. and Hutter, F. (2019). *Hyperparameter optimization*. Springer International Publishing.

Gao, Q., Luo, Z., Klabjan, D., and Zhang, F. (2022). Efficient architecture search for continual learning. *IEEE Transactions on Neural Networks and Learning Systems*, 34(11):8555–8565.

Hazan, E., Rakhlin, A., and Bartlett, P. (2007). Adaptive online gradient descent. *Advances in Neural Information Processing Systems*, 20.

He, X., Pan, J., Jin, O., Xu, T., Liu, B., Xu, T., Shi, Y., Atallah, A., Herbrich, R., Bowers, S., et al. (2014). Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, pages 1–9.

Hoi, S. C., Sahoo, D., Lu, J., and Zhao, P. (2021). Online learning: A comprehensive survey. *Neurocomputing*, 459:249–289.

Jain, K., Xie, J., Regan, K., Chen, C., Han, J., Li, S., Li, Z., Phillips, T., Sussman, M., Troup, M., et al. (2024). Data efficiency for large recommendation models. *arXiv preprint arXiv:2410.18111*.

Jiang, A. H., Wong, D. L.-K., Zhou, G., Andersen, D. G., Dean, J., Ganger, G. R., Joshi, G., Kaminksy, M., Kozuch, M., Lipton, Z. C., et al. (2019). Accelerating deep learning by focusing on the biggest losers. *arXiv preprint arXiv:1910.00762*.

Kadra, A., Janowski, M., Wistuba, M., and Grabocka, J. (2023). Scaling laws for hyperparameter optimization. *Advances in Neural Information Processing Systems*, 36:47527–47553.

Katharopoulos, A. and Fleuret, F. (2018). Not all samples are created equal: Deep learning with importance sampling. In *International Conference on Machine Learning*, pages 2525–2534. PMLR.

Killamsetty, K., Durga, S., Ramakrishnan, G., De, A., and Iyer, R. (2021a). Grad-match: Gradient matching based data subset selection for efficient deep model training. In *International Conference on Machine Learning*, pages 5464–5474. PMLR.

Killamsetty, K., Sivasubramanian, D., Ramakrishnan, G., and Iyer, R. (2021b). Glister: Generalization based data subset selection for efficient and robust learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8110–8118.

Klein, A., Falkner, S., Springenberg, J. T., and Hutter, F. (2017). Learning curve prediction with bayesian neural networks. In *International Conference on Learning Representations*.

Kumar, M., Dahl, G. E., Vasudevan, V., and Norouzi, M. (2018). Parallel architecture and hyperparameter search via successive halving and classification. *arXiv preprint arXiv:1805.10255*.

Kurian, G., Sardashti, S., Sims, R., Berger, F., Holt, G., Li, Y., Willcock, J., Wang, K., Quiroz, H., Salem, A., et al. (2025). Scalable machine learning training infrastructure for online ads recommendation and auction scoring modeling at Google. *arXiv preprint arXiv:2501.10546*.

Linden, G., Smith, B., and York, J. (2003). Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80.

Ling, X., Deng, W., Gu, C., Zhou, H., Li, C., and Sun, F. (2017). Model ensemble for click prediction in bing search ads. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 689–698.

Liu, H., Simonyan, K., and Yang, Y. (2019). Darts: Differentiable architecture search. In *International Conference on Learning Representations*.

Liu, Z., Zou, L., Zou, X., Wang, C., Zhang, B., Tang, D., Zhu, B., Zhu, Y., Wu, P., Wang, K., and Cheng, Y. (2022). Monolith: Real time recommendation system with collisionless embedding table. In *Proceedings of the 5th Workshop on Online Recommender Systems and User Modeling co-located with the 16th ACM Conference on Recommender Systems*, volume 3303.

Loshchilov, I. and Hutter, F. (2015). Online batch selection for faster training of neural networks. *arXiv preprint arXiv:1511.06343*.

