CAPE: Generalized Convergence Prediction Across Architectures Without Full Training

Anonymous authors
Paper under double-blind review

Abstract

Training deep neural networks to convergence is expensive and time-consuming, especially when exploring new architectures or hardware setups. Prior work has focused on estimating per-iteration cost or total training time assuming a fixed step count, but has largely ignored the critical challenge of predicting how many steps a model will take to converge. We introduce CAPE (*Convergence-Aware Prediction Engine*), a lightweight, probing-based system that accurately predicts the number of training steps required for convergence without executing full training runs. CAPE probes models at initialization using a small batch of data to extract both structural and dynamical features, including parameter count, gradient norm, NTK trace, dataset size, and learning rate. Using these features, we build a meta-dataset spanning a wide range of model types and train a meta-model to forecast convergence steps. CAPE attains mean absolute errors of 3–9 optimization steps across MLP, CNN, RNN, and Transformer models, consistently surpassing strong baselines. This performance remains stable across a fourfold range in typical convergence horizons (15–60 steps), offering practical value for rapid model selection and budget planning. CAPE offers a practical and generalizable solution for convergence forecasting, supporting faster model selection, efficient scheduling, and resource-aware training.

1 Introduction

Deep neural networks (DNNs) require efficient training methods because their development depends on ample computational resources and extended training periods. The current techniques (Bergstra & Bengio, 2012; Li et al., 2018; Parker-Holder et al., 2020) require extensive experimental testing to find optimal hyperparameters and training schedules, which results in both time consuming delays and unnecessary resource consumption. The Neural Tangent Kernel (NTK) theory stands out as a significant contribution to understanding how deep neural networks converge in recent research (Arora et al., 2019; Jacot et al., 2018; Lee et al., 2019; Wang et al., 2022; Mu et al., 2020), which analyzes the training dynamics in high-dimensional settings. However, both theoretical and empirical works highlight critical limitations: NTK validity depends on tight rescaling conditions (Boix-Adserà & Littwin, 2023), extensions with regularization are still confined near initialization (Clerico & Guedj, 2024), and empirical scaling behaviours deviate significantly from NTK predictions (Vyas et al., 2023). These gaps limit applicability to real-world scenarios. Furthermore, studies on meta-learning (Ji et al., 2020; Ye et al., 2021; Chen et al., 2020; Guan et al., 2022; Harrison et al., 2022; Zhou et al., 2019; Tack et al., 2022) aim to improve learning efficiency by leveraging knowledge from past tasks, but cannot often predict training time for held-out configurations (i.e., new model–dataset–hyperparameter tuples). Work on predicting computational cost (Justus et al., 2018; Pourali et al., 2025; Geoffrey et al., 2021) has shown promise, but often relies on linear models that fail to capture the inherent non-linearities of DNN training. To address these limitations, we introduce CAPE (Convergence-Aware Prediction Engine), a novel approach that combines probing-based feature extraction with meta-learning to predict the number of training steps required for convergence accurately.

Our proposed system utilizes a probing-based feature extraction technique to capture the essential characteristics of a DNN at initialization, eliminating the need for full training. This approach draws inspiration from recent advances in understanding the relationship between training dynamics and generalization performance

(Wang et al., 2022; Ronen et al., 2019; Marion & Berthier, 2023). By extracting features such as the number of parameters, gradient norms, and a proxy for the NTK trace, we capture both structural and dynamic aspects of the model. These features are then used to construct a meta-dataset, encompassing a wide range of DNN architectures (MLPs, CNNs, RNNs, Transformers) and datasets. The meta-dataset is crucial for training a meta-model, a regression model capable of learning the complex mapping between the extracted features and the actual number of steps required for convergence. This meta-model is designed to generalize to held-out datasets, providing a robust and efficient prediction mechanism. The use of a meta-learning approach allows us to learn from the diverse training experiences captured in the meta-dataset, leading to improved generalization capabilities compared to traditional linear regression models (Justus et al., 2018; Zancato et al., 2020).

The key advantage of our approach lies in its ability to predict the number of steps needed during the training to reach convergence *before* initiating full training. This capability offers significant benefits for various aspects of the deep learning workflow. Researchers and practitioners can leverage this predictive power to make informed decisions regarding model selection, optimizing training schedules, and efficiently allocating computational resources. By accurately estimating the training cost upfront, our system contributes to a more efficient and effective deep learning pipeline, reducing the time and resources spent on experimentation and optimization. The proposed system addresses the limitations of existing methods by combining the power of probing-based feature extraction with the generalization capabilities of meta-learning, offering a practical and scalable solution for predicting DNN training convergence. This work contributes to the broader goal of making deep learning more accessible and efficient, enabling researchers and practitioners to focus on model development and deployment rather than lengthy training processes.

To summarize, our contributions are the following:

(i) We introduce a probing-based convergence prediction system that estimates the number of training steps required for deep neural networks (DNNs) to converge, without executing full training runs.

