Asymmetric Roles in Deep CTR Generalization: A Block-wise Retuning Diagnosis

Ruifan Wu[1,2] **Haijun Zhao**[3]* **Ying Jiang**[1,2]†
[1]School of Computer Science and Engineering, Sun Yat-sen University, China
[2]Guangdong Province Key Laboratory of Computational Science, China
[3]WeChat, Tencent Inc
wurf26@mail.sysu.edu.cn asterzhao@tencent.com jiangy32@mail.sysu.edu.cn

Abstract

Deep click-through rate (CTR) models commonly consist of sparse embedding blocks and feature-interaction blocks, which jointly learn predictive signals from high-dimensional categorical features. This naturally raises the question of whether different blocks in an end-to-end trained CTR model play the same role in generalization. To address this question, we investigate the generalization behavior of deep CTR models from a block-wise diagnostic perspective. Specifically, we formulate deep CTR models as two-block systems comprising an embedding block and a feature-interaction block, and introduce a retuning-based diagnostic protocol to assess the roles of different blocks in generalization by retaining one jointly trained block and re-estimating the other. Experiments across multiple datasets and architectures reveal a consistent asymmetry in these roles: retaining the learned feature-interaction state supports better held-out generalization after counterpart re-estimation than retaining the learned embedding state. Motivated by this diagnosis, we further develop a component-selective embedding aggregation method that averages multiple independently retuned embedding tables while keeping the feature-interaction block fixed. The resulting model retains the original single-path inference architecture and incurs no additional online inference cost. Experiments on Avazu, Criteo, and Taobao show that the proposed method improves representative deep CTR models over standard training and retuning baselines.

1 Introduction

Click-Through Rate (CTR) prediction is central to ranking in recommender systems and online advertising, where it estimates the probability of a user clicking an item under a given context. Most deep CTR models follow an *Embedding & Feature Interaction* paradigm Zhu et al. [2021]: sparse categorical features are first mapped to dense embeddings, and a feature-interaction (FI) module then models cross-feature interactions for prediction. Motivated by advances in deep architectures, many recent CTR models focus on designing stronger interaction modules Cheng et al. [2016], Guo et al. [2017], Lian et al. [2018], Wang et al. [2021], Fu et al. [2025], Shan et al. [2016], Song et al. [2019], although retrospective studies suggest that performance differences among recent models are often marginal Zhu et al. [2021].

A recent empirical study Zhang et al. [2022] further highlights a critical characteristic of deep CTR models: they often achieve their best performance at the end of the first training epoch, followed by a sharp performance drop at the beginning of the second epoch, which cannot be recovered through

*Work done while the author was a student at Sun Yat-sen University.
†Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

further training. This behavior, referred to as the **one-epoch phenomenon**, has been observed across many CTR models and is believed to be related to model structure, feature sparsity, and optimization dynamics. While this phenomenon has been empirically identified, its internal mechanisms, especially how different modules contribute to it, remain insufficiently understood.

To investigate this behavior, we draw inspiration from evaluation protocols in representation learning, such as fine-tuning and linear probing Chen et al. [2020], Devlin et al. [2018]. We propose a simple yet effective empirical strategy, *reinitializing and tuning* (**retuning**), where we freeze one part of the model (either the embedding or the feature interaction module) and retrain the other. This approach enables us to assess the role of each trained block in generalization under counterpart re-estimation. Our experiments reveal a notable asymmetry: retaining FI and re-estimating embeddings yields models with better held-out performance and smaller train–validation gaps than retaining embeddings and re-estimating FI.

Building on this diagnosis, we seek an intervention that re-estimates embeddings while preserving the trained FI module. Although embedding retuning improves generalization, the retuned embeddings are still obtained from sparse and run-dependent training trajectories, and may retain run-to-run variability. We therefore keep the feature-interaction module fixed, independently re-estimate the embedding block several times, and average the resulting embedding tables before deployment. The resulting method, Embedding Soups, preserves the original single-path inference architecture and introduces no additional online inference cost. We conduct extensive experiments to validate its effectiveness, robustness, and compatibility across representative CTR models and real-world datasets.

In summary, our contributions are threefold:

- We investigate the learning behaviors of embeddings and feature interaction modules in deep CTR models, and find that retaining the learned FI state supports better held-out generalization under counterpart re-estimation than retaining embeddings.
- Based on these findings, we propose a model-agnostic optimization strategy, *embedding retuning and soups*, which improves CTR model performance without introducing additional inference cost.
- We validate our approach on multiple representative deep CTR models and real-world datasets, and provide further analysis that deepens our understanding of training dynamics in CTR modeling, including a local variance-based explanation for why *Embedding Soups* works.

2 Problem setup and positioning

2.1 Sparse CTR models as two-block systems

CTR prediction is commonly formulated as a binary classification problem. Given an input instance with n feature fields, $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$, each $\mathbf{x}_i$ is usually a high-dimensional sparse vector, typically represented by a one-hot encoding of a categorical feature. Most deep CTR models follow an embedding–interaction paradigm. The embedding block maps each sparse field into a dense vector $\mathbf{e}_i = \mathbf{x}_i \mathbf{E}_i$, $1 \le i \le n$, where $\mathbf{E}_i \in \mathbb{R}^{|F_i| \times d}$ is the embedding table for field i, $|F_i|$ is the number of valid feature values, and d is the embedding dimension. The resulting embeddings are concatenated as $\mathbf{e} = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n]$. The feature-interaction block then predicts the click probability by $\hat{y} = \sigma(f_{\mathbf{W}}(\mathbf{e}))$, where $f_{\mathbf{W}}$ is parameterized by $\mathbf{W}$ and σ is the sigmoid function. The model is trained by minimizing the empirical binary cross-entropy loss over both the embedding parameters $\mathbf{E} = [\mathbf{E}_1, \ldots, \mathbf{E}_n]$ and the interaction parameters $\mathbf{W}$.

We view a deep CTR model as a coupled two-block system $(\mathbf{E}, \mathbf{W})$. The embedding block performs sparse categorical representation learning and contains a large number of entry-wise parameters whose update frequencies are highly imbalanced. In contrast, the feature-interaction block operates on dense embeddings and learns interaction patterns shared across samples. This structural difference motivates a component-wise view of generalization.

2.2 One-epoch overfitting and sparse embedding regularization

Deep CTR models often exhibit the one-epoch phenomenon: validation performance peaks after very few training passes, often around the end of the first epoch, and then deteriorates with continued

training. Prior work connects this behavior to feature sparsity, fast optimization, and the distribution shift induced by trained embeddings Zhang et al. [2022]. MEDA further supports the role of embeddings by reinitializing categorical embeddings at epoch boundaries to enable multi-epoch training Fan et al. [2024]. Representative training curves and a compact architecture summary are provided in Appendix A.1.

Recent adaptive regularization methods provide a stronger embedding-side explanation. They relate one-epoch overfitting to the norm budget of large-scale sparse embedding layers through Rademacher complexity, and design frequency-aware regularization according to the effective update interval of each categorical feature value Li and Lyu [2026]. These studies show that sparse embedding overfitting and feature frequency are central to the degradation.

Our work builds on this view but addresses a different question. Existing regularization-based approaches mainly ask how to control embedding overfitting within a single training trajectory. We instead ask whether the embedding block and the feature-interaction block play symmetric roles in the generalization degradation. This distinction is important because observing embedding-related generalization degradation does not by itself determine whether the interaction module should also be discarded, regularized, reinitialized, or averaged.

2.3 From one-epoch overfitting to component-wise generalization

The one-epoch phenomenon thus provides an empirical lens for studying component-wise generalization. Differences in generalization after re-estimating different blocks suggest that whole-model explanations and uniform regularization may obscure the internal structure underlying this failure mode.

We focus on the following question: after standard end-to-end training, does retaining the feature-interaction block support better held-out generalization under counterpart re-estimation than retaining the embedding block? To answer this, Section 3 introduces a block-wise retuning diagnostic that evaluates the conditional reusability of embeddings and feature interactions.

This perspective also motivates the method developed later. If the retained interaction state supports useful embedding re-estimation and the resulting estimates vary across runs, then a natural intervention is to keep that FI state fixed and aggregate the independently re-estimated embeddings. In this sense, embedding aggregation is not introduced as a generic ensemble trick, but as a component-selective response to asymmetric generalization.

3 Diagnosing asymmetric generalization

We use block-wise retuning to examine how the two blocks support generalization in deep CTR training. Retuning is used only as a diagnostic in this section; the algorithmic use of retuned embeddings is introduced in Section 4. The goal is to determine which learned block state supports better held-out generalization after its counterpart is re-estimated.

3.1 Block-wise retuning protocol

Let $(\mathbf{E}_0, \mathbf{W}_0)$ denote the embedding and feature-interaction (FI) parameters obtained by standard end-to-end training. We consider two complementary directions. To test whether the learned embeddings support generalization after FI retuning, we freeze $\mathbf{E}_0$, reinitialize the FI module, and retune only $\mathbf{W}$:

$$\mathbf{W}_{\mathrm{rt}} = \arg\min_{\mathbf{W}} \widehat{\mathcal{R}}(\mathbf{E}_0, \mathbf{W}),$$

where $\widehat{\mathcal{R}}$ denotes the empirical training risk. Conversely, to test whether the learned FI module supports generalization after embedding retuning, we freeze $\mathbf{W}_0$, reinitialize the embedding block, and retune only $\mathbf{E}$:

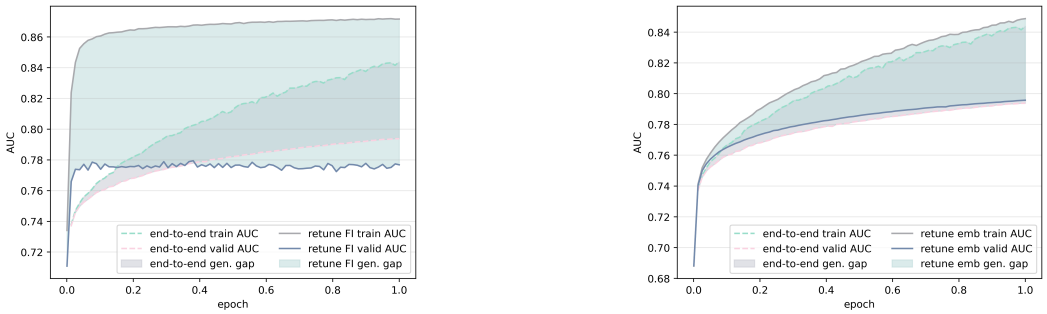
$$\mathbf{E}_{\mathrm{rt}} = \arg\min_{\mathbf{E}} \widehat{\mathcal{R}}(\mathbf{E}, \mathbf{W}_0).$$

In practice, both directions use the same data, optimizer, training budget, and checkpoint-selection rule. If the first direction attains high training AUC but poor validation AUC, the frozen embeddings do not support good generalization after FI retuning. If the second direction recovers or improves validation performance, the FI module supports generalization with re-estimated embeddings. Unless

otherwise specified, we use a deep neural network (DNN) Cheng et al. [2016] as the representative FI module and report AUC, logloss, and the train–validation AUC gap.

3.2 Asymmetric roles in generalization

The two directions expose a clear asymmetry. With frozen end-to-end embeddings, FI retuning keeps increasing training AUC while validation AUC remains low; with a frozen FI module, embedding retuning improves validation AUC and keeps the train–validation gap much smaller (Figure 1). Table 1 shows the same pattern on all three datasets: embedding retuning gives higher AUC, lower logloss, and a smaller gap than FI retuning. This pattern extends to four additional FI architectures (DeepFM Guo et al. [2017], IPNN Qu et al. [2016], DCN-v2 Wang et al. [2021], and FinalMLP Mao et al. [2023]) on Avazu (Table 2): embedding retuning improves test AUC over end-to-end training, whereas FI retuning reduces it. Thus, the learned FI module supports better generalization after counterpart retuning than the learned embeddings. After tuning the optimization settings on Avazu with DNN and DCN-v2, FI retuning achieves higher training AUC but lower validation AUC than embedding retuning (Appendix B.3.1). The performance ordering also persists under timestamp-based evaluation on Avazu and an order-preserving proxy on Criteo (Appendix B.3.2).