McMahan, H. B., Holt, G., Young, M., Ebner, D., Grady, J., Nie, L., Phillips, T., Davydov, E., Golovin, D., et al. (2013). Ad click prediction: A view from the trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1222–1230.

Mirzasoleiman, B., Bilmes, J., and Leskovec, J. (2020). Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*, pages 6950–6960. PMLR.

Naumov, M., Mudigere, D., Shi, H.-J. M., Huang, J., Sundaraman, N., Park, J., Wang, X., Gupta, U., Wu, C.-J., Azzolini, A. G., et al. (2019). Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091*.

Paul, M., Ganguli, S., and Dziugaite, G. K. (2021). Deep learning on a data diet: Finding important examples early in training. *Advances in Neural Information Processing Systems*, 34:20596–20607.

Pooladzandi, O., Davini, D., and Mirzasoleiman, B. (2022). Adaptive second order coresets for data-efficient machine learning. In *International Conference on Machine Learning*, pages 17848–17869. PMLR.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. (2017). Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911. PMLR.

Shen, J., Khodak, M., and Talwalkar, A. (2022). Efficient architecture search for diverse tasks. *Advances in Neural Information Processing Systems*, 35:16151–16164.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25.

Toneva, M., Sordoni, A., des Combes, R. T., Trischler, A., Bengio, Y., and Gordon, G. J. (2019). An empirical study of example forgetting during deep neural network learning. In *International Conference on Learning Representations*.

Wistuba, M. and Pedapati, T. (2020). Learning to rank learning curves. In *International Conference on Machine Learning*, pages 10303–10312. PMLR.

Yang, T., Zhang, L., Jin, R., and Yi, J. (2016). Tracking slowly moving clairvoyant: Optimal dynamic regret of online learning with true and noisy gradient. In *International Conference on Machine Learning*, pages 449–457. PMLR.

Yang, Y., Kang, H., and Mirzasoleiman, B. (2023). Towards sustainable learning: Coresets for data-efficient deep learning. In *International Conference on Machine Learning*, pages 39314–39330. PMLR.

Zinkevich, M. (2003). Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th International Conference on Machine Learning*, pages 928–936.

Zoph, B. and Le, Q. (2022). Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*.

## A Related Work

**Recommendation and Advertising Systems**. Recommendation and advertising systems form the backbone of many online platforms, such as YouTube (Covington et al., 2016), Facebook ads (He et al., 2014), Bing search (Ling et al., 2017), Google ads (Anil et al., 2022; Coleman et al., 2024; Kurian et al., 2025; McMahan et al., 2013), TikTok (Liu et al., 2022), and Amazon recommendations (Linden et al., 2003). Online learning is particularly relevant in these applications due to the constant influx of new data and the need to adapt to evolving user preferences and item popularity Anil et al. (2022); McMahan et al. (2013). This inherent distribution shift in online data introduces unique challenges, necessitating distinct solutions compared to standard "offline" learning approaches that assume distributionally stationary data (Bedi et al., 2018; Fahrbach et al., 2023; Hazan et al., 2007; Yang et al., 2016; Zinkevich, 2003). Our work addresses one such challenge: efficient hyperparameter search. While this problem has been extensively studied for offline learning, the complexities introduced by the distribution shift demand new analyses and tools.

**Data-Efficient Training**. The increasing scale of modern datasets has driven significant interest in data-efficient learning techniques. These methods aim to achieve high performance with limited, selectively chosen subset of the full training data. For instance, one line of work selects weighted subsets of the full data that best approximates the full gradient (Killamsetty et al., 2021a,b; Mirzasoleiman et al., 2020; Pooladzandi et al., 2022; Yang et al., 2023). Another approach leverages "importance signals", such as (expected) gradient norm (Alain et al., 2015; Katharopoulos and Fleuret, 2018; Paul et al., 2021), per-example loss (Jiang et al., 2019; Loshchilov and Hutter, 2015), and forgettability (Toneva et al., 2019), to guide data reduction by retaining "important" examples. This selection process can occur after some steps of training or after a full training run on some cheaper proxy model (Coleman et al., 2019). While data-efficient training is a crucial component of our proposal, existing methods do not address the distribution shift inherent in online learning. Consequently, data-efficient online learning has relied on simpler strategies like uniform or label-dependent sub-sampling of examples at more conservative rates (Jain et al., 2024). Our work overcomes this limitation by enabling a successful hyperparameter search even if the candidate configurations do not maintain their best performance on aggressively reduced data. This allows for the exploration of more aggressive data reduction strategies, which, when supported by our prediction strategies, facilitates significantly more efficient hyperparameter search.