(ii) We extract both structural and dynamical features—including parameter count, gradient norms, NTK trace proxy, dataset size, and learning rate—by probing the model at initialization using a small batch of data.

(iii) We construct a meta-dataset that spans diverse architectures (MLPs, CNNs, RNNs, Transformers) and datasets, capturing convergence behavior across a wide range of model configurations.

(iv) We develop a regression model based on gradient-boosted trees, trained on the meta-dataset to learn the complex mapping between initialization-time features and the number of steps to convergence. This meta-model generalizes to held-out datasets not used during meta-training.

(v) We demonstrate that our system enables early estimation of training cost, allowing researchers and practitioners to make informed decisions about model selection, resource budgeting, and training schedules—improving overall efficiency in deep learning workflows.

2 Related Works

2.1 Convergence Prediction in Deep Learning

Recent works on convergence analysis (Allen-Zhu et al., 2019; Zou et al., 2020; Ji et al., 2020; Gao et al., 2024; Zancato et al., 2020) in deep learning focus heavily on theoretical guarantees derived from over-parameterized networks and NTK-based formulations. These methods show that gradient descent can converge globally when networks are sufficiently wide and initialized under specific distributions, assuming either data separability or proximity to initialization. Beyond NTK, other lines of work establish convergence guarantees under weaker conditions such as gradient domination (Weissmann et al., 2025), logistic loss on two-layer networks (Gopalani et al., 2024), small initialization regimes (Kumar & Haupt, 2025), and optimizer-specific analyses of RMSProp/Adam (Zhang et al., 2025). For Transformers, convergence analysis has also considered implicit bias and self-attention dynamics (Vasudeva et al., 2024). Some models reformulate training dynamics as stochastic differential equations in function space, enabling closed form predictions of time-to-accuracy using the eigenvalues of the NTK (Zancato et al., 2020; Wan et al., 2021; Lee et al., 2019). However, these approaches are mostly confined to fine-tuning or pre-trained regimes, often requiring the network to remain in a small perturbation region around initialization (Allen-Zhu et al., 2019; Zancato et al., 2020). While such works provide valuable theoretical insights, their reliance on restrictive assumptions limits computational practicality and precludes advance prediction of convergence before training begins—a gap our system directly addresses.

2.2 Learning Curve Extrapolation and Training Time Estimation

Learning curve extrapolation is widely used to estimate final performance from early training signals, especially within AutoML and neural architecture search workflows. Traditional approaches use Bayesian curve fitting, Gaussian processes, or ensemble-based estimators that require observing partial training trajectories (Domhan et al., 2015; Klein et al., 2017). More recent meta-learned models like LC-PFN (Adriaensen et al., 2023) reduce reliance on handcrafted priors and instead fit generalizable predictors over many tasks, enabling one-shot extrapolation of loss or accuracy. However, such methods still depend on partial curve data and are inapplicable when no training has occurred. Some frameworks like MOTE-NAS jointly predict resource use and model quality but focus more on cost or latency than convergence (Zhang et al., 2024b). Our work differs by offering zero-shot convergence prediction—estimating the number of steps required for training from scratch using initialization-only features, without relying on any part of the learning curve (Zela et al., 2020; Bahri et al., 2024). Complementary studies have shown how learning rate schedules, particularly cooldown phases, critically shape convergence dynamics (Dremov et al., 2025), further motivating the need for predictors that generalize beyond handcrafted training trajectories.

2.3 Meta-Learning and Probing-Based Estimation

Meta-learning has proven effective for transferring training knowledge across tasks and model types, especially when applied to optimizer adaptation, initialization heuristics, and performance modeling (Ye et al., 2021; Zhang et al., 2024a; Guan et al., 2022). Convergence in distributed and federated settings has also been studied, where Local SGD exhibits accelerated rates for over-parameterized models (Qin et al., 2022), complementing meta-generalization perspectives. Some approaches use task-conditioned priors or learned regularizers that adapt to optimization landscapes dynamically, often leading to improved generalization across datasets and objectives (Tack et al., 2022; Jiang et al., 2021; Grant et al., 2018). Probing-based estimation complements this by extracting features like gradient norm, NTK trace proxies, and parameter counts at initialization to guide predictions about training dynamics (Zhu et al., 2022; Xia et al., 2020). These methods often avoid full training by leveraging small-batch statistics and meta-trained regressors, reducing computational overhead (Adriaensen et al., 2023; Wang & Ma, 2022). Despite their strengths, prior work typically focuses on few-shot learning or inner-loop optimization efficiency rather than directly predicting full-model convergence from scratch, which our work uniquely addresses.

3 Predicting Convergence Without Training

In this section, we formalize the task of predicting the convergence steps of a deep neural network (DNN) without executing full training. Rather than relying on learning curve extrapolation or empirical tuning, CAPE approach estimates the number of gradient updates (i.e., steps) needed to reach a predefined convergence threshold, using only features extracted at initialization.