(a) Frozen embedding, retune FI.

(b) Frozen FI, retune embedding.

Figure 1: AUC of retuned DNN models on the training and validation sets of Avazu. The dashed line denotes the end-to-end reference.

Table 1: Block-wise retuning diagnostic across datasets with DNN. Embedding retuning freezes FI and retunes embeddings; FI retuning freezes embeddings and retunes FI. AUC and logloss are test scores. Gap is the train–validation AUC gap.

Protocol	Avazu			Criteo			Taobao Ads		
	AUC ↑	logloss ↓	gap	AUC ↑	logloss ↓	gap	AUC ↑	logloss ↓	gap
Embedding retuning	0.7960	0.3701	0.0345	0.8104	0.4412	0.0168	0.6379	0.1937	0.1369
FI retuning	0.7752	0.4034	0.0959	0.7987	0.4615	0.0468	0.6307	0.2055	0.1448

Table 2: Block-wise retuning across FI architectures on Avazu. Test ΔAUC is relative to each architecture's end-to-end model; gap is training AUC minus validation AUC.

FI module	Test ΔAUC ↑		Train–validation gap
	Embedding retuning	FI retuning	FI retuning
DeepFM	+0.0020	-0.0179	0.0960
IPNN	+0.0017	-0.0172	0.0988
DCN-v2	+0.0015	-0.0185	0.0968
FinalMLP	+0.0020	-0.0192	0.1006

3.3 Epoch-boundary interventions

To relate this asymmetry to the one-epoch phenomenon, we compare continued training and reinitialization at the epoch boundary (Table 3). Starting from the same first-epoch checkpoint, we compare continuing to train both blocks, FI only, or embeddings only, and reinitializing and training either block. In each single-block intervention, the other block remains frozen. In the three settings where continued joint training reduces validation AUC, only reinitializing and training the embeddings improves it. Updating FI alone can cause a larger decline. These results support an interpretation in terms of asymmetric coupling between the jointly trained blocks.

Table 3: Validation ΔAUC after the second epoch, measured from the same first-epoch checkpoint. All interventions use the same training budget; only the indicated blocks are updated.

Dataset / FI module	Continue training			Reinitialize and train	
	Both blocks	FI only	Embeddings only	Embeddings	FI
Avazu–DNN	-0.0050	-0.0186	-0.0024	+0.0020	-0.0183
Avazu–DCN-v2	-0.0049	-0.0179	-0.0027	+0.0017	-0.0188
Taobao–DNN	-0.0106	-0.0059	-0.0038	+0.0024	-0.0051
Criteo–DNN	+0.0010	-0.0033	+0.0013	+0.0013	-0.0041

3.4 Robustness to FI training and sparsity

We further vary the training-instance ratio used to learn the FI module and the feature-frequency threshold used to control sparsity. Table 4 gives a compact Avazu summary. Even a partially trained FI module supports meaningful embedding retuning, while pruning sparse feature values hurts end-to-end training more severely than retuning under a fixed FI. These controls support the same diagnosis: the FI module is reusable for embedding retuning once useful interactions have been learned. Full AUC and logloss tables on all datasets are in Appendix B.3.

Table 4: Compact Avazu controls for the component-wise diagnosis. FI training ratio controls how much data is used before embedding retuning; frequency threshold controls sparse-feature pruning. Gain is retuned AUC minus end-to-end AUC under the same setting.

Control	Setting	end-to-end AUC ↑	Embedding-retuned AUC ↑	Gain
FI training ratio	20%	0.7681	0.7904	+0.0223
	60%	0.7848	0.7946	+0.0098
	100%	0.7942	0.7960	+0.0018
Frequency threshold	2	0.7942	0.7960	+0.0018
	25	0.7745	0.7962	+0.0217
	250	0.7679	0.7933	+0.0254

These diagnostics motivate the method introduced next: keep the trained FI module fixed, re-estimate the embedding block, and aggregate multiple independently retuned embeddings.

4 Component-selective embedding aggregation

The diagnostics in Section 3 suggest a simple but structured intervention: preserve the trained FI module and operate only on the embedding block. In this section, we formalize this idea through fixed-FI embedding retuning and embedding aggregation. The resulting method keeps the inference architecture unchanged while reducing the variability of sparse embedding estimates.

4.1 From diagnosis to method

The diagnostics in Section 3 suggest that the two blocks should be treated differently after end-to-end training. The FI module $\mathbf{W}_0$ supports effective embedding re-estimation, whereas the resulting

embedding estimates retain run-dependent variation. We therefore keep $\mathbf{W}_0$ fixed and operate only on the embedding block.

This gives a component-selective strategy: re-estimate the embeddings under the fixed FI module and aggregate multiple such estimates. Unlike prediction ensembling or full-model averaging, the aggregation is applied only to the independently re-estimated embedding block, while the final deployed model remains a single-path CTR model.

4.2 Fixed-FI embedding retuning

We consider a deep CTR model $f(\mathbf{x}, \mathbf{E}, \mathbf{W})$, where $\mathbf{E}$ denotes the embedding parameters and $\mathbf{W}$ denotes the FI parameters. Given the end-to-end trained FI module $\mathbf{W}_0$, we reinitialize the embedding block and optimize it under the fixed FI module:

$$\mathbf{E}^{(i)} = \arg\min_{\mathbf{E}} \widehat{\mathcal{R}}(\mathbf{E}, \mathbf{W}_0; \xi_i),$$

where $\widehat{\mathcal{R}}$ is the empirical training risk and ξ_i denotes the randomness of the i-th retuning run, such as initialization and mini-batch ordering.

A single retuned embedding table already improves the compatibility between embeddings and the fixed FI module. However, it is still a sparse estimate obtained from one training trajectory, and may retain run-specific fluctuations. We therefore run embedding retuning independently for N times, obtaining

$$\mathbf{E}^{(1)}, \mathbf{E}^{(2)}, \ldots, \mathbf{E}^{(N)},$$

all associated with the same FI module $\mathbf{W}_0$.

4.3 Embedding Soups

Given N independently retuned embedding tables associated with the same FI module $\mathbf{W}_0$, Embedding Soups forms

$$\mathbf{E}^{\text{soup}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{E}^{(i)},$$

and deploys the single model

$$f(\mathbf{x}, \mathbf{E}^{\text{soup}}, \mathbf{W}_0).$$

The procedure is simple: train $(\mathbf{E}_0, \mathbf{W}_0)$ end-to-end, freeze $\mathbf{W}_0$, retune N embedding tables independently, and average them before deployment. Although the averaging operation is related to model soups Wortsman et al. [2022], our use is block-selective rather than full-model averaging. The FI module is intentionally kept fixed because it is the more reusable block under the diagnostic in Section 3. The final model contains one embedding table and one FI module, so no additional online inference cost is introduced; the extra cost is limited to offline retuning and temporary checkpoint storage.

4.4 A local variance interpretation

We provide a local interpretation of why simple block-wise averaging can improve generalization under the fixed-FI retuning regime. This interpretation clarifies the role of averaging. Let

$$\mathcal{R}(\mathbf{E}, \mathbf{W}) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{P}} \left[\ell(f(\mathbf{x}, \mathbf{E}, \mathbf{W}), y)\right]$$

be the population risk. Under the fixed FI module $\mathbf{W}_0$, let $\mathbf{E}^\star(\mathbf{W}_0)$ denote a local population-risk minimizer. A retuned embedding table can be written locally as

$$\mathbf{E}^{(i)} = \mathbf{E}^\star(\mathbf{W}_0) + \boldsymbol{\epsilon}^{(i)},$$

where $\boldsymbol{\epsilon}^{(i)}$ represents run-specific fluctuation caused by initialization, stochastic optimization, and sparse supervision. The averaged embedding satisfies

$$\mathbf{E}^{\text{soup}} = \mathbf{E}^\star(\mathbf{W}_0) + \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{\epsilon}^{(i)}.$$

Thus, if the fluctuations from different retuning runs are not fully correlated, averaging reduces their variance while keeping the same FI module.

This can be made explicit under a local quadratic approximation. Let $\theta = \mathrm{vec}(\mathbf{E})$ and suppose that, around $\theta^\star = \mathrm{vec}(\mathbf{E}^\star(\mathbf{W}_0))$, the fixed-FI risk is approximated by

$$\widetilde{\mathcal{R}}(\theta) \approx \frac{1}{2}(\theta - \theta^\star)^\top H(\theta - \theta^\star), \qquad H \succeq 0.$$

Assume that all ingredients and their average lie in a compatible basin where this approximation holds. Let $\theta^{(i)} = \theta^\star + \varepsilon^{(i)}$, where

$$\mathbb{E}[\varepsilon^{(i)}] = 0, \qquad \mathrm{Cov}(\varepsilon^{(i)}) = \Sigma, \qquad \mathrm{Cov}(\varepsilon^{(i)}, \varepsilon^{(j)}) = \rho\Sigma \quad (i \neq j),$$

with $\rho \in [0, 1)$. For

$$\bar{\theta} = \frac{1}{N}\sum_{i=1}^{N} \theta^{(i)} = \mathrm{vec}(\mathbf{E}^{\mathrm{soup}}),$$

we obtain

$$\mathbb{E}\,\widetilde{\mathcal{R}}(\theta^{(i)}) - \mathbb{E}\,\widetilde{\mathcal{R}}(\bar{\theta}) = \frac{1}{2}\left(1 - \frac{1}{N}\right)(1 - \rho)\mathrm{tr}(H\Sigma).$$

This expression shows that averaging reduces the local risk contribution of run-specific embedding fluctuations. The gain increases with the number of ingredients but exhibits diminishing returns, and it becomes smaller when different retuning runs are highly correlated.

For long-tailed sparse CTR features, embedding entries with small effective sample counts are typically less stable estimates. The variance-reduction view is therefore consistent with the sparse-estimation nature of CTR embeddings. In this sense, Embedding Soups can be understood as a block-selective variance-reduction mechanism that exploits the asymmetric generalization structure of sparse CTR models. This local argument provides a testable explanation of variance reduction under the fixed-FI retuning regime.

5 Experiment

We evaluate on three real-world CTR benchmarks: Avazu Avazu [2014], Criteo Criteo Labs [2014], and Taobao Ads Tianchi [2018]. We use five representative deep CTR models—DNN Cheng et al. [2016], DeepFM Guo et al. [2017], IPNN Qu et al. [2016], DCN-v2 Wang et al. [2021], and FinalMLP Mao et al. [2023]—and report AUC and logloss. The experiment is repeated with three random seeds; paired two-tailed t-tests are used for significance testing. Unless otherwise specified, embedding soups average four independently retuned embedding tables. Code, preprocessing scripts, and model implementations are provided in the supplementary material.

For all protocols, model selection is based on validation AUC, and we report the corresponding test-set performance. End-to-end training, embedding retuning, and all control baselines use the same validation split and the same early-stopping criterion unless otherwise specified. In embedding retuning, the FI module selected from end-to-end training is frozen, the embedding block is reinitialized, and only the embedding parameters are optimized. Dataset statistics, preprocessing details, and hyperparameters are provided in Appendix A.2. The experiments are organized around four questions: whether the gains are consistent across datasets and architectures, whether they can be explained by extra budget, capacity, regularization, or restart selection, whether deployment remains a single-model path, and whether the behavior matches the averaging-based interpretation.

5.1 Overall performance

Table 5 summarizes the main AUC and logloss results. Embedding retuning already improves the average AUC by 0.0015 and reduces logloss by 0.0011, confirming the asymmetry that a frozen FI module remains useful when paired with newly estimated embeddings. Embedding soups further improve the average AUC by 0.0032 and reduce logloss by 0.0021 over end-to-end training. The soup improvement appears in every model–dataset pair for both metrics, with significant AUC gains in all cases, supporting the model-agnostic nature of the component-selective intervention. Although the absolute AUC gains are numerically small, CTR benchmarks are saturated and prior CTR studies

commonly treat a 0.001-level AUC lift as meaningful Guo et al. [2017], Wang et al. [2021]; our average soup gain exceeds this level while preserving a single deployed model. The corresponding best epochs are reported in Appendix A.3: end-to-end training selects epoch 1 except for three Criteo settings that select epoch 2, whereas all retuned ingredients and soups select epoch 1, so the gains are not due to longer training.