**Hyperparameter & Architecture Search**. Hyperparameter and neural architecture search are critical yet computationally expensive tasks in machine learning. Traditional hyperparameter search methods include grid search, random search (Bergstra and Bengio, 2012), and Bayesian optimization (Snoek et al., 2012). More recent approaches leverage techniques like evolutionary algorithms (Real et al., 2017) and gradient-based optimization Liu et al. (2019). Neural architecture search aims to automate the design of neural network architectures, often employing search strategies similar to those used in hyperparameter search (Elsken et al., 2019; Zoph and Le, 2022). While these methods have shown considerable success, they often require substantial computational resources. This has motivated research in efficient hyperparameter search, aiming to identify the top configurations with reduced cost (Feurer and Hutter, 2019; Gao et al., 2022; Shen et al., 2022). A common technique adopted by practitioners is early stop the training runs of non-promising configurations based on their performance during the initial stages. This strategy can also rely on predicting the configurations' full learning trajectory using Bayesian optimization (Chandrashekaran and Lane, 2017; Klein et al., 2017), autoregressive models (Wistuba and Pedapati, 2020), or scaling laws (Domhan et al., 2015; Kadra et al., 2023). However, such predictions become particularly challenging in online learning, as the loss does not follow a smooth monotonic trend due to the distribution shift in the data. Our work addresses this challenge through several angles, including mitigating the distribution shift with a reference model trained on the same historical data, and

aggregating predicted "sliced" performance over clusters of examples with similar distributions rather than predicting the overall performance directly.

## B Performance-Based Stopping

**Algorithm 1** Performance-Based Stopping

**Input**: configurations $\Omega$, stopping steps $\mathcal{T}_{\text{stop}} \subseteq \{1, \dots, T\}$, ratio $\rho$ of stopped configurations at any stopping step
**Output**: ranking $r$

$\Omega_{\text{remaining}} \leftarrow \Omega$
Initialize ranking sequence $r \leftarrow []$.
**while** $\Omega_{\text{remaining}} \neq \emptyset$ and $t \leq T$ **do**
  Incremental training with $(x_t, y_t)$.
  **if** $t \in \mathcal{T}_{\text{stop}}$ **then**
    $\hat{m}_{\text{remaining}} \leftarrow \textsc{PredictPerformance}(\Omega_{\text{remaining}}, t, T)$
    $r_{\text{remaining}} \leftarrow \textsc{RankConfigurations}(\Omega_{\text{remaining}}, \hat{m}_{\text{remaining}})$
    $\Omega_{\text{pruned}}, r_{\text{pruned}} \leftarrow$ last $\rho \cdot |\Omega_{\text{remaining}}|$ configurations in $r$ and their rankings
    $r \leftarrow \text{concatenate}(r_{\text{pruned}}, r)$
    $\Omega_{\text{remaining}} \leftarrow \Omega_{\text{remaining}} \setminus \Omega_{\text{pruned}}$
$m_{\text{remaining}} \leftarrow \textsc{ComputePerformance}(\Omega_{\text{remaining}})$
**return** $\text{concatenate}(\textsc{RankConfigurations}(\Omega_{\text{remaining}}, m_{\text{remaining}}), r)$

## C Additional Experimental Details

### C.1 Ranking Metrics

In addition to `regret` $@k$, we also consider the Pairwise Error Rate (PER), which measures the fraction of all pairs of configurations that are incorrectly ordered by a predicted ranking $r$ compared to the ground truth ranking $r^*$.