Assumption 1 (Fixed Initialization). *The model parameters $\theta \in \mathbb{R}^d$ are initialized using a fixed, reproducible scheme (e.g., Xavier or He initialization), and remain unchanged during probing.*

Let $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$ denote a neural network with parameters θ, and let the dataset be $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$. Define the empirical training loss after t steps as $\mathcal{L}_t(\theta)$.

Definition (Convergence Step Count). *We define the convergence time T^* as the minimum number of training steps required for the loss to drop below a fixed fraction $\epsilon \in (0,1)$ of the initial loss:*

$$T^* = \min \{t \in \mathbb{N} \mid \mathcal{L}_t(\theta) \leq \epsilon \cdot \mathcal{L}_0(\theta)\} \tag{1}$$

Assumption 2 (Smooth Loss Decay). *We assume the loss $\mathcal{L}_t(\theta)$ decays smoothly and monotonically under standard optimizers (e.g., SGD or Adam), allowing the convergence condition to be well-defined.*

To predict T^* without training, we extract a feature vector $\mathbf{z} \in \mathbb{R}^k$ from the model at initialization and a small subset of training data:

Assumption 3 (Representative Probing Subset). *A randomly sampled subset $\mathcal{D}_{probe} \subset \mathcal{D}$, where $|\mathcal{D}_{probe}| \ll N$, provides a sufficient approximation of the model's training dynamics.*

$$\mathbf{z} = \phi(f_\theta, \mathcal{D}_{\text{probe}}) \tag{2}$$

where f_θ denotes the neural network parameterized by θ and $\phi(\cdot)$ is the probing-based feature extractor defined in Section 4.

Probe-batch vs. full-dataset convergence Recall from Assumption 3 that we measure convergence on a small *probe* subset $D_{\text{probe}} \subset D$ of size $B \ll |D|$. Concretely, we define

$$T^* = \min \{t \in \mathbb{N} \,|\, L_t(\theta) \leq \varepsilon L_0(\theta)\}, \quad L_t(\theta) = \frac{1}{B} \sum_{(x,y)\in D_{\text{probe}}} \ell(f_\theta^t(x), y) \tag{3}$$

so that *both* our feature probes (e.g. $\log \|\nabla \ell\|^2$, NTK-trace) *and* the threshold check $L_t \leq \varepsilon L_0$ use only this one batch. By contrast, a "full-dataset" convergence measure would require

$$L_t^{\text{full}}(\theta) = \frac{1}{|D|} \sum_{(x,y)\in D} \ell(f_\theta^t(x), y), \quad T_{\text{full}} = \min \left\{t \,\middle|\, L_t^{\text{full}}(\theta) \leq \varepsilon L_0^{\text{full}}(\theta)\right\} \tag{4}$$

which entails an entire-epoch pass over D per step. Our *probe-batch* approach is orders of magnitude cheaper—yet, by aggregating M independent probe trials in the meta-dataset, the regressor learns to map these "mini-batch" dynamics onto the true full-training behavior. We then aim to learn a regression function $g : \mathbb{R}^k \to \mathbb{R}$ that predicts convergence steps based on the probing features:

$$\hat{T} = g(\mathbf{z}) \tag{5}$$

A meta-dataset $\mathcal{M} = \{(\mathbf{z}^{(j)}, T^{*(j)})\}_{j=1}^M$ is constructed from different models and datasets, each labeled with the actual convergence step count $T^{*(j)}$.

$$\min_{g\in\mathcal{G}} \frac{1}{M} \sum_{j=1}^{M} \left(g(\mathbf{z}^{(j)}) - T^{*(j)}\right)^2 \tag{6}$$

Task Formalization: We consider a broad class of supervised learning tasks—ranging from scalar regression to multi-class classification to sequence modeling—each instantiated by a function

$$f_\theta : \mathcal{X} \to \mathcal{Y}$$

and corresponding loss

$$\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$$

Concretely:

- **Scalar regression**: $\mathcal{Y} = \mathbb{R}$, use an MLP with a single-unit head and $\ell(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$ (MSE).
- **Multi-class classification**: $\mathcal{Y} = \{1, \ldots, C\}$, use a C-way linear head and $\ell(\hat{\mathbf{z}}, y) = -\log\left[\text{softmax}(\hat{\mathbf{z}})_y\right]$ (cross-entropy).
- **Sequence modeling & others**: e.g. Transformer encoder–decoder with token-wise cross-entropy, or multi-label heads with either binary-CE or MSE on one-hot vectors.

In all cases, we define convergence based on the same probe batch D_{probe}, using the definition of T^* previously established in Eq. equation 4, with the loss ℓ adjusted to the task type.

Assumption 4 (Meta-Generalization). *The meta-regression model g trained on $\mathcal{M}$ generalizes to held-out datasets drawn from a similar distribution.*

To improve training stability and scale sensitivity, we minimize the squared error in log space:

$$\min_{g \in \mathcal{G}} \frac{1}{M} \sum_{j=1}^{M} \left(\log g(\mathbf{z}^{(j)}) - \log T^{*(j)} \right)^2 \tag{7}$$

Proposition 1 (Asymptotic Consistency). *Assuming $\mathcal{M}$ is sufficiently diverse and $\mathcal{G}$ is a rich function class (e.g., universal approximators), the predictor g converges in probability to the ground-truth mapping $\phi(f_\theta, \mathcal{D}_{probe}) \mapsto T^*$ as $M \to \infty$.*

This formulation enables convergence step prediction without full training, relying solely on computational signals extracted from initialization.