Table 5: Main results on three CTR datasets. E2E, EmbRT, and Soup denote end-to-end training, embedding retuning, and Embedding Soups. Avg. diff. is relative to the end-to-end baseline of the same model. $*$ denotes $p < 0.01$ under a paired two-tailed t-test.

Model	Protocol	Avazu		Criteo		Taobao Ads		Avg. diff.	
		AUC ↑	logloss ↓	AUC ↑	logloss ↓	AUC ↑	logloss ↓	AUC ↑	logloss ↓
DNN	E2E	0.7942	0.3716	0.8096	0.4424	0.6356	0.1948	–	–
	EmbRT	0.7960*	0.3701*	0.8104*	0.4412*	0.6379*	0.1937*	+0.0016	-0.0013
	Soup	**0.7980***	**0.3689***	**0.8120***	**0.4397***	**0.6393***	**0.1936***	**+0.0033**	**-0.0022**
DeepFM	E2E	0.7944	0.3712	0.8087	0.4430	0.6355	0.1947	–	–
	EmbRT	0.7964*	0.3699*	0.8104*	0.4414*	0.6379*	0.1937*	+0.0020	-0.0013
	Soup	**0.7982***	**0.3687***	**0.8120***	**0.4399***	**0.6392***	**0.1936**	**+0.0036**	**-0.0022**
IPNN	E2E	0.7964	0.3700	0.8104	0.4418	0.6357	0.1946	–	–
	EmbRT	0.7982*	0.3688*	0.8110*	0.4408*	0.6379*	0.1937*	+0.0015	-0.0010
	Soup	**0.8001***	**0.3676***	**0.8126***	**0.4392***	**0.6393***	**0.1936***	**+0.0032**	**-0.0020**
DCN-v2	E2E	0.7944	0.3712	0.8103	0.4418	0.6357	0.1944	–	–
	EmbRT	0.7959*	0.3702*	0.8109*	0.4408*	0.6377*	0.1938*	+0.0014	-0.0009
	Soup	**0.7979***	**0.3689***	**0.8125***	**0.4393***	**0.6392***	**0.1936***	**+0.0031**	**-0.0019**
FinalMLP	E2E	0.7949	0.3709	0.8097	0.4422	0.6353	0.1946	–	–
	EmbRT	0.7969*	0.3695*	0.8111*	0.4406*	0.6350	0.1945	+0.0010	-0.0010
	Soup	**0.7988***	**0.3683***	**0.8126***	**0.4393***	**0.6366***	**0.1942***	**+0.0027**	**-0.0020**
Avg. diff.	EmbRT	+0.0018	-0.0013	+0.0010	-0.0013	+0.0017	-0.0007	+0.0015	-0.0011
Avg. diff.	Soup	**+0.0037**	**-0.0025**	**+0.0026**	**-0.0028**	**+0.0032**	**-0.0009**	**+0.0032**	**-0.0021**

5.2 What drives the generalization gain?

Table 6 presents a set of DNN-based controls for analyzing the source of the gains from embedding soups. All methods use the DNN backbone. For Embedding Soups with the default soup size $N = 4$, we first align the training budget: *end-to-end, long-train* extends end-to-end training to match the total soup training time as closely as possible (Appendix B.1), while *end-to-end, best-of-restarts* performs multiple independent end-to-end runs under the same training budget and reports the run with the best validation performance. We then align embedding parameter capacity: the capacity controls use a single model and enlarge the embedding capacity by a factor of N under both end-to-end training and embedding retuning, matching the total embedding parameter scale induced by N retuned embeddings. Table 6 reports the stronger $N = 8$ control, while the full $N = 4$ and $N = 8$ results are provided in Appendix B.1. We also evaluate standard regularization by selecting the best end-to-end result from sweeps over dropout, weight decay, and early stopping. Additional controls apply dropout and Gaussian noise directly to embeddings during end-to-end training (Appendix B.1). Finally, *Best-of-N retuning* selects the single retuned embedding with the best validation performance from the same $N = 4$ retuning runs used by Embedding Soups.

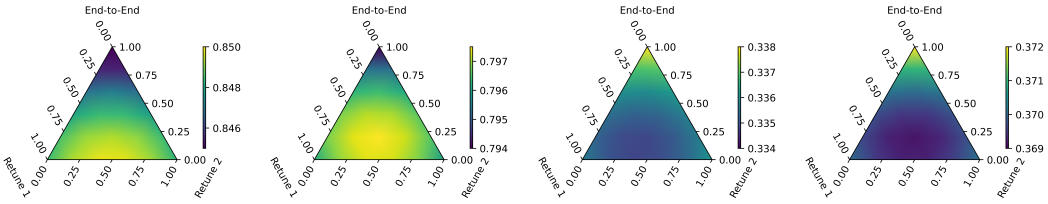
Across datasets, these controls remain below embedding soups. The results indicate that the gains are not explained by additional training budget, larger embedding capacity, standard regularization, or selecting a favorable single retuning run; instead, the averaging operation over compatible retuned embeddings is essential. Embedding Soups also outperforms these embedding-perturbation baselines, at a higher offline computational cost.

Figure 2 further connects the empirical controls to the component-wise and variance-based interpretations. Under the same fixed FI module, we visualize the AUC and logloss landscapes over convex combinations of one end-to-end embedding table and two independently retuned embedding

Table 6: DNN controls for alternative explanations.

Method	Avazu		Criteo		Taobao Ads	
	AUC ↑	logloss ↓	AUC ↑	logloss ↓	AUC ↑	logloss ↓
end-to-end	0.7942	0.3716	0.8096	0.4424	0.6356	0.1948
end-to-end, long-train	0.7943	0.3712	0.8094	0.4428	0.6358	0.1945
end-to-end, best-of-restarts	0.7943	0.3712	0.8096	0.4423	0.6359	0.1945
end-to-end, larger emb.	0.7946	0.3711	0.8097	0.4422	0.6357	0.1949
embedding retuning, larger emb.	0.7960	0.3702	0.8110	0.4407	0.6379	0.1937
Best regularization	0.7943	0.3712	0.8094	0.4428	0.6360	0.1943
Best-of-N retuning	0.7962	0.3701	0.8105	0.4413	0.6383	0.1937
Embedding soups	**0.7980**	**0.3689**	**0.8120**	**0.4397**	**0.6393**	**0.1936**

tables. The interior of the simplex remains in a high-AUC and low-logloss region, indicating that fixed-FI retuning produces embedding estimates that are compatible with the same fixed FI module rather than isolated solutions. This observation supports the asymmetric diagnosis in Section 3: the FI module provides a stable reusable structure, whereas the embedding side admits multiple compatible estimates that can be selectively aggregated. It is also consistent with the local variance interpretation in Section 4.4, where averaging reduces run-specific embedding fluctuations without moving the model into a high-risk region. Appendix C.1.1 compares cross-seed block swaps and embedding retuning on Avazu with DNN.



(a) Train AUC (b) Test AUC (c) Train logloss (d) Test logloss

Figure 2: Ternary plots for the AUC and logloss landscapes of embedding parameters. The three vertices represent $\mathbf{E}_0$, $\mathbf{E}^{(1)}$, and $\mathbf{E}^{(2)}$; each interior point corresponds to a convex combination of the three embedding tables under the same FI module.

Before comparing with existing one-epoch remedies, we clarify the distinction between Embedding Soups and standard ensembling or full-model soups. Prediction ensembling combines multiple deployed models and therefore increases online serving cost, whereas Embedding Soups averages independently retuned embedding tables before deployment and keeps a single inference path (a direct latency comparison is reported in Appendix B.2). Full-model soups average all model parameters, while our method keeps the FI module fixed and averages only the embedding block. Thus, the distinction lies not only in deployment cost, but also in the operated component: Embedding Soups is a block-selective intervention motivated by the asymmetric generalization diagnosis in Section 3.

We next examine whether Embedding Soups is complementary to existing one-epoch mitigation methods. We first compare against AdamAR Li and Lyu [2026], a recent adaptive regularization method for large-scale sparse feature embedding models. On Avazu with DNN, AdamAR yields only a small improvement over standard end-to-end training ($0.7943/0.3713$ vs. $0.7942/0.3716$ for AUC/logloss), while Embedding Soups gives a larger gain ($0.7980/0.3689$). Combining AdamAR with Embedding Soups further improves the result to $0.7981/0.3688$, suggesting that frequency-aware regularization and component-selective averaging address different sources of generalization degradation. This is consistent with their roles: AdamAR regularizes a single training trajectory, whereas Embedding Soups reduces run-specific embedding fluctuations across independently retuned embeddings under a fixed FI module. We also evaluate compatibility with strong multi-epoch baselines, including MEDA and ME. Embedding Soups further improves these methods in several settings, indicating that it is complementary to existing one-epoch mitigation approaches. The benefit depends on the configuration: adding Embedding Soups to ME+MEDA reduces AUC from

0.8011 to 0.7982 for FinalMLP on Avazu with embedding dimension 64. Detailed AdamAR and MEDA/ME results are reported in Appendices C.2 and C.3.

5.3 Deployment cost

Embedding soups preserve single-model deployment. Table 7 separates online and offline costs. Online, soups have latency comparable to end-to-end training because the averaged embeddings are deployed as a single embedding table with the same FI module. Offline, soups require additional retuning, leading to a total training time that is 3.72 to 5.16 times that of standard end-to-end training. The peak GPU memory also grows moderately rather than linearly with the number of soup ingredients: Avazu-DNN uses 2.67GB versus 1.61GB for end-to-end training, and Criteo-DNN uses 0.70GB versus 0.42GB. Thus, the main practical limitation of Embedding Soups is additional offline retuning, making it most suitable when online serving cost is the primary constraint; detailed per-epoch measurements are given in Appendix B.2.

Table 7: Online and offline cost summary for DNN.

Dataset	end-to-end online latency (ms)	Soup online latency (ms)	Soup total training time relative to end-to-end
Avazu	0.5871	0.6124	5.16
Criteo	1.0923	1.1021	3.72
Taobao Ads	0.4634	0.4496	4.96

6 Related work

Reinitialization Researchers have recently drawn more attention to reinitializing subnetworks as an effective technique to improve neural network training Ash and Adams [2020], Alabdulmohsin et al. [2021], Zaidi et al. [2023]. Reinitialization also helps improve generalization and maintain neural plasticity in deep reinforcement learning Igl et al. [2021], Nikishin et al. [2022], D'Oro et al. [2022], Schwarzer et al. [2023]. Recent study Fan et al. [2024] adopts reinitialization on the embedding layer in each epoch to perform data augmentation and enables multi-epoch training on CTR tasks. Experiments and further discussion around this research are available in App. C.3.

Weight averaging and ensembling Weight averaging is a long-standing approach in optimization and deep learning Polyak and Juditsky [1992], with representative variants including stochastic weight averaging, FedAvg, and model soups Izmailov et al. [2018], McMahan et al. [2017], Wortsman et al. [2022]; related work also studies when parameter averaging is valid through linear mode connectivity Frankle et al. [2020]. Prediction ensembling improves performance by combining multiple deployed models, and ensemble-style designs are common in deep CTR models Wang et al. [2021], Mao et al. [2023], Guo et al. [2023], but they increase online serving cost. Our method differs from both lines: it averages only independently retuned embeddings under a fixed FI module, and deploys the averaged embedding table as a single CTR model.

7 Conclusion

In this paper, we revisit the one-epoch phenomenon in deep CTR models from a component-wise generalization perspective. Through block-wise retuning, we show that retaining the learned FI state supports better held-out generalization under counterpart re-estimation than retaining embeddings. Motivated by this asymmetry, we propose embedding soups, a component-selective averaging method that aggregates independently retuned embeddings under a fixed interaction module and improves generalization without increasing online inference cost. Experiments across multiple CTR datasets and architectures validate the effectiveness of this approach. The epoch-boundary interventions further point to asymmetric coupling between the jointly trained blocks in the one-epoch phenomenon.