$$\text{PER}(r) = \frac{1}{\frac{1}{2}|\Omega|(|\Omega| - 1)} \sum_{i=1}^{|\Omega|} \sum_{j=i+1}^{|\Omega|} \mathbb{1}\{\overline{m}(r(i)) > \overline{m}(r(j))\}, \tag{2}$$

where $\mathbb{1}\{P\} = 1$ when $P$ is true and 0 otherwise. While useful, PER treats all misrankings equally and considers all configurations, whereas we are primarily interested in accurately ranking the top configurations with significant performance differences. For this reason, `regret` $@k$ is our main metric. We provide corresponding results in Figure 5.

### C.2 Candidate Configurations

In the "FM" and "MoE" experiments, we vary optimization parameters: learning rate, weight decay, and final learning rate. We sweep through three values for each of these hyperparameters:

- learning rate: $[10^{-4}, 10^{-3}, 10^{-2}]$
- weight decay: $[10^{-6}, 2 \cdot 10^{-6}, 10^{-5}]$
- final learning rate: $[10^{-3}, 10^{-2}, 10^{-1}]$

In the "FM v2" experiment, in addition to the optimization parameters above, we vary the memory structure of the embeddings. We divide the features into two groups: "high" and "low"

cardinality features and share the embedding tables among them (using hashing). Then, we vary the embedding dimensions as well as the hash buckets for the high and low cardinality features while maintaining a constant training speed and memory footprint. To ensure that all embeddings have the same dimensions in the FM computation, we project them to the same embedding size.

In the "CN" and "MLP" experiments, in addition to the optimization parameters above, we vary the number of layers and hidden dimensions, respectively:

- number of layers in CN: $[2, 3, 5]$
- hidden dimensions in MLP: $[(598, 598, 598, 598), (1196, 1196, 1196, 1196)]$

### C.3 Compute Resources

Without applying our data reduction and prediction strategies, running one configuration over the 24-day Criteo data on 4 TPUv3s takes less than an hour for the architectures and hyperparameters we tried. We manage to reduce this cost up to 10 times with our strategies. Trajectory prediction requires learning the parameters of a predictive law. We did this using CPUs, with less than an hour for learning the trajectory predictions for each configuration in an experiment of about 100 configurations in total.

### C.4 Details on Performance-Based Stopping

We provide an outline of our performance-based stopping strategy in Algorithm 1, with two hyperpameters: stopping steps $\mathcal{T}_{\text{stop}}$ and ratio of stopped configurations at any stopping step $\rho$. For all the experiments, we choose $\rho = 0.5$. We choose this value since it is a reasonable starting point but it is possible to improve our results with a carefully tuned $\rho$. For $\mathcal{T}_{\text{stop}}$, we fix the number of steps between consecutive stopping times in $\mathcal{T}_{\text{stop}}$ – i.e., we stop fraction $\rho$ of remaining configurations at equally spaced time steps. By varying the frequency at which we stop the configurations, we obtain different points in the performance-based stopping curves in our plots.

### C.5 Details on Trajectory Prediction

Trajectory prediction fits a law to the recently observed data and makes predictions for the trajectory. In our experiments, we use the last 3 "visited" days in the training data for fitting, by averaging the loss over all the time steps in a day. Then, we use the fitted parameters to predict the loss in the last 3 days, i.e., the evaluation window. In the main body, we use inverse power laws (or `InversePowerLaw` in Table 1). In Section D.3, we explore other choices of laws, listed in Table 1, and their combinations.

Table 1: Other choices of fitting laws for trajectory prediction.

| Law | Formulation (function of $D$) |
|---|---|
| `InversePowerLaw` | $E + \frac{A}{D^\alpha}$ |
| `VaporPressure` | $\exp\left(A + \frac{B}{D} + C \cdot \log D\right)$ |
| `LogPower` | $\frac{A}{1+(\frac{D}{\exp B})^\alpha}$ |
| `ExponentialLaw` | $E - \exp\left(-A \cdot D^\alpha + B\right)$ |

### C.6 Details on Stratified Prediction

Stratified prediction can be used with both constant or trajectory prediction. In the main body, we exclusively used it with trajectory prediction as we observe better ranking accuracy. Figure 3 shows that stratified trajectory prediction is consistently better than stratified constant prediction across all experiments we consider. Note that in all other plots in the paper, "stratified prediction" refers to stratified trajectory prediction.