4 Probing-Based Feature Extraction and Meta-Learning

CAPE's implementation consists of three key components: probing feature design, meta-dataset construction, and meta-regressor training. We present the full implementation of our convergence prediction system, beginning with the design of initialization-time probing features that characterize both model structure and dynamics. We then describe how these features are used to construct a comprehensive meta-dataset across architectures and training conditions. Finally, we outline the meta-regressor architecture and training process, and assess its ability to generalize to held-out datasets.

4.1 Probing Feature Design

We extract a set of informative features from a randomly initialized model using a small probing batch from the training dataset. As noted in Assumption 1, initialization is fixed and reproducible, and features are computed without performing parameter updates, making the process efficient and architecture-agnostic.

Let f_θ be a DNN with parameters $\theta \in \mathbb{R}^d$, and let $\mathcal{D}_{\text{probe}} \subset \mathcal{D}$ be a randomly sampled subset with size $B \ll |\mathcal{D}|$, as assumed in Assumption 3. The feature extraction function is defined as:

$$\mathbf{z} = \phi(f_\theta, \mathcal{D}_{\text{probe}}, \eta, B, N) \tag{8}$$

where η is the learning rate, B is the batch size, and N is the total dataset size. Each of these values contributes directly to the expected training dynamics. To capture key aspects of initialization-time behavior, we compute the following features:

Parameter Count P. The total number of trainable parameters is used as a proxy for model capacity:

$$P = \|\theta\|_0 \tag{9}$$

This feature reflects architectural complexity—deeper or wider models typically have more parameters and may take longer to converge due to greater representational capacity.

Average Gradient Norm g^2. This measures the sensitivity of the loss to parameter updates at initialization:

$$g^2 = \frac{1}{B} \sum_{(x,y)\in\mathcal{D}_{\text{probe}}} \left\|\nabla_\theta \ell(f_\theta(x), y)\right\|^2 \tag{10}$$

A large gradient norm indicates a steeper loss surface and possibly faster initial descent, whereas small gradients suggest slower convergence or vanishing gradients. These dynamics echo observations of early directional convergence under small initializations in homogeneous networks (Kumar & Haupt, 2025), underscoring the importance of initialization in determining convergence speed.

NTK Trace Proxy τ. This curvature-sensitive measure approximates the trace of the neural tangent kernel:

$$\tau = \frac{1}{B} \sum_{x\in\mathcal{D}_{\text{probe}}} \left\|\nabla_\theta f_\theta(x)\right\|^2 \tag{11}$$

It captures the sensitivity of outputs to parameter perturbation, which relates to implicit regularization and convergence rates.

Log-Transformed Feature Vector. To improve numerical stability and reduce variance across configurations, we log-transform all features:

$$\mathbf{z} = \left[\log(P), \log(g^2), \log(\tau), \log(N), \log(\eta), \log(B)\right] \tag{12}$$

These features can be extracted in under one second on modern hardware, making them suitable for real-time or batch evaluation. For each dataset and architecture, we automatically instantiate the correct output head (scalar, C-way, or sequence) and select the matching loss function

$$\ell(\hat{y}, y) = \begin{cases} \frac{1}{2}(\hat{y} - y)^2 & \text{(regression)} \\ -\log\left[\text{softmax}(\hat{\mathbf{z}})_y\right] & \text{(classification)} \end{cases}$$

This same ℓ is used both for *probe-batch feature extraction* and for measuring the convergence step T^*.

4.2 Meta-Dataset Construction

To train the predictor described in Section 3, we construct a meta-dataset:

$$\mathcal{M} = \left\{(\mathbf{z}^{(j)}, T^{*(j)})\right\}_{j=1}^{M} \tag{13}$$

where each $(\mathbf{z}^{(j)}, T^{*(j)})$ pair represents the extracted features and the actual number of steps to reach convergence threshold ϵ. To support broad generalization, the meta-dataset is constructed from a diverse range of architectures, datasets, and training conditions as discussed in Section 5.

4.3 Meta-Regressor Architecture and Training

Given the meta-dataset, we learn a function $g : \mathbb{R}^k \to \mathbb{R}$ such that:

$$\hat{T} = g(\mathbf{z}) \tag{14}$$

where $\hat{T}$ approximates the true convergence steps T^*. We use XGBoost (Chen & Guestrin, 2016), a gradient-boosted decision tree model, as our meta-regressor. Owing to its superior predictive performance compared to alternative models, we adopt XGBoost for the main analysis. A detailed description of all evaluated regression models and their configurations is provided in Appendix A.4. It is trained on log-transformed features and convergence steps, with early stopping and regularization to prevent overfitting. The XGBoost model is trained using log-transformed targets to minimize scale sensitivity. Regularization includes dropout (for MLPs), depth pruning (for trees), and early stopping. Model selection is performed via 3-fold cross-validation on the training meta-set.