Acknowledgments and Disclosure of Funding

Funding. This work was supported by the Guangdong Science and Technology Program (Grant No. 2024B0101040005), the National Natural Science Foundation of China (Grant No. 12671517), and the Natural Science Foundation of Guangdong Province (Grant No. 2026A1515010335).

Competing interests. The authors declare no competing interests.

References

Ibrahim Alabdulmohsin, Hartmut Maennel, and Daniel Keysers. The impact of reinitialization on generalization in convolutional neural networks. *arXiv preprint arXiv:2109.00267*, 2021.

Jordan Ash and Ryan P Adams. On warm-starting neural network training. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3884–3894. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/288cd2567953f06e460a33951f55daaf-Paper.pdf.

Avazu. Click-through rate prediction. https://www.kaggle.com/c/avazu-ctr-prediction, 2014.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/chen20j.html.

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, pages 7–10, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450347952. doi: 10.1145/2988450.2988454. URL https://doi.org/10.1145/2988450.2988454.

Criteo Labs. Display advertising challenge. https://ailab.criteo.com/display-advertising-challenge-criteo/, 2014.

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.

Pierluca D'Oro, Max Schwarzer, Evgenii Nikishin, Pierre-Luc Bacon, Marc G Bellemare, and Aaron Courville. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

Zhongxiang Fan, Zhaocheng Liu, Jian Liang, Dongying Kong, Han Li, Peng Jiang, Shuang Li, and Kun Gai. Multi-epoch learning with data augmentation for deep click-through rate prediction. *arXiv preprint arXiv:2407.01607*, 2024.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3259–3269. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/frankle20a.html.

Zichuan Fu, Xiangyang Li, Chuhan Wu, Yichao Wang, Kuicai Dong, Xiangyu Zhao, Mengchen Zhao, Huifeng Guo, and Ruiming Tang. A unified framework for multi-domain ctr prediction via large language models. *ACM Transactions on Information Systems*, 43(5):1–33, 2025.

Raphael Gontijo-Lopes, Yann Dauphin, and Ekin D Cubuk. No one representation to rule them all: Overlapping features of training methods. *arXiv preprint arXiv:2110.12899*, 2021.

Huifeng Guo, Ruiming TANG, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for ctr prediction. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 1725–1731, 2017. doi: 10.24963/ijcai.2017/239. URL https://doi.org/10.24963/ijcai.2017/239.

Xingzhuo Guo, Junwei Pan, Ximei Wang, Baixu Chen, Jie Jiang, and Mingsheng Long. On the embedding collapse when scaling up recommendation models. *arXiv preprint arXiv:2310.04400*, 2023.

Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Qun8fv4qSby.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015. URL http://arxiv.org/abs/1412.6980.

Mang Li and Wei Lyu. Adaptive regularization for large-scale sparse feature embedding models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=QFH5mwP9oH.

Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. Xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 1754–1763, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3220023. URL https://doi.org/10.1145/3219819.3220023.

Kelong Mao, Jieming Zhu, Liangcai Su, Guohao Cai, Yuru Li, and Zhenhua Dong. Finalmlp: An enhanced two-stream mlp model for ctr prediction. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(4):4552–4560, Jun. 2023. doi: 10.1609/aaai.v37i4.25577. URL https://ojs.aaai.org/index.php/AAAI/article/view/25577.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 20–22 Apr 2017. URL https://proceedings.mlr.press/v54/mcmahan17a.html.

Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? *Advances in neural information processing systems*, 33:512–523, 2020.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 16828–16847. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/nikishin22a.html.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.

Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. Product-based neural networks for user response prediction. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 1149–1154, 2016. doi: 10.1109/ICDM.2016.0151.

Steffen Rendle. Factorization machines. In *2010 IEEE International conference on data mining*, pages 995–1000. IEEE, 2010.

David Ruppert. Efficient estimations from a slowly convergent robbins-monro process. Technical report, Cornell University Operations Research and Industrial Engineering, 1988.

Max Schwarzer, Johan Samir Obando Ceron, Aaron Courville, Marc G Bellemare, Rishabh Agarwal, and Pablo Samuel Castro. Bigger, better, faster: Human-level atari with human-level efficiency. In *International Conference on Machine Learning*, pages 30365–30380. PMLR, 2023.

Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. Deep crossing: Web-scale modeling without manually crafted combinatorial features. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 255–262, 2016.

Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1161–1170, 2019.

Tianchi. Ad display/click data on taobao.com. `https://tianchi.aliyun.com/dataset/56`, 2018.

Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the Web Conference 2021*, WWW '21, pages 1785–1797, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383127. doi: 10.1145/3442381.3450078. URL `https://doi.org/10.1145/3442381.3450078`.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 23965–23998. PMLR, 17–23 Jul 2022. URL `https://proceedings.mlr.press/v162/wortsman22a.html`.

Sheheryar Zaidi, Tudor Berariu, Hyunjik Kim, Jorg Bornschein, Claudia Clopath, Yee Whye Teh, and Razvan Pascanu. When does re-initialization work? In Javier Antorán, Arno Blaas, Fan Feng, Sahra Ghalebikesabi, Ian Mason, Melanie F. Pradier, David Rohde, Francisco J. R. Ruiz, and Aaron Schein, editors, *Proceedings on "I Can't Believe It's Not Better! - Understanding Deep Learning Through Empirical Falsification" at NeurIPS 2022 Workshops*, volume 187 of *Proceedings of Machine Learning Research*, pages 12–26. PMLR, 03 Dec 2023. URL `https://proceedings.mlr.press/v187/zaidi23a.html`.

Zhao-Yu Zhang, Xiang-Rong Sheng, Yujing Zhang, Biye Jiang, Shuguang Han, Hongbo Deng, and Bo Zheng. Towards understanding the overfitting phenomenon of deep click-through rate models. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, CIKM '22, pages 2671–2680, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392365. doi: 10.1145/3511808.3557479. URL `https://doi.org/10.1145/3511808.3557479`.

Jieming Zhu, Jinyang Liu, Shuai Yang, Qi Zhang, and Xiuqiang He. Open benchmarking for click-through rate prediction. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, pages 2759–2769, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384469. doi: 10.1145/3459637.3482486. URL `https://doi.org/10.1145/3459637.3482486`.

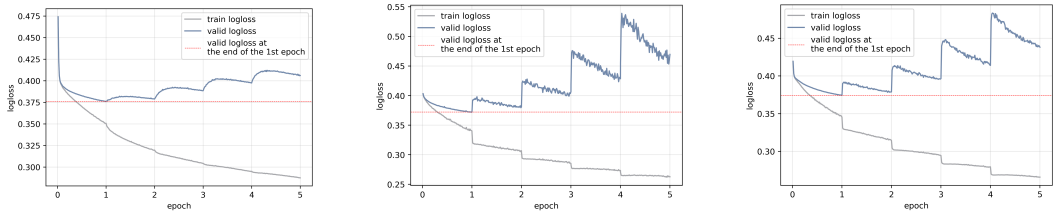
A Additional experimental setup

A.1 Representative one-epoch behavior

Table 8 summarizes representative CTR model families by their feature representation and interaction modules. Figure 3 shows representative Avazu training curves for FM, DNN, and DCN-v2. These background results illustrate that early validation peaks are observed across models with different feature-interaction capacities.

Table 8: Representative CTR model families. FR denotes feature representation, FI denotes feature interaction, and OEP denotes the observed severity of the one-epoch phenomenon in representative Avazu training curves.

Model	FR	FI	#Params of FI	OEP
LR	scalar emb.	sum	1	No
FM Rendle [2010]	vector emb.	IP & sum	1	Mild
DNN Cheng et al. [2016]	vector emb.	MLP	$\|\mathbf{W}_{\text{mlp}}\|$	Severe
DCN-v2 Wang et al. [2021]	vector emb.	cross layers	$\|\mathbf{W}_{\text{cross}}\|$	Severe



(a) FM (b) DNN (c) DCN-v2

Figure 3: Representative training curves on Avazu. Validation performance peaks early and then degrades for models with different feature-interaction modules.

A.2 Dataset and implementation details

We conduct experiments on Avazu, Criteo, and Taobao Ads. For Avazu and Criteo, we use the preprocessed files and random 8:1:1 train/validation/test split from FuxiCTR Zhu et al. [2021]. For Taobao Ads, we use the first seven days for training and the last day for testing, without behavior logs. Criteo numerical features are discretized by $\lfloor \log_2(x) \rfloor$. Infrequent categorical values are replaced with OOV tokens using thresholds 2, 10, and 1 for Avazu, Criteo, and Taobao Ads.

All models are implemented in PyTorch Paszke et al. [2019] and optimized with Adam Kingma and Ba [2015]. The learning rate is 10^{-3} for Avazu and Criteo and 3×10^{-5} for Taobao Ads. The embedding size is 16 and hidden layers are fixed to 1024-512-256 unless otherwise specified. We follow the remaining model-specific hyperparameters from FuxiCTR Zhu et al. [2021]. All experiments can be run on a single RTX 3090 GPU.

Table 9: Statistics of the CTR prediction datasets. "#Features" denotes valid sparse feature values after thresholding and preprocessing.

Dataset	#Instances	#Fields	#Features	%Positives
Avazu	40M	24	4.4M	17%
Criteo	46M	39	1.1M	26%
Taobao Ads	27M	23	2.8M	5.14%

Supplementary experiment settings. The cross-architecture diagnostic, epoch-boundary interventions, and additional Avazu–DCN-v2 logloss comparison use three independent runs. The other

comparisons use one base-model run. Embedding Soups averages four independently retuned embedding tables. In the tables, E2E denotes end-to-end training, EmbRT denotes embedding retuning, FIRT denotes FI retuning, and Soup denotes Embedding Soups.

A.3 Best epochs for main results

Table 10 reports the best epochs corresponding to the main results in Table 5. The results show that all embedding-retuning ingredients and all soups peak at epoch 1, while end-to-end training also peaks very early.

Table 10: Best epochs for the main results in Table 5. RT ingredients reports the four independently retuned embedding ingredients used to form the soup; "all 1" means that all four ingredients select epoch 1.

Dataset	Model	end-to-end	RT ingredients	Soup
Avazu	DNN	1	all 1	1
	DeepFM	1	all 1	1
	IPNN	1	all 1	1
	DCN-v2	1	all 1	1
	FinalMLP	1	all 1	1
Criteo	DNN	2	all 1	1
	DeepFM	1	all 1	1
	IPNN	1	all 1	1
	DCN-v2	2	all 1	1
	FinalMLP	2	all 1	1
Taobao Ads	DNN	1	all 1	1
	DeepFM	1	all 1	1
	IPNN	1	all 1	1
	DCN-v2	1	all 1	1
	FinalMLP	1	all 1	1

B Additional experimental analyses

B.1 Alternative explanations and control experiments

The main text reports a compact DNN control summary in Table 6. This subsection expands those controls with additional settings and model variants.

Table 11 gives the full budget and capacity controls used to check whether the gains of soups are explained by longer training, more restarts, or larger embedding capacity. Compared with the compact main table, it additionally includes the $k = 4$ capacity setting. Long-train extends end-to-end training to approximately match the total soup training time, best-of-restarts selects the best end-to-end run under repeated restarts with a larger budget, and k is the embedding-capacity multiplier.

Completed training schedules The long-train control continues end-to-end training for eight epochs beyond the corresponding standard run. Early stopping is disabled, so all scheduled epochs are completed before a checkpoint is selected using validation performance. The control completes 10 epochs on Avazu, 11 on Criteo, and 10 on Taobao. Validation selects the checkpoint from epoch 1 on Avazu and Taobao and from epoch 2 on Criteo. Table 12 separates the number of completed epochs from the selected checkpoint. The schedules use measured per-epoch runtime on the same hardware to approximate the sequential GPU time required for one end-to-end run and four embedding-retuning runs.

Table 13 expands the best-regularization row in Table 6 to additional architectures. The baseline is selected by validation AUC from dropout, weight decay, and early stopping sweeps.

Training-phase embedding perturbations During end-to-end training, we apply perturbations after the embedding lookup and before the FI module. Dropout is applied independently to each

Table 11: Training-budget and embedding-capacity controls for DNN. Long-train approximately matches soup training time; best-of-restarts reports the best validation run among repeated end-to-end restarts. RT denotes embedding retuning; k is the embedding-capacity multiplier.