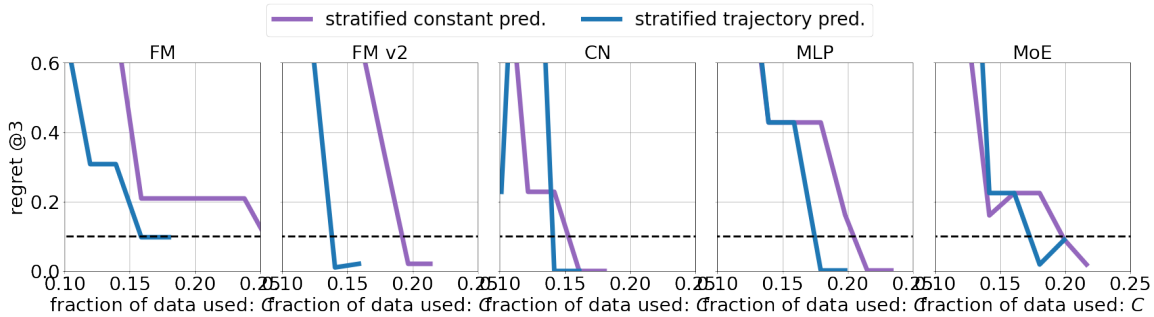


Figure 3: Comparison between stratified constant prediction and stratified trajectory prediction. In all other plots in the paper, "stratified prediction" refers to stratified trajectory prediction.

# D Additional Experimental Results

## D.1 One-Shot Early Stopping vs Performance-Based Stopping

In the main text, we stated that performance-based stopping reaches lower `regret` @3 with significantly smaller amount of data used, compared to one-shot early stopping. We now provide the full set of results across all model types in Figure 4.

Even though performance-based stopping should not be evaluated with ranking metrics that consider *all* the configurations, like PER, as the core idea behind performance-based stopping is to not worry about poor configurations, we still provide a comparison between one-shot early stopping and performance-based stopping based on PER in Figure 5. We note that this is not a fair comparison since performance-based stopping is actually not expected to rank the poor configurations that stop early on rank accurately. However, Figure 5 shows that the gap between one-shot early stopping and performance-based stopping is so large that the latter reaches much lower PER.