5 Experiments

We evaluate our framework across four model families of MLPs, CNNs, RNNs, and Transformers, each instantiated with randomized yet controlled architectural configurations to capture a broad spectrum of convergence behaviors.

Model families and training protocol. MLPs are constructed using 2–4 fully connected layers with hidden sizes sampled from $\{128, 256, 512, 1024\}$, combined with BatchNorm, dropout, and either GELU or ReLU activations. These configurations mirror compact feedforward backbones often used in classification benchmarks and tabular modeling tasks.

CNNs are instantiated from three reduced-scale families derived from widely used architectures: variants of *VGG* (Simonyan & Zisserman, 2014) (sequential 3×3 convolutional stacks with optional pooling), *ResNet* (He et al., 2016) (post-activation residual blocks with projection downsampling), and *MobileNetV2* (Sandler et al., 2018) (inverted residual blocks with width multipliers in $\{0.5, 0.75\}$). These compact versions preserve the design principles of their full-scale counterparts while remaining computationally lightweight.

RNNs employ either GRU (Cho et al., 2014) or LSTM (Hochreiter & Schmidhuber, 1997) cells with 1–3 stacked layers, hidden sizes from $\{128, 256, 512\}$, and optional bidirectionality. Inputs are processed as sequences of image rows, reflecting how recurrent architectures are often applied to sequential or time-series data. Transformers are instantiated from compact vision architectures introduced in recent research: *Vision Transformer (ViT)* (Dosovitskiy et al., 2020), *Compact Convolutional Transformer (CCT)* (Hassani et al., 2021), and *Pooling-based Vision Transformer (PiT)* (Heo et al., 2021). These models incorporate patch embeddings, convolutional tokenizers, or hierarchical pooling to adapt Transformer designs to small-scale vision tasks.

For training, we use consistent optimization protocols across families. All model classes adopt learning rates $LR \in \{5\times10^{-4}, 10^{-3}, 2\times10^{-3}\}$ and batch sizes $B \in \{32, 64, 128\}$, applied uniformly across architectures. Convergence thresholds are defined relative to the initial batch loss L_0: we use $\epsilon \in \{0.1, 0.15, 0.2\}$ for all model families, reflecting fast-to-moderate convergence under our adaptive training protocol. Training is capped at 5000 steps with plateau-based early stopping; if convergence is not reached, we record $T^* = 5000$.

To construct the meta-dataset used for training the convergence predictor, we utilized a workstation equipped with an Nvidia RTX 4000 Ada Generation GPU, an Intel Core i9-14900K processor, and 64 GB of RAM. The total time required to generate the full meta-dataset—spanning MLPs, CNNs, RNNs, and Transformers was approximately 40 hours.

Meta-Feature Extraction. To estimate convergence steps T^* from initialization, we extract six computational features from a probing batch—parameter count ($\log P$), batch size ($\log B$), squared gradient norm ($\log G^2$), NTK trace proxy ($\log \tau$), learning rate ($\log \mathrm{LR}$), and dataset size ($\log N$)—and record them for each configuration in a meta-dataset. We then train a gradient-boosted decision tree (XGBoost; 200 trees, maximum depth 4, learning rate 0.05, row subsampling 0.9, column subsampling by tree 0.9, $\ell_1 = 0$, $\ell_2 = 1$, seed 42) to regress $\log T^*$ on this meta-dataset, exponentiating predictions to obtain T^*. At evaluation time, we compute the same features on a small probing batch for the target setting and apply the trained regressor. As discussed in Appendix A.4, we choose XGBoost for its strong performance on tabular data, fast training speed, and low sensitivity to feature scaling—properties that align well with the needs of our convergence prediction task.

Datasets. All model families are evaluated on the meta-training corpus—MNIST (Deng, 2012), FashionMNIST (Xiao et al., 2017), and CIFAR-10/100 (Krizhevsky et al., 2009)—and we assess cross-dataset generalization on held-out datasets SVHN (Netzer et al., 2011) and IMDB (Maas et al., 2011), which are excluded from meta-training and reserved for evaluation. Dataset-specific normalization follows established practice; grayscale datasets (MNIST, FashionMNIST) are resized to 32×32 and replicated to three channels for Convolutional/Transformer models. SVHN is used at native 32×32 RGB. For IMDB (text), we form a fixed 3,072-dimensional hashed representation and reshape it to 3×32×32 for Convolutional/Transformer models (vector form is used directly for MLP/RNN). All evaluations share the same hyperparameter grids;

in-corpus results use a stratified cross-fold protocol, and held-out results apply the same protocol with evaluation sets disjoint from the meta-training corpus.