Method	Avazu		Criteo		Taobao Ads	
	AUC ↑	logloss ↓	AUC ↑	logloss ↓	AUC ↑	logloss ↓
end-to-end, long-train	0.7943	0.3712	0.8094	0.4428	0.6358	0.1945
end-to-end, best-of-restarts	0.7943	0.3712	0.8096	0.4423	0.6359	0.1945
end-to-end, $k = 4$	0.7942	0.3714	0.8097	0.4423	0.6358	0.1943
end-to-end, $k = 8$	0.7946	0.3711	0.8097	0.4422	0.6357	0.1949
RT, $k = 4$	0.7960	0.3701	0.8106	0.4411	0.6378	0.1938
RT, $k = 8$	0.7960	0.3702	0.8110	0.4407	0.6379	0.1937
Embedding soups	**0.7980**	**0.3689**	**0.8120**	**0.4397**	**0.6393**	**0.1936**

Table 12: Training schedules for the DNN control. "Completed" is the number of epochs actually executed. "Selected" is the epoch of the checkpoint chosen by validation performance.

	Standard E2E		Long-train		Added
Dataset	Completed	Selected	Completed	Selected	epochs
Avazu	2	1	10	1	8
Criteo	3	2	11	2	8
Taobao	2	1	10	1	8

field's embedding tensor. Gaussian noise is sampled independently with zero mean at every training mini-batch. Both perturbations are disabled during evaluation. For DNN on Avazu, we test dropout probabilities of 0.05 and 0.1; the other settings use 0.1, as in the regularization controls. We test Gaussian-noise standard deviations of 0.005 and 0.01. Hyperparameters are selected using validation performance.

Table 15 reports test logloss for the same configurations. Embedding Soups performs better than the tested perturbations in these settings. Gaussian noise nevertheless improves the DNN model on Taobao and may offer a useful alternative when computational resources are limited. Each perturbation baseline requires one end-to-end training run, whereas Embedding Soups requires one end-to-end run followed by four embedding-retuning runs. Relative to standard end-to-end training, the measured sequential GPU time is 1.01–1.15 times as long for dropout, 1.00–2.36 times as long for Gaussian noise, and 3.72–5.16 times as long for Embedding Soups. Parallel execution can reduce wall-clock time while retaining the same total GPU workload.

Table 16 expands the best-of-N comparison in Table 6 by adding validation-weighted and Fisher-weighted variants under the same fixed-FI retuning protocol. Weighted variants do not show stable advantages over uniform averaging, while all soup variants outperform best-of-N selection.

Table 13: Best regularization baseline vs. embedding soups. The best regularization result is selected separately for each model and dataset by validation AUC.

Method	Avazu		Criteo		Taobao Ads	
	AUC ↑	logloss ↓	AUC ↑	logloss ↓	AUC ↑	logloss ↓
DNN: best reg.	0.7943	0.3712	0.8094	0.4428	0.6360	0.1943
DNN: soups	**0.7980**	**0.3689**	**0.8120**	**0.4397**	**0.6393**	**0.1936**
DCN-v2: best reg.	0.7945	0.3712	0.8105	0.4416	0.6358	0.1945
DCN-v2: soups	**0.7979**	**0.3689**	**0.8125**	**0.4393**	**0.6392**	**0.1936**
FinalMLP: best reg.	0.7950	0.3708	0.8096	0.4431	0.6354	0.1945
FinalMLP: soups	**0.7988**	**0.3683**	**0.8126**	**0.4393**	**0.6366**	**0.1942**

Table 14: Training-phase perturbation comparisons. Values are test AUC; selected dropout probabilities and noise standard deviations appear in parentheses.

Setting	E2E	Dropout	Gaussian noise	EmbRT	Soup
Avazu–DNN	0.7943	0.7935 (0.05)	0.7937 (0.005)	0.7960	0.7981
Avazu–DCN-v2	0.7945	0.7933 (0.1)	0.7938 (0.005)	0.7961	0.7979
Taobao–DNN	0.6358	0.6358 (0.1)	0.6373 (0.01)	0.6379	0.6394
Criteo–DNN	0.8094	0.8094 (0.1)	0.8093 (0.01)	0.8104	0.8120

Table 15: Test logloss for the same perturbation settings as Table 14; lower is better.

Setting	E2E	Dropout	Gaussian noise	EmbRT	Soup
Avazu–DNN	0.3712	0.3717	0.3716	0.3701	0.3688
Avazu–DCN-v2	0.3712	0.3720	0.3715	0.3701	0.3689
Taobao–DNN	0.1945	0.1945	0.1940	0.1937	0.1936
Criteo–DNN	0.4428	0.4428	0.4430	0.4414	0.4398

Table 17 further checks Best-of-N selection on DCN-v2 and FinalMLP. Together with the DNN comparison in Table 16, the results show that soups consistently outperform selecting the best single retuned embedding across architectures.

Selection, parameter averaging, and prediction ensembling Table 18 compares three ways to use the same four independently retuned embedding tables. Best-of-N selects one table using validation performance. Embedding Soups averages the tables in parameter space and deploys the resulting single model. Prediction ensembling instead averages the outputs of four models, each using one retuned table and the shared FI module. Prediction ensembling can perform slightly better, but it requires evaluating multiple models at inference time (Appendix B.2).

The matched-capacity control changes the parameter allocation by reducing embedding-like parameters and increasing FI parameters, making the embedding/FI parameter ratio less skewed than in the default DNN. Table 19 shows that soups remain better than matched end-to-end training and matched retuning under this alternative allocation.

B.2 Training and deployment cost

Table 7 summarizes the deployment trade-off in the main text. Table 20 gives the expanded DNN training and inference cost, including per-epoch runtime and the single-retuning protocol. The per-epoch runtime is similar across end-to-end training, retuning, and soups; the extra cost of soups mainly comes from repeated offline retuning runs and temporary checkpoint storage. At deployment, the final soup is still a single model with one embedding table and one FI module.

We also compare with prediction ensembling of the same retuned models. The ensemble reaches AUCs of 0.7983, 0.8120, and 0.6395 on Avazu, Criteo, and Taobao Ads, close to the uniform soup AUCs of 0.7980, 0.8120, and 0.6393. However, its online latency is about $3.26\times$, $4.66\times$, and $3.92\times$ that of soups on the three datasets.

Table 16: Aggregation comparison for DNN under the same fixed-FI retuning protocol. Best-of-N selects the best single retuned embedding; the soup variants average the same candidate embeddings using different weights.

Method	Avazu		Criteo		Taobao Ads	
	AUC ↑	logloss ↓	AUC ↑	logloss ↓	AUC ↑	logloss ↓
Best-of-N	0.7962	0.3701	0.8105	0.4413	0.6383	0.1937
Validation-weighted soup	0.7975	0.3692	0.8118	0.4401	**0.6393**	**0.1936**
Fisher soup	**0.7980**	**0.3689**	0.8120	0.4399	**0.6393**	**0.1936**
Uniform soup	**0.7980**	**0.3689**	**0.8120**	**0.4397**	**0.6393**	**0.1936**

Table 17: Best-of-N retuning versus embedding soups on additional architectures. Best-of-N selects the best single retuned embedding from the same candidate set used by soups.

Method	Avazu		Criteo		Taobao Ads	
	AUC ↑	logloss ↓	AUC ↑	logloss ↓	AUC ↑	logloss ↓
DCN-v2: Best-of-N	0.7961	0.3701	0.8109	0.4408	0.6379	0.1937
DCN-v2: soups	**0.7979**	**0.3689**	**0.8125**	**0.4393**	**0.6392**	**0.1936**
FinalMLP: Best-of-N	0.7970	0.3695	0.8109	0.4407	0.6352	0.1945
FinalMLP: soups	**0.7988**	**0.3683**	**0.8126**	**0.4393**	**0.6366**	**0.1942**

Table 18: Aggregation methods using the same four retuned embedding tables. Avazu reports test logloss (lower is better); Criteo and Taobao report test AUC (higher is better).

Dataset	FI	Best-of-N	Soup	Prediction ensemble
Avazu	DNN	0.3701	0.3688	0.3687
	DCN-v2	0.3701	0.3689	0.3688
	FinalMLP	0.3695	0.3684	0.3683
Criteo	DNN	0.8105	0.8120	0.8120
	DCN-v2	0.8109	0.8123	0.8124
	FinalMLP	0.8109	0.8125	0.8125
Taobao	DNN	0.6383	0.6394	0.6395
	DCN-v2	0.6379	0.6392	0.6393
	FinalMLP	0.6352	0.6366	0.6368

Table 19: Matched-capacity control for DNN. All rows use the same matched parameter allocation; only the training protocol differs.

Method	Avazu		Criteo		Taobao Ads	
	AUC ↑	logloss ↓	AUC ↑	logloss ↓	AUC ↑	logloss ↓
Matched end-to-end	0.7931	0.3720	0.8094	0.4423	0.6354	0.1953
Matched retuning	0.7948	0.3709	0.8104	0.4413	0.6357	0.1941
Matched soups	**0.7967**	**0.3697**	**0.8117**	**0.4400**	**0.6375**	**0.1939**

Table 20: Training and deployment cost for DNN. Sec/epoch measures offline training speed, total time is normalized by standard end-to-end training, and latency is online inference time for the deployed model.

Dataset	Protocol	sec/epoch	total time (× end-to-end)	latency (ms)
Avazu	end-to-end	91.6	1.00	0.5871
	retuning	95.0	2.07	0.6219
	soup	94.5	5.16	0.6124
Criteo	end-to-end	118.3	1.00	1.0923
	retuning	118.4	1.67	1.1118
	soup	120.1	3.72	1.1021
Taobao Ads	end-to-end	45.7	1.00	0.4634
	retuning	46.6	2.04	0.4561
	soup	45.3	4.96	0.4496

B.3 Additional component-wise controls

The compact Avazu summary in Table 4 is expanded in Tables 21 and 22, which include best epoch and logloss for all settings. Tables 23–26 report the same two controls on Criteo and Taobao Ads. In the frequency-threshold tables, a column marked "retune@r" means that embeddings are retuned using features thresholded at r, while the FI module was trained using the column's main threshold. These results lead to the following observations:

- **Partial FI training can already support embedding retuning.** In Table 23, the FI module trained on 20% of the data already allows embedding retuning to improve validation performance, supporting the FI reuse observed in Section 3.2.
- **Criteo is less dependent on the sparsest feature values.** In Table 24, the two FI-threshold-2 columns, retune@2 and retune@10, have similar performance, and the threshold-100/250 settings can even outperform the default threshold-10 setting after retuning.
- **Extremely sparse features can hurt FI learning in some settings.** Although pruning high-sparsity features usually weakens end-to-end models, Table 24 shows that FI modules trained after moderate pruning can support stronger embedding retuning.

Since model performance on Criteo does not heavily rely on extremely sparse features, embedding soups also show a smaller, though still statistically significant, improvement on Criteo than on Avazu and Taobao Ads. The threshold-1000 column in Table 24 also suggests that the benefit of embedding retuning can diminish once too many sparse feature values are removed.

Table 21: Effect of FI training ratio on Avazu. The FI module is first trained with the indicated fraction of training instances, then embeddings are retuned on the full training set with that FI module fixed. The 0% column corresponds to no FI pretraining.

Training ratio	0%	20%	40%	60%	80%	100%
end-to-end best epoch	–	1	1	1	1	1
end-to-end AUC ↑	–	0.7681	0.7783	0.7848	0.7905	0.7942
end-to-end logloss ↓	–	0.3865	0.3810	0.3770	0.3736	0.3716
Retuned best epoch	8	1	1	1	1	1
Retuned AUC ↑	0.7821	0.7904	0.7930	0.7946	0.7955	0.7960
Retuned logloss ↓	0.3788	0.3738	0.3721	0.3711	0.3705	0.3701

Table 22: Effect of feature-frequency threshold on Avazu. Larger thresholds prune more sparse feature values before FI training; embeddings are then retuned under the resulting fixed FI module.