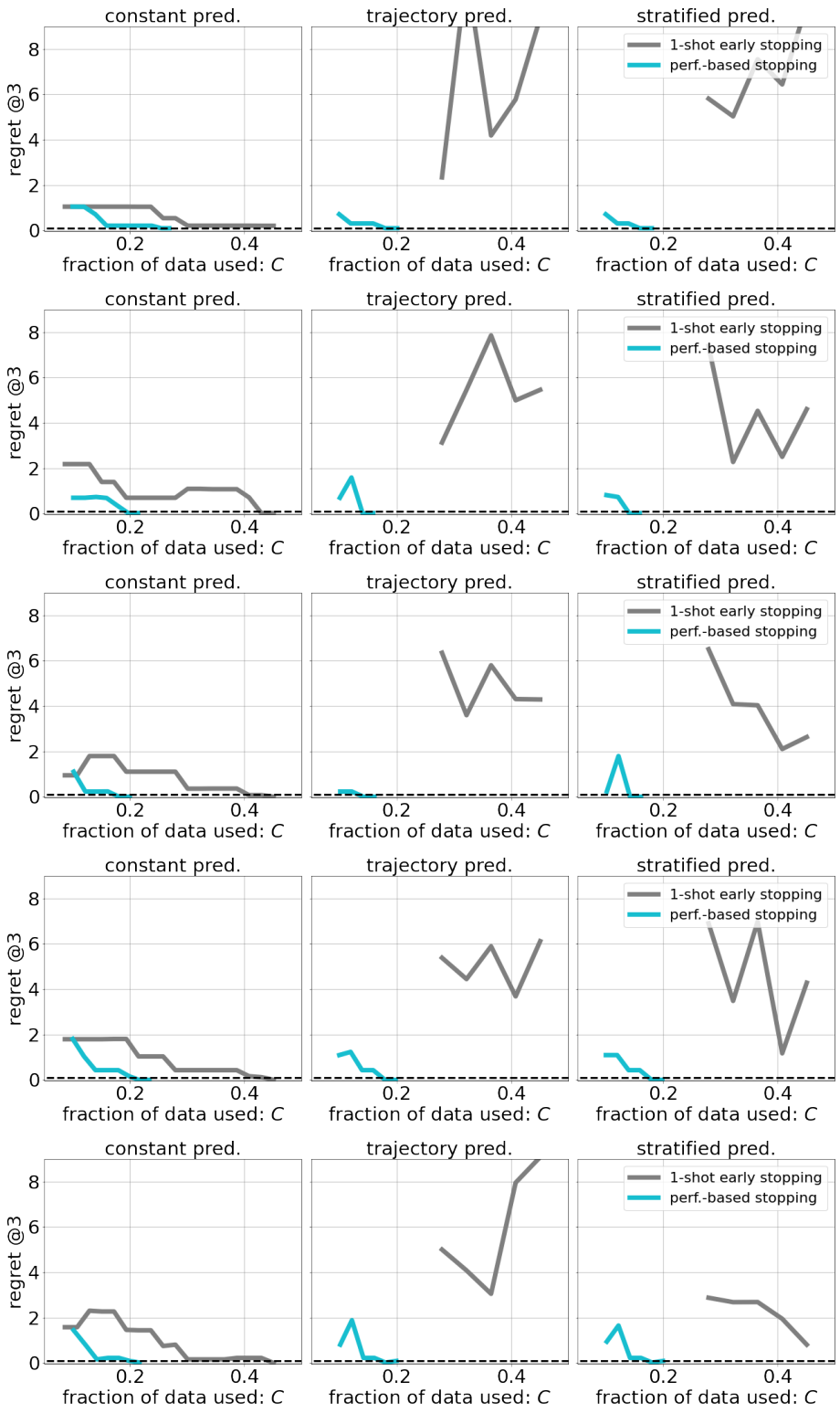


Figure 4: `regret` @3 comparison of one-shot early stopping and performance-based stopping, when used with **(left)** constant, **(center)** trajectory, and **(right)** stratified prediction. Rows represent, top to bottom: FM, FM v2, CN, MLP, MoE. Performance-based stopping is consistently superior.