5.1 Evaluation Metrics

Our evaluation methodology is inspired by the framework introduced in (Zancato et al., 2020). For each configuration, we compute the predicted number of steps $T_{\text{predicted}}$ and compare it to the actual number of steps T_{real} required to reach convergence. We use the absolute prediction error $|T_{\text{predicted}} - T_{\text{actual}}|$ as our primary evaluation metric. Each configuration is repeated for 100 randomized trials to account for variance due to initialization and data sampling. To assess the predictive performance of the meta-regressors, we employ the following evaluation metrics:

- **Mean Absolute Error (MAE).** The MAE quantifies the average absolute difference between the predicted convergence time (T_{pred}) and the actual convergence time (T_{act}) where lower MAE values indicate better predictive accuracy:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|T_{\text{pred},i} - T_{\text{act},i}|.$$

- **Root Mean Squared Error (RMSE).** Computed in *steps* to emphasize larger deviations:

$$\text{RMSE} \;=\; \sqrt{\frac{1}{n}\sum_{i=1}^{n}\big(T_{\text{pred},i} - T_{\text{act},i}\big)^2}.$$

- **Median Absolute Error (MedianAE).** A robust absolute deviation in *steps*:

$$\text{MedianAE} \;=\; \text{median}_{1\le i\le n}\,\big|T_{\text{pred},i} - T_{\text{act},i}\big|.$$

- **Coefficient of Determination (R^2).** Explained variance of predictions relative to the variance of ground-truth steps:

$$R^2 \;=\; 1 - \frac{\sum_{i=1}^{n}\big(T_{\text{act},i} - T_{\text{pred},i}\big)^2}{\sum_{i=1}^{n}\big(T_{\text{act},i} - \overline{T}_{\text{act}}\big)^2}, \qquad \overline{T}_{\text{act}} = \frac{1}{n}\sum_{i=1}^{n} T_{\text{act},i}.$$

- **Bounded Relative Accuracy (AvgAccuracy).** Per-configuration accuracy in *steps*, defined as one minus the relative absolute error (truncated to $[0,1]$) and averaged:

$$\text{AvgAccuracy} \;=\; \frac{1}{n}\sum_{i=1}^{n}\min\left\{1,\ \max\left\{0,\ 1 - \frac{|T_{\text{pred},i} - T_{\text{act},i}|}{\max(T_{\text{act},i},\,1)}\right\}\right\}.$$

5.2 Evaluation Setup and Baselines for Convergence Prediction

Goal. We evaluate CAPE's probe-based meta-regressor for predicting the number of optimization steps required to reach a target loss ratio ϵ (denoted T^*). Ground truth T^* is computed with AdamW on the *same probe batch* used for feature extraction, with a soft cap of 5,000 steps and plateau early-exit (patience = 200, min-delta = 10^{-4}). We sweep learning rate LR $\in \{5{\times}10^{-4},\ 10^{-3},\ 2{\times}10^{-3}\}$, batch size $B \in \{32,\ 64,\ 128\}$, and convergence targets $\epsilon \in \{0.10,\ 0.15,\ 0.20\}$ across the datasets. We report MAE, RMSE, median absolute error, and R^2 on step counts; for presentation, we additionally apply a simple *epoch-space linear calibration* fitted on a 10% hold-out and used uniformly for all predictors.

Baselines. To our knowledge, there is no prior work that *zero-shot* predicts the number of optimization steps T^* required to reach a target loss ratio ϵ for a new (architecture, dataset, hyperparameter) instance

Architecture	Method	MAE	RMSE	MedianAE	R2	AvgAccuracy
MLP	NTKL	5.18	6.96	4.23	0.07	0.52
	SL	4.80	6.03	4.13	0.30	0.53
	LCE	4.73	6.64	3.57	0.15	0.58
	CAPE	**3.79**	**4.72**	**3.32**	**0.57**	**0.64**
RNN	NTKL	15.51	20.68	12.94	0.02	0.52
	SL	7.03	10.01	5.23	0.77	0.77
	LCE	22.67	28.72	17.93	-0.9	0.44
	CAPE	**4.21**	**6.16**	**2.88**	**0.91**	**0.88**
CNN	NTKL	9.13	12.40	7.25	0.23	0.57
	SL	4.46	6.13	3.29	0.81	0.78
	LCE	3.14	4.23	2.40	0.91	0.81
	CAPE	**2.86**	**4.06**	**2.00**	**0.92**	**0.87**
Transformer	NTKL	16.38	20.77	14.69	0.03	0.48
	SL	16.33	20.69	14.66	0.04	0.48
	LCE	16.85	21.90	13.88	-0.07	0.5
	CAPE	**9.03**	**12.89**	**6.19**	**0.63**	**0.68**

Table 1: Convergence–step prediction (T^*) across architecture families

using only single-batch probes together with a learned regressor. The closest effort is (Zancato et al., 2020), which also predicts steps *without training* but relies on a linearization of the network (NTK-style dynamics) and analytic approximations, rather than learned cross-architecture mappings from cheap probe features. Learning-curve extrapolation methods (Domhan et al., 2015) require a short warm-up and are therefore *not* zero-shot. Runtime/iteration-time predictors such as (Geoffrey et al., 2021) and (Pourali et al., 2025) target throughput or per-iteration time, not steps-to-ϵ.