Frequency threshold	2	10	25	100	250
end-to-end best epoch	1	2	2	3	5
end-to-end AUC ↑	0.7942	0.7822	0.7745	0.7693	0.7679
end-to-end logloss ↓	0.3716	0.3791	0.3829	0.3857	0.3864
Retuned best epoch	1	1	1	1	1
Retuned AUC ↑	0.7960	0.7960	0.7962	0.7958	0.7933
Retuned logloss ↓	0.3701	0.3703	0.3701	0.3704	0.3721

B.3.1 Diagnostics across architectures and FI-retuning configurations

Table 2 reports test-AUC changes for DeepFM, IPNN, DCN-v2, and FinalMLP. For the same architectures, the mean changes in validation AUC from embedding retuning are $+0.0019$, $+0.0016$, $+0.0015$, and $+0.0019$, respectively. The corresponding changes for FI retuning are -0.0179, -0.0173, -0.0185, and -0.0192. Each change is measured relative to the corresponding end-to-end model. These comparisons follow the block-wise retuning protocol in Section 3.1.

Formalization of the diagnostic. Let A_E and A_W denote the finite-budget optimization procedures that reinitialize and train one block while retaining its counterpart:

$$\mathbf{E}_{\mathrm{rt}} = A_E(\mathbf{W}_0), \qquad \mathbf{W}_{\mathrm{rt}} = A_W(\mathbf{E}_0). \tag{1}$$

Table 23: Effect of FI training ratio on Criteo. The DNN FI module is trained end-to-end using the indicated fraction of training instances, and the embedding-retuning rows then retune embeddings on the full training set with that FI fixed. Training is capped at 10 epochs.

FI training ratio		0%	20%	40%	60%	80%	100%
end-to-end	best epoch	-	1	1	1	2	2
	AUC ↑	-	0.7978	0.8030	0.8059	0.8080	0.8096
	logloss ↓	-	0.4533	0.4480	0.4458	0.4437	0.4424
Emb. retuning	best epoch	10	2	1	1	1	1
	AUC ↑	0.8024	0.8051	0.8075	0.8092	0.8101	0.8104
	logloss ↓	0.4487	0.4467	0.4440	0.4424	0.4416	0.4412

Table 24: Effect of feature-frequency threshold on Criteo. The main column header is the threshold used for FI training; retune@r indicates that embeddings are retuned using features thresholded at r. The default Criteo threshold is 10.

FI threshold		2 (retune@2)	2 (retune@10)	10	100	250	1000
end-to-end	best epoch	1	1	2	3	3	4
	AUC ↑	0.8091		0.8096	0.8101	0.8096	0.8076
	logloss ↓	0.4424		0.4424	0.4415	0.4420	0.4438
Emb. retuning	best epoch	1	1	1	1	1	1
	AUC ↑	0.8101	0.8099	0.8104	0.8112	0.8115	0.8077
	logloss ↓	0.4416	0.4417	0.4412	0.4405	0.4402	0.4440

Table 25: Effect of FI training ratio on Taobao Ads. The DNN FI module is trained end-to-end using the indicated fraction of training instances, and the embedding-retuning rows then retune embeddings on the full training set with that FI fixed. Training is capped at 10 epochs.

FI training ratio		0%	20%	40%	60%	80%	100%
end-to-end	best epoch	-	1	1	1	1	1
	AUC ↑	-	0.5943	0.6121	0.6205	0.6286	0.6356
	logloss ↓	-	0.2148	0.1968	0.1967	0.1953	0.1948
Emb. retuning	best epoch	10	1	1	1	1	1
	AUC ↑	0.4891	0.6345	0.63749	0.63760	0.63787	0.6379
	logloss ↓	0.2239	0.1946	0.19381	0.19376	0.19374	0.1937

Table 26: Effect of feature-frequency threshold on Taobao Ads. The main column header is the threshold used for FI training; retune@r indicates that embeddings are retuned using features thresholded at r. The default Taobao Ads threshold is 1.

FI threshold		1	10 (retune@1)	10 (retune@10)	25	100	250
end-to-end	best epoch	1	1		1	3	6
	AUC ↑	0.6356	0.63395		0.63105	0.62149	0.60926
	logloss ↓	0.1948	0.19483		0.19480	0.19555	0.19612
Emb. retuning	best epoch	1	1	1	1	1	1
	AUC ↑	0.6379	0.63749	0.63660	0.63713	0.63747	0.63730
	logloss ↓	0.1937	0.19385	0.19395	0.19394	0.19385	0.19392

Embedding retuning (EmbRT) evaluates $(\mathbf{E}_{\mathrm{rt}}, \mathbf{W}_0)$; FI retuning (FIRT) evaluates $(\mathbf{E}_0, \mathbf{W}_{\mathrm{rt}})$. We define

$$\begin{aligned} C_W &= \mathrm{AUC}_{\mathrm{val}}(\mathbf{E}_{\mathrm{rt}}, \mathbf{W}_0), \quad G_W = \mathrm{AUC}_{\mathrm{train}}(\mathbf{E}_{\mathrm{rt}}, \mathbf{W}_0) - C_W, \\ C_E &= \mathrm{AUC}_{\mathrm{val}}(\mathbf{E}_0, \mathbf{W}_{\mathrm{rt}}), \quad G_E = \mathrm{AUC}_{\mathrm{train}}(\mathbf{E}_0, \mathbf{W}_{\mathrm{rt}}) - C_E. \end{aligned} \tag{2}$$

The scores C_W and C_E measure validation performance when the FI or embedding parameters are retained, respectively; G_W and G_E are the corresponding train–validation AUC gaps of the complete models.

We examine the sensitivity of FI retuning to its optimization configuration by varying its learning-rate multiplier over $\{0.3, 1, 3\}$ relative to the Avazu base learning rate of 10^{-3}. For DCN-v2, we also vary the maximum training duration over $\{1, 2, 4\}$ epochs. Table 27 reports the FI-retuning configuration selected by validation AUC and compares it with embedding retuning. The selected FI-retuning models achieve higher training AUC but lower validation AUC. Thus, the poorer validation performance accompanies stronger training fit and a larger train–validation gap.

Table 27: FI-retuning hyperparameter comparison on Avazu. "Best FIRT" denotes the configuration with the highest validation AUC among the tested candidates. Gap denotes training AUC minus validation AUC.

FI	Protocol	Train AUC	Validation AUC	Test AUC	Gap
DNN	EmbRT	0.8491	0.7959	0.7960	0.0532
DNN	Best FIRT	0.8697	0.7758	0.7763	0.0939
DCN-v2	EmbRT	0.8479	0.7958	0.7961	0.0520
DCN-v2	Best FIRT	0.8709	0.7766	0.7770	0.0943

Epoch-boundary protocol. All interventions in Table 3 start from the same parameters $(\mathbf{E}_1, \mathbf{W}_1)$ obtained at the end of the first epoch. The continued-training interventions preserve both blocks and update either both, FI only, or embeddings only. The reinitialization interventions reinitialize and train either the embeddings or FI while freezing the other block. Each intervention uses the same training budget for the second pass through the data. Validation-AUC changes are measured at the end of that pass relative to the unmodified first-epoch model.

In the three settings where continued joint training reduces validation AUC, reinitializing and retraining the embeddings is the only intervention that improves on the unmodified checkpoint. Updating FI alone can cause a larger decline. In contrast, joint training improves AUC for DNN on Criteo. These findings support an interpretation in terms of asymmetric coupling between the jointly trained blocks. A model with newly retuned embeddings can also overfit during the next retuning epoch, consistent with the early checkpoint selections in Appendix A.3.

B.3.2 Additional split comparisons

Table 28 examines whether the block-wise diagnostic is sensitive to the data partition. All methods use the same partition within each setting. Avazu samples are ordered chronologically using the provided hour field. For Criteo, samples retain their released row order because timestamps and day boundaries are unavailable. We use this partition as an order-preserving proxy. We allocate 80% of the samples to training, 10% to validation, and 10% to testing. Preprocessing is fitted on the training partition and then applied to validation and test data. Training budgets and checkpoint-selection rules are shared across methods.

For DCN-v2 on Avazu, mean test logloss is 0.3713 for end-to-end training, 0.3701 for embedding retuning, and 0.3688 for Embedding Soups. The corresponding standard deviations are 0.000152, 0.000091, and 0.000130. These aggregate statistics complement the AUC results in Table 28. Across both datasets, embedding retuning retains its advantage over FI retuning under the evaluated partitions.

B.4 Comparative analysis of SWA

Stochastic weight averaging Izmailov et al. [2018] (SWA) is a technique used in training deep neural networks to improve the models' generalization. Different from other model merging techniques or

Table 28: Additional ordered-split comparisons: timestamp-based evaluation on Avazu and an order-preserving proxy on Criteo. Scores are test AUC; the last column is the FIRT train–validation AUC gap.

Dataset	FI	E2E	EmbRT	Soup	FIRT	FIRT gap
Avazu	DNN	0.7941	0.7962	0.7980	0.7743	0.0971
Avazu	DCN-v2	0.7944	0.7962	0.7980	0.7725	0.0984
Avazu	FinalMLP	0.7948	0.7968	0.7987	0.7751	0.1009
Criteo	DNN	0.8095	0.8105	0.8120	0.7981	0.0477
Criteo	DCN-v2	0.8103	0.8107	0.8123	0.7973	0.0511
Criteo	FinalMLP	0.8099	0.8110	0.8125	0.7973	0.0473

our proposed embedding soups, SWA averages the weights along the same training trajectory. We perform SWA on both feature interaction (MLP) and embedding parameters by maintaining an EMA model along the training trajectory:

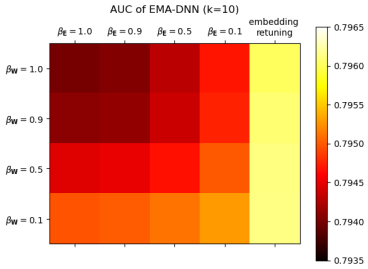
$$
\begin{aligned}
\mathbf{W}^{\mathrm{EMA},0} &= \mathbf{W}^{0,0} \\
\mathbf{E}^{\mathrm{EMA},0} &= \mathbf{E}^{0,0} \\
\mathbf{W}^{\mathrm{EMA},i} &= \beta_{\mathbf{W}}\mathbf{W}^{0,ik} + (1-\beta_{\mathbf{W}})\mathbf{W}^{\mathrm{EMA},i-1}, \text{for each } k \text{ iterations,} \\
\mathbf{E}^{\mathrm{EMA},i} &= \beta_{\mathbf{E}}\mathbf{E}^{0,ik} + (1-\beta_{\mathbf{E}})\mathbf{E}^{\mathrm{EMA},i-1}
\end{aligned}
$$

where $\mathbf{W}^{\mathrm{EMA},i}$ and $\mathbf{E}^{\mathrm{EMA},i}$ denote the parameters of the EMA feature interaction and embedding at the i-th step of EMA updates. $\mathbf{W}^{0,ik}$ and $\mathbf{E}^{0,ik}$ denote the parameters of the end-to-end trained feature interaction and embedding at the ik-th iterations of the end-to-end training.

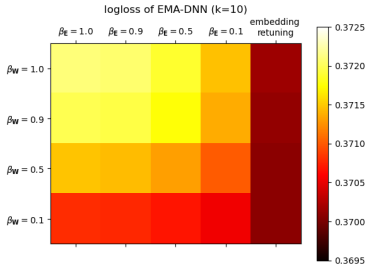
When β approaches one, the EMA parameters approach latest end-to-end trained ones. We run experiments with $\beta_{\mathbf{W}} \in \{0.1, 0.5, 0.9, 1.0\}$, $\beta_{\mathbf{E}} \in \{0.1, 0.5, 0.9, 1.0\}$, and $k \in \{10, 30, 100, 300\}$. All experiments share the same hyperparameter configuration including seed, which means all EMA models are built from the same training trajectory. Moreover, we run *embedding retuning* over these EMA feature interactions to comprehend the advantage of SWA. In order to understand the connection between our proposed embedding soups and SWA, we also apply EMA in retuning embeddings and perform model soups over EMA-retuned embeddings.