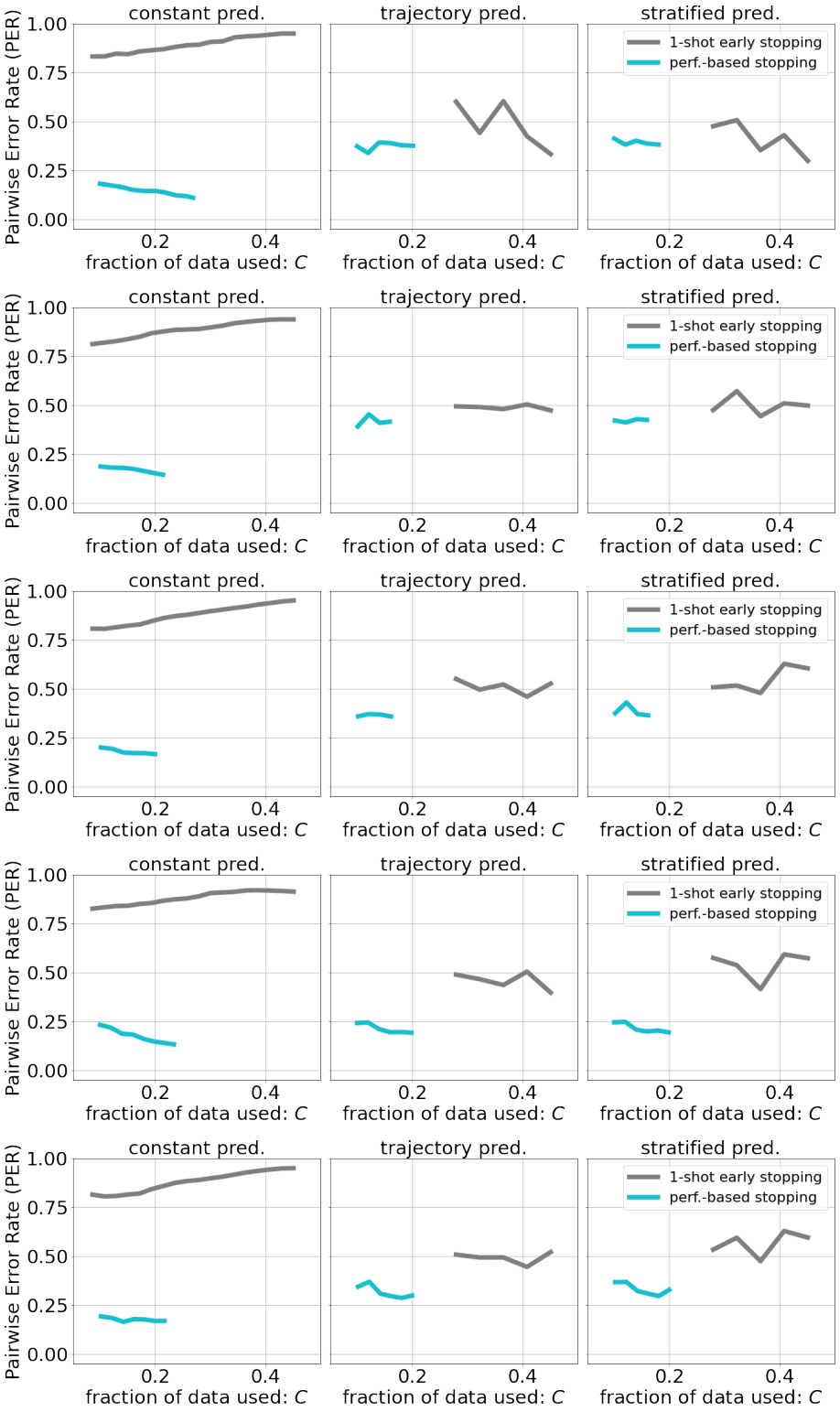


Figure 5: PER comparison of one-shot early stopping and performance-based stopping, when used with **(left)** constant, **(center)** trajectory, and **(right)** stratified prediction. Rows represent, top to bottom: FM, FM v2, CN, MLP, MoE.

## D.2 Comparison between Prediction Strategies

We stated in the main text that advanced prediction strategies are critical, with stratified prediction outperforming trajectory prediction, and both being far superior to constant prediction. Figure 6 provides the detailed results.

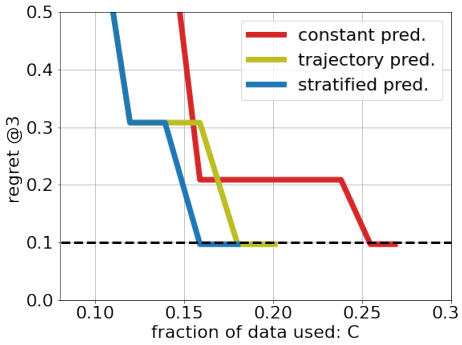


Figure 6: Comparison between our prediction strategies (Constant, Trajectory, Stratified) when used with performance-based stopping, shown for MoE. Stratified and Trajectory prediction consistently outperform Constant prediction, with Stratified prediction offering the best performance.