- **NTK linearization (NTKL).** On a fresh mini-batch we form the gradient Gram matrix; its spectrum serves as a rate proxy, which we map to steps via $T \approx \log(1/\epsilon)/\hat{k}$, mirroring the zero-cost linearization spirit of (Zancato et al., 2020) while matching our probe-batch protocol.
- **Static linear (SL).** A linear regressor on static features $[\log P, \log B, \log \mathrm{LR}, \log N]$—in the spirit of iteration/runtime predictors (Geoffrey et al., 2021; Pourali et al., 2025)—is repurposed to predict $\log T^*$; no dynamic probes are used.
- **Learning-curve extrapolation (LCE).** We run a short warm-up (e.g., 60 steps) on the probe batch, fit a simple model in log-loss space, and extrapolate the step at which the loss reaches ϵL_0, following (Domhan et al., 2015).

We compare CAPE with low/zero-shot baselines—NTKL (NTK linearized), SL (Static-Linear), and LCE (short-run loss-curve extrapolation)—on MLP, RNN, CNN, and Transformer models. Columns report MAE, RMSE, MedianAE, R^2 and AvgAccuracy. The bolded cells indicates the best value within each architecture block. Results aggregate over datasets, learning rates $\{5 \times 10^{-4}, 10^{-3}, 2 \times 10^{-3}\}$, batch sizes $\{32, 64, 128\}$, thresholds $\varepsilon \in \{0.10, 0.15, 0.20\}$, and 100 trials per setting.

Observations. As discussed in Table 1, CAPE is the most accurate and stable predictor across all architecture families. For MLPs, CAPE attains MAE 3.79, RMSE 4.72, and MedianAE 3.32, with $R^2 = 0.57$ and average accuracy of 64%. The margin widens for RNNs—MAE 4.21, RMSE 6.16, MedianAE 2.88,

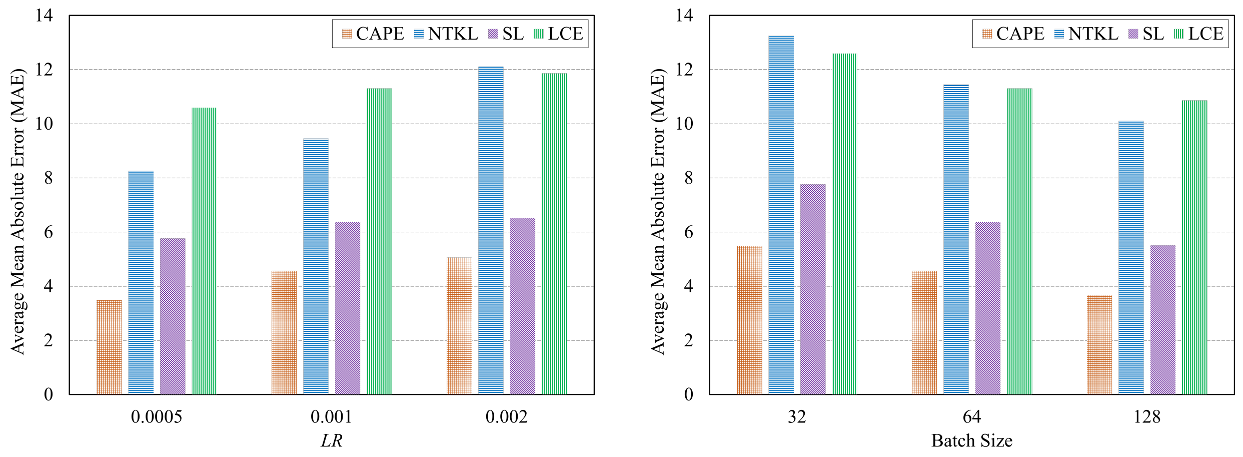

Figure 1: Overall average MAE (steps) across architectures and hyperparameters; CAPE yields the lowest error among all compared baselines. MAE increases with learning rate and decreases with batch size for all methods.

$R^2 = 0.91$ and **average accuracy of 88%**. CNNs show the strongest overall agreement: MAE 2.86, RMSE 4.06, and MedianAE 2.00, paired with the top $R^2 = 0.92$ and average accuracy of 87%. Transformers follow the same trend (MAE 9.03, RMSE 12.89, MedianAE 6.19, $R^2 = 0.63$, average accuracy of 68%), clearly surpassing all baselines whose errors cluster in the mid–teens to low–twenties with weak or near-zero explained variance. As observed in Figure 1, CAPE achieves the lowest *average MAE (in steps)* across all architectures when compared with NTKL, SL, and LCE under the matched evaluation protocol. The results are aggregated over datasets and full sweeps of learning rates $\{5 \times 10^{-4}, 10^{-3}, 2 \times 10^{-3}\}$ and batch sizes $\{32, 64, 128\}$, indicating robustness across hyperparameters rather than a narrow-setting effect. The comparison spans zero-/low-shot baselines (NTKL, SL, LCE) and CAPE within a unified protocol targeting the same ground-truth T^*, reinforcing that the improvement reflects a genuine predictive gain rather than any evaluation setup artifact. Across all methods we observe a consistent directional trend: as the learning rate increases, prediction error rises, while larger batch sizes reduce error (Zancato et al., 2020). This behavior is expected—higher learning rates move trajectories away from the local geometry captured by small-step analyses (degrading CAPE/NTKL/LCE), whereas larger batches reduce gradient noise and stabilize both probes and short-horizon fits.