Figure 4 reports EMA-DNN performance on Avazu for different EMA update intervals and coefficients. Figure 5 compares soups built from directly retuned embeddings and EMA-retuned embeddings. From the figures, we have the following observations:

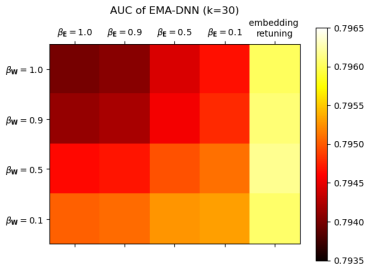
- **EMA feature interaction alone is not substantially stronger than the end-to-end feature interaction.** Applying EMA to both embeddings and FI parameters improves performance, but the FI-only difference is small (AUC 0.7960 $\rightarrow$ 0.7962 after embedding retuning). With end-to-end embeddings fixed ($\beta_{\mathbf{E}} = 1.0$), EMA FI still improves over the original end-to-end model (AUC 0.7942 $\rightarrow$ 0.7950), suggesting that trajectory smoothing can improve alignment with sparsely updated embeddings.
- **EMA embedding itself can improve over both end-to-end trained embeddings and retuned embeddings, but improvements from EMA are not orthogonal to those from soups.** Compared to the end-to-end trained embeddings ($\beta_{\mathbf{E}} = 1.0$), EMA embeddings can improve performance significantly (AUC 0.7942 $\rightarrow$ 0.7947 under one effective EMA setting). EMA-retuned embeddings can also improve single-run retuning (AUC 0.7962 $\rightarrow$ 0.7966), but the incremental benefit diminishes as more soup ingredients are added.
- **Better alignment between FI and embeddings can improve performance.** EMA on only one block can help, but stronger improvements appear when EMA is applied to both blocks. Effective settings often use similar $\beta_{\mathbf{W}}$ and $\beta_{\mathbf{E}}$, again suggesting that FI–embedding alignment matters.



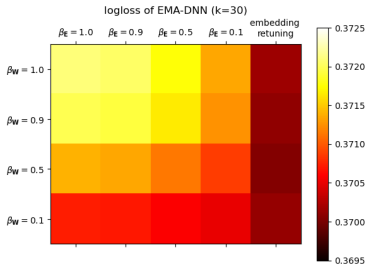
(a) AUC ($k = 10$)



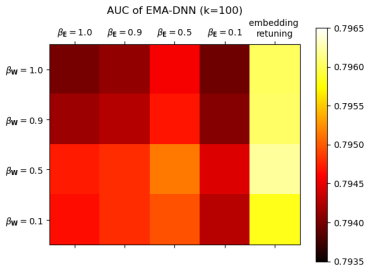
(b) logloss ($k = 10$)



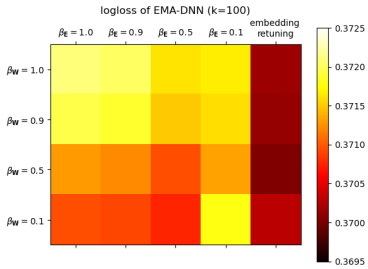
(c) AUC ($k = 30$)



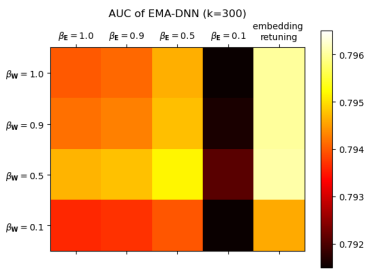
(d) logloss ($k = 30$)



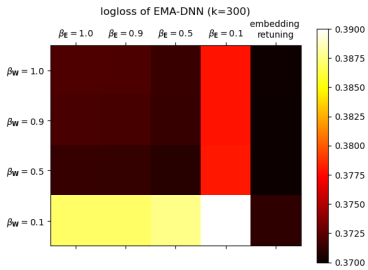
(e) AUC ($k = 100$)



(f) logloss ($k = 100$)



(g) AUC ($k = 300$)



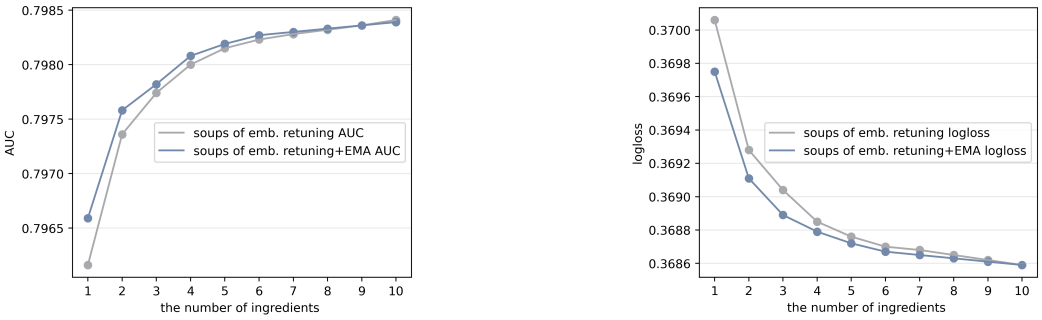
(h) logloss ($k = 300$)

Figure 4: EMA-DNN performance on Avazu. Each pair of subfigures reports AUC and logloss for one EMA update interval k.

C Connections to related methods

C.1 Model soups

Model soup Wortsman et al. [2022] and other model averaging techniques Frankle et al. [2020], Izmailov et al. [2018] motivate *embedding soups*. Our work is not only a simple adaptation of model soups:



(a) AUC of embedding soups

(b) logloss of embedding soups

Figure 5: Embedding soups built from directly retuned embeddings versus EMA-retuned embeddings on Avazu.

- **Motivation** Model soups are inspired by the observation that fine-tuned models optimized independently from the same initialization lie in the same basin of the error landscape Neyshabur et al. [2020]; ensembling independent runs trained with different hyperparameters improves performance Gontijo-Lopes et al. [2021]. While our embedding soups are inspired by the observation that, though retuning embeddings with a frozen feature interaction module, models with the retuned embedding still overfit at the second epoch. The phenomenon hints that models with the retuned embeddings are still prone to overfitting. Therefore, we seek to utilize an average-like method to mitigate the overfitting issue.
- **Finding** Though weight averaging is a long-standing technique in optimization and machine learning since the 1990s Polyak and Juditsky [1992], Ruppert [1988], it is widely believed that neural network training landscapes are highly non-convex with many disconnected solutions. Model soups surprisingly find that averaging independently fine-tuned models can achieve high performance. Our setting adds two CTR-specific findings: retuning embeddings improves deep CTR models, and retuned embeddings under a fixed FI module often lie in a connected low-loss region, as shown in Figure 2. We discuss failure cases in Appendix C.3.

C.1.1 Compatibility and alignment

For DNN on Avazu, the jointly trained embedding and FI blocks achieve test AUC 0.7943. Raw cross-seed swaps of embedding and FI blocks yield AUC values of 0.5044, 0.3895, and 0.5280. In contrast, reinitializing and retuning embeddings under a fixed FI module achieves 0.7960, and averaging independently retuned embedding tables achieves 0.7981. Successful retuning shows that the learned FI module can support newly estimated embeddings. The contrast with cross-seed swaps suggests a role for aligning the embeddings with the fixed FI module.

Figure 2 shows a connected region with high AUC and low logloss when interpolating between the original embedding table and two retuned tables under the same FI module. This connected region illustrates the compatible-basin assumption in Section 4.4. At larger embedding dimension, retuned embeddings can occupy different basins, as observed in the failure cases in Appendix C.3.

C.1.2 Frequency-binned embedding measurements

We measure run-to-run embedding variability across $N = 4$ independently retuned embedding tables while keeping the FI parameters $\mathbf{W}_0$ fixed. Let $\mathbf{E}_{\mathrm{rt}}^{\mathrm{avg}} = N^{-1} \sum_i \mathbf{E}_{\mathrm{rt}}^{(i)}$ denote their average, and let B_b contain the categorical feature values in training-frequency bin b. We define the cross-run mean squared deviation, V_b, and the drift from the jointly trained embeddings $\mathbf{E}_0$, $D_b(\mathbf{E})$, as follows:

$$V_b = \frac{1}{N|B_b|} \sum_{i=1}^{N} \sum_{j \in B_b} \|\mathbf{E}_{\mathrm{rt},j}^{(i)} - \mathbf{E}_{\mathrm{rt},j}^{\mathrm{avg}}\|_2^2, \qquad D_b(\mathbf{E}) = \frac{1}{|B_b|} \sum_{j \in B_b} \|\mathbf{E}_j - \mathbf{E}_{0,j}\|_2. \tag{3}$$

Table 29 reports V_b, the mean individual-retuning drift $\overline{D}_b = N^{-1}\sum_i D_b(\mathbf{E}_{\mathrm{rt}}^{(i)})$ for individual retuned tables, and the soup drift $D_b(\mathbf{E}^{\mathrm{soup}})$. The final column reports the test-AUC change from replacing each frequency group individually with its averaged embeddings.

Table 29: Embedding measurements grouped by training frequency on Avazu. The FI module is held fixed for four embedding-retuning runs. The last column reports the test-AUC change when only the indicated frequency group in one retuned table is replaced by the averaged embeddings.

FI	Frequency	V_b	$\overline{D}_b$	$D_b(\mathbf{E}^{\mathrm{soup}})$	Bin ΔAUC
DNN	≤ 2	0.002663	0.092354	0.079834	+0.000018
	3–10	0.004760	0.131996	0.115456	+0.000025
	11–100	0.009128	0.199230	0.177879	+0.000022
	101–1000	0.013021	0.234163	0.208514	+0.000020
	> 1000	0.006895	0.167510	0.147663	+0.000041
DCN-v2	≤ 2	0.002401	0.091506	0.080272	+0.000017
	3–10	0.004397	0.137923	0.123044	+0.000024
	11–100	0.008735	0.216630	0.197129	+0.000021
	101–1000	0.012580	0.252699	0.228913	+0.000018
	> 1000	0.006966	0.177456	0.158186	+0.000033
FinalMLP	≤ 2	0.002388	0.089152	0.077710	+0.000018
	3–10	0.004420	0.132459	0.117066	+0.000025
	11–100	0.008804	0.204433	0.184224	+0.000020
	101–1000	0.012282	0.241511	0.218229	+0.000017
	> 1000	0.006022	0.165608	0.148123	+0.000034

Across all three architectures, variability is highest for feature values that occur 101–1000 times in the training data. This pattern may reflect differences in update frequency: the rarest embeddings may remain close to initialization, whereas moderately frequent values receive enough updates for different runs to diverge without enough observations to stabilize their estimates. In every frequency bin, the averaged embeddings lie closer to the jointly trained embeddings than individual retuned embeddings do on average. This contraction follows from averaging. Replacing one frequency bin of a retuned table with the corresponding averaged embeddings produces small positive AUC changes across bins. These individual replacements suggest that improvements are distributed across frequency groups. Averaging the entire table also changes interactions between groups, so its gain can differ from the sum of the individual effects.

One possible mechanism is that sparse, entry-wise embedding updates interact with a jointly evolving FI mapping. An embedding entry receives updates only when its categorical feature value occurs, whereas the shared FI parameters receive gradients from almost every mini-batch. During joint training, the FI mapping may change between successive updates to an entry, creating a nonstationary interaction function and path dependence. Fixing $\mathbf{W}_0$ removes this source of nonstationarity and provides a common interaction function and coordinate reference, while sparsity and unequal update frequencies remain.

C.2 Adaptive regularization

AdamAR Li and Lyu [2026] is a recent adaptive regularization method for large-scale sparse feature embedding models. It addresses one-epoch overfitting from the perspective of frequency-aware embedding regularization. This makes it an important recent-work baseline for our setting, because it provides a frequency-aware single-trajectory baseline for our embedding retuning and aggregation protocol.

Table 30 compares AdamAR with embedding soups on Avazu using DNN. AdamAR slightly improves standard end-to-end training, while embedding soups provide a larger improvement. Combining AdamAR with soups gives a small additional gain, suggesting that adaptive embedding regularization and embedding soups are complementary.