## D.3 Other Choices of Scaling Laws

Figure 7 compares different choices of laws, `InversePowerLaw`, `VaporPressure`, `LogPower`, `ExponentialLaw` defined in Table 1, and their weighted combination, for trajectory prediction. It is seen that they all behave similarly and reach the target `regret` @3 level around the same data use fraction. In the combined law, we learn both the weights and the parameters of each law jointly.

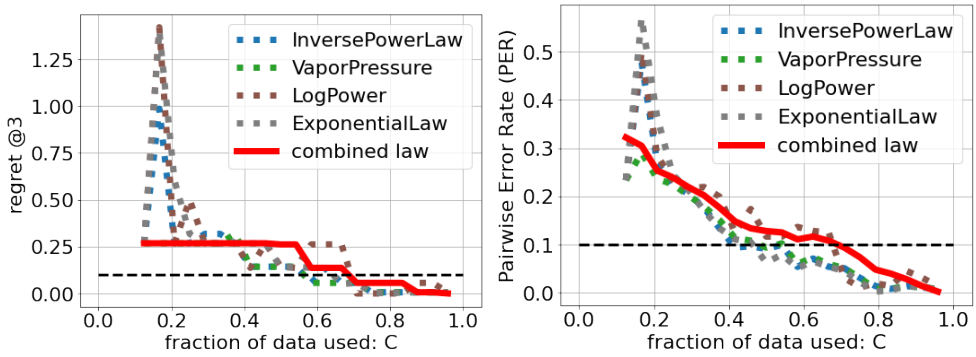


Figure 7: Comparison between different choices of laws for trajectory prediction, with respect to **(left)** `regret` @3 and **(right)** PER. The simple inverse power law performs as well as more complex alternatives.

## D.4 Late Starting vs Early Stopping

Lastly, we explore whether starting some configuration runs later in training data would provide an improvement in the ranking accuracy vs data reduction tradeoff. Figure 8 shows how one-shot early stopping behaves when the configurations start training at different days. Start at day= 0 (purple) corresponds to the standard one-shot early stopping that we have been analyzing. The other curves apply the same algorithm but on the runs that start training after some number of days in training data passes. We see no significant difference among different start days.

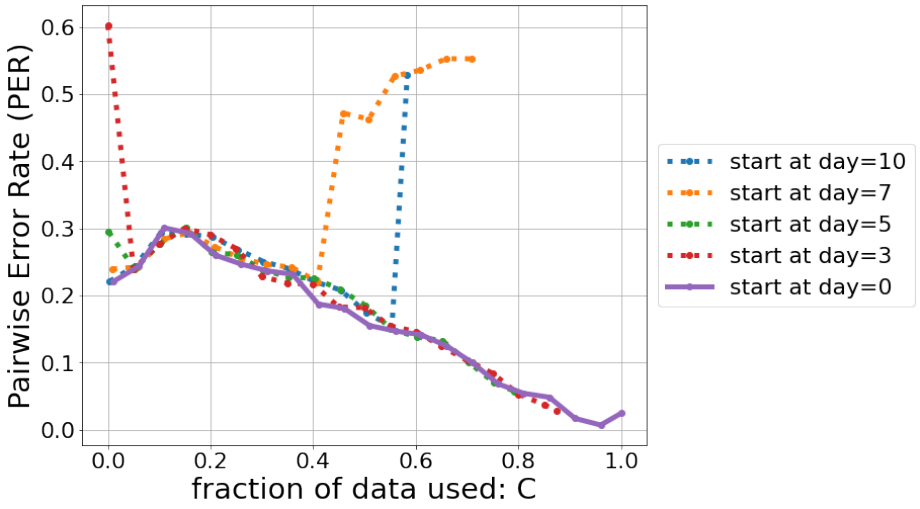


Figure 8: PER comparison between different starting times, when used with one-shot early stopping. This comparison checks whether we could get additional data reduction gain by late starting the configuration runs. However, our ranking predictions with late-started runs provide about the same data reduction vs PER tradeoff.