Table 30: Comparison with AdamAR Li and Lyu [2026] on Avazu using DNN. Higher AUC and lower logloss are better.

Method	AUC ↑	logloss ↓
end-to-end	0.7942	0.3716
end-to-end+AdamAR	0.7943	0.3713
Embedding soups	0.7980	0.3689
AdamAR+embedding soups	**0.7981**	**0.3688**

C.2.1 Additional model and dataset comparisons

Table 31 extends this comparison to additional model and dataset combinations. For each method, the regularization coefficient is selected independently using validation performance. The candidate values are $\{10^{-6}, 10^{-5}, 10^{-4}\}$ for DCN-v2 on Avazu and DNN on Criteo, and $\{10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ for DNN on Taobao. Adding Embedding Soups improves the validation-selected AdamAR models in these settings.

Table 31: Additional AdamAR comparisons. AUC and logloss are test scores.

Setting	Method	Selected α	AUC	logloss
Avazu–DCN-v2	AdamAR	10^{-6}	0.7944	0.3712
	AdamAR+Soup	10^{-6}	0.7979	0.3689
Criteo–DNN	AdamAR	10^{-4}	0.8099	0.4422
	AdamAR+Soup	10^{-4}	0.8125	0.4394
Taobao–DNN	AdamAR	10^{-4}	0.6359	0.1945
	AdamAR+Soup	10^{-6}	0.6394	0.1936

C.3 MEDA and ME

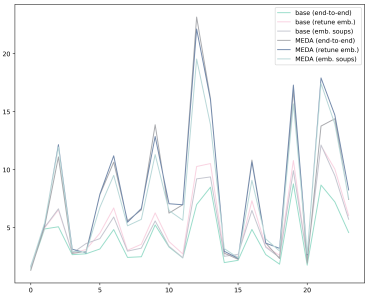
MEDA (**M**ulti-**E**poch learning with **D**ata **A**ugmentation Fan et al. [2024]) is a recently proposed technique for training deep CTR models with multiple epochs. MEDA reinitializes the embedding parameters at the beginning of each epoch to avoid overfitting and enable multi-epoch training. As shown in Table 32, MEDA improves deep CTR models on both Avazu and Criteo, and its gains are complementary to embedding soups. Unlike SWA in Appendix B.4, MEDA changes the training protocol: the FI module is trained for multiple epochs, while each embedding table is still trained for one epoch. Embedding soups can therefore further improve compatibility with the stronger MEDA-trained FI module. Embedding retuning alone gives relatively small improvements with a MEDA-trained FI module ($\Delta\text{AUC} < 0.0005$), so we omit those intermediate rows for readability.

ME (**M**ultiple **E**mbedding-set specific feature interaction Guo et al. [2023]) is another recent technique for training more powerful deep CTR models with scalable training costs. To mitigate embedding collapse in end-to-end training, ME replaces the original feature interaction module with grouped feature interactions. Each group uses a separate embedding set, encouraging embeddings to learn from diverse interaction patterns. We set the number of groups to $G = 4$ in our experiments.

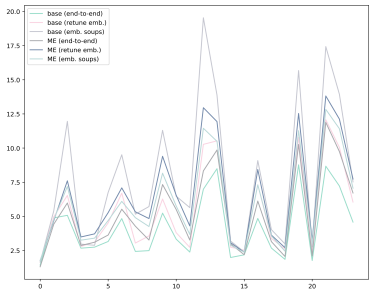
Table 32 reports how embedding soups combine with MEDA and ME across three deep CTR models. The results show that these techniques can be complementary, with the largest gains usually obtained when MEDA and embedding soups are combined.

Note that in some cases, results of embedding soups oscillate when scaling up the embedding dimension to $d = 64$ directly or using ME. Our experiments reveal two cases: First, multiple sets of retuned embeddings are not guaranteed to lie in the same basin (e.g., 1 out of 3 repeated runs of DNN on the Avazu dataset and all repeated runs of ME-FinalMLP on both the Avazu and Criteo datasets). Second, embedding retuning degrades the performance of the models (e.g., MEDA-DCN-v2 on the Criteo dataset).

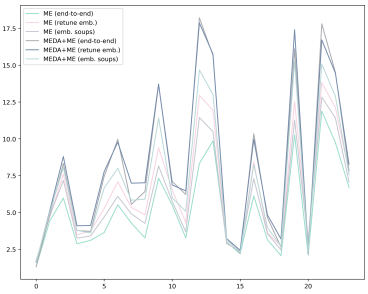
ME also provides a spectral metric for embeddings, called information abundance. We compute the singular values of each field's embedding matrix and define information abundance as the sum of singular values normalized by the maximum singular value, following Guo et al. [2023]. Figure 6



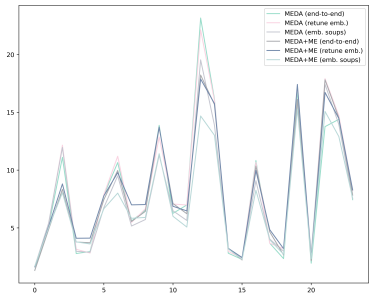
(a) base vs MEDA



(b) base vs ME



(c) ME vs MEDA-ME



(d) MEDA vs MEDA-ME

Figure 6: Information abundance of DCN-v2 embeddings on Avazu. The x-axis is the field index, and the y-axis is the information-abundance value defined from the normalized singular-value spectrum.

visualizes this metric for DCN-v2 on Avazu. MEDA substantially increases information abundance in Figures 6a and 6c. ME alone can also increase information abundance, but combining ME with MEDA does not always increase it further. Embedding retuning generally increases information abundance when the FI module is not trained with MEDA, while souped embeddings often have lower information abundance than single retuned embeddings, consistent with averaging acting as a denoising operation. Overall, information abundance is useful as a diagnostic but does not monotonically track predictive performance.

C.3.1 Taobao comparison

Table 33 evaluates MEDA with and without Embedding Soups across all five architectures on Taobao. Adding Embedding Soups improves both AUC and logloss in each setting.

Table 32: Compatibility with MEDA Fan et al. [2024] and ME Guo et al. [2023] on Avazu and Criteo. Higher AUC and lower logloss are better. The column d is the embedding dimension. Values in parentheses are standard deviations for settings with relatively high variance; other standard deviations are typically below 3×10^{-4}. Embedding soups use four ingredients by default.

Model	d	Method	Avazu		Criteo	
			AUC ↑	logloss ↓	AUC ↑	logloss ↓
DNN	16	end-to-end	0.7942	0.3716	0.8096	0.4424
		MEDA	0.7989	0.3683	0.8124	0.4395
		emb. soups	0.7980	0.3689	0.8120	0.4397
		MEDA+emb. soups	0.8015	0.3665	0.8142	0.4377
	64	end-to-end	0.7949	0.3711	0.8097	0.4422
		ME	0.7961	0.3701	0.8100	0.4419
		MEDA	0.8008	0.3671	0.8126	0.4393
		emb. soups	0.7983 (1.2e-3)	0.3687 (8.0e-4)	0.8125	0.4392
		ME+MEDA	0.8017	0.3666	0.8131	0.4388
		ME+emb. soups	0.7999	0.3677	0.8126	0.4393
		MEDA+emb. soups	0.8033	0.3655	0.8145	0.4375
		ME+MEDA+emb. soups	0.8034	0.3654	0.8150	0.4370
DCNv2	16	end-to-end	0.7944	0.3712	0.8103	0.4418
		MEDA	0.7989	0.3683	0.8129	0.4396
		emb. soups	0.7979	0.3689	0.8125	0.4393
		MEDA+emb. soups	0.8014	0.3666	0.8145	0.4375
	64	end-to-end	0.7957	0.3705	0.8107	0.4410
		ME	0.7968	0.3698	0.8111	0.4407
		MEDA	0.8023	0.3663	0.8132	0.4388
		emb. soups	0.7994	0.3680	0.8132	0.4387
		ME+MEDA	0.8018	0.3665	0.8136	0.4383
		ME+emb. soups	0.8004	0.3673	0.8134	0.4385
		MEDA+emb. soups	0.8046	0.3647	0.8145 (5.1e-4)	0.4375 (5.5e-4)
		ME+MEDA+emb. soups	0.8034	0.3654	0.8153	0.4367
FinalMLP	16	end-to-end	0.7949	0.3709	0.8097	0.4422
		MEDA	0.7992	0.3681	0.8121	0.4397
		emb. soups	0.7988	0.3683	0.8126	0.4393
		MEDA+emb. soups	0.8019	0.3664	0.8141	0.4379
	64	end-to-end	0.7957	0.3704	0.8098	0.4424
		ME	0.7967	0.3702	0.8101	0.4421
		MEDA	0.8006	0.3673	0.8123	0.4397
		emb. soups	0.7997	0.3678	0.8129	0.4389
		ME+MEDA	0.8011	0.3669	0.8128	0.4391
		ME+emb. soups	0.7978 (6.6e-3)	0.3691 (6.0e-3)	0.8098 (9.4e-4)	0.4433 (1.5e-3)
		MEDA+emb. soups	0.8033	0.3655	0.8144	0.4376
		ME+MEDA+emb. soups	0.7982 (2.3e-3)	0.3689 (1.7e-3)	0.8132 (4.7e-4)	0.4388 (4.5e-4)

Table 33: MEDA and MEDA+Soup on Taobao. AUC and logloss are test scores.

FI module	MEDA AUC	MEDA logloss	+Soup AUC	+Soup logloss	ΔAUC
DNN	0.6355	0.1949	0.6394	0.1936	+0.0039
DCN-v2	0.6358	0.1943	0.6392	0.1936	+0.0034
DeepFM	0.6357	0.1945	0.6391	0.1936	+0.0034
FinalMLP	0.6358	0.1944	0.6366	0.1943	+0.0008
IPNN	0.6359	0.1944	0.6395	0.1936	+0.0036

D Broader impacts

Improved CTR prediction may help recommendation and advertising systems present more relevant content. Embedding Soups preserves the single-model inference architecture, but requires additional offline training and associated computational resources. Better click prediction does not by itself ensure user welfare, fairness, or privacy: optimizing clicks can reinforce exposure biases or favor attention-seeking content, and the use of behavioral data can create privacy risks. These outcomes are not measured by the AUC and logloss experiments in this work. Deployment therefore requires application-specific assessment of data governance, exposure across user groups, and objectives beyond click prediction.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the main empirical findings, the proposed Embedding Soups method, and the claimed inference-cost benefit. The scope is asymmetric roles in generalization under block-wise retuning and their connection to embedding aggregation.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Sections 3–4.4 and Appendices B.3, C.1, B.2, and C.3 discuss optimization and coupling confounds, the local compatible-basin assumption, offline cost, and MEDA/ME failures.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper provides local variance-based intuition and derivations, but does not present formal theoretical results as standalone theorems requiring complete proof obligations.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper reports datasets, splits, model architectures, optimizers, learning rates, embedding sizes, and the retuning/averaging protocol needed to reproduce the main results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The paper describes the availability of code, data preprocessing scripts, and experimental implementation in the supplementary material, together with instructions for reproducing the experiments.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental section specifies data splits, preprocessing choices, baselines, optimizer, learning rates, embedding size, hidden-layer configuration, random seeds, and evaluation metrics.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The main performance comparisons use paired two-tailed t-tests. Appendix A.2 describes the repetitions for the additional diagnostics.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper reports the main compute resource used for the experiments, including that the experiments can be run on a single RTX 3090 GPU.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses public recommendation datasets and standard CTR modeling experiments, and the authors have considered the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: Appendix D discusses potential relevance benefits, extra offline computation, exposure biases, attention-seeking incentives, and privacy risks. These societal outcomes are not measured by our CTR experiments.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release high-risk models or datasets such as pretrained language models, image generators, or newly scraped sensitive datasets.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Existing datasets, baselines, and software packages used in the paper are cited in the main text, including prior CTR benchmarks and PyTorch.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The paper introduces code and scripts for the proposed method and experiments, and the supplementary material documents their use for reproducing the reported results.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing experiments or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or human-subject studies, so IRB or equivalent approval is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs are not used as an important, original, or non-standard component of the core research methodology.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.