Why certain architectures have higher MAEs? Mean Absolute Error (MAE) is measured in absolute steps, so families with larger typical ground-truth T^* naturally show larger absolute errors for the same relative miss. In our runs, the typical T^* scale increases with architectural complexity—approximately 15 for MLPs, 32 for CNNs, 50 for RNNs, and 60 for Transformers, making higher MAEs for Transformers expected even when fit quality is solid. Consistent with this scale effect and with greater early-step non-stationarity in attention normalization dynamics, Transformers exhibit higher MAE (9.03) than MLPs (3.79) and CNNs (2.86), while maintaining competitive R^2 and AvgAccuracy; conversely, CNNs and MLPs, which operate at lower T^* and exhibit more stationary early dynamics, achieve smaller MAEs and strong agreement. All families are evaluated under the same unified protocol and baselines such as LCE are afforded a 60-step warm-up whereas CAPE uses a single probe step, so the observed differences track intrinsic architectural dynamics rather than evaluation artifacts.

Baselines and cost. The NTK linearization proxy (NTKL) is consistently weak across families. The static linear model (SL) is occasionally competitive but generally trails CAPE. Learning-curve extrapolation (LCE) is unstable—most visibly on RNNs and Transformers, where R^2 is near zero or negative—despite being afforded a favorable budget: we execute *60 optimization steps* per configuration to enable extrapolation. In contrast, CAPE relies on a single one–mini-batch probe (roughly one step of work). Thus, even when LCE is allowed 60 warm-up steps, CAPE delivers lower error and higher explained variance at markedly lower computational cost—i.e., better results for less budget—underscoring its practicality for rapid, resource-constrained convergence-step prediction. To ensure reproducibility, we have provided the code and data that we used for our experimental evaluation in an anonymous GitHub repository[1].

5.3 Feature Ablation: Contribution of CAPE Components

As indicated in Table 2, using the *complete feature set* yields the lowest MAE on every dataset; removing any individual feature degrades accuracy. The largest degradations arise when excluding $\log LR$ and $\log N$ (typically +3–5 MAE points), underscoring the centrality of optimization scale and data regime. Dropping $\log P$ produces the next-largest increase ($\sim$+2–3.5), while $\log B$ has a smaller but consistent effect ($\sim$+1–2.5). The probe-derived terms, $\log G^2$ and $\log \tau$, contribute incremental yet reliable improvements (omission costs $\sim$+0.1–0.9), suggesting they act as calibrators that refine predictions. Overall, the components are complementary, with $\log LR$ and $\log N$ emerging as the dominant factors.

Model	CIFAR10	CIFAR100	FashionMNIST	MNIST
CAPE	**2.24**	**2.65**	**3.99**	**4.14**
$\hookrightarrow$ w/o $\log B$	4.44	5.13	5.7	5.17
$\hookrightarrow$ w/o $\log G^2$	2.43	2.73	4.16	4.26
$\hookrightarrow$ w/o $\log LR$	5.13	7.62	7.64	8.15
$\hookrightarrow$ w/o $\log N$	5.79	7.38	7.81	8.75
$\hookrightarrow$ w/o $\log P$	4.72	5.86	6.75	7.53
$\hookrightarrow$ w/o $\log \tau$	2.36	3.51	4.12	4.48

Table 2: Feature ablation of CAPE across MLP, CNN, RNN, and Transformer. Entries are per-dataset mean MAE (steps), averaged over learning rates, batch sizes, and ϵ targets.

5.4 Cross-Dataset Generalization on Held-Out Datasets

Dataset	**Method**	**Average MAE**	**Average Accuracy**
IMDB	NTKL	35.59	0.46
	SL	30.76	0.53
	LCE	40.83	0.48
	CAPE	**25.63**	**0.68**
SVHN	NTKL	36.76	0.49
	SL	52.93	0.35
	LCE	77.63	0.33
	CAPE	**31.64**	**0.51**

Table 3: Convergence–step prediction (T^*) on held-out datasets (cross-dataset generalization).

We evaluate cross-dataset generalization by applying our predictors to held-out datasets of Street View House Numbers (SVHN) (Netzer et al., 2011) and IMDB (Maas et al., 2011) that were *not* used to construct the meta-dataset. We compare CAPE against three non-learning baselines—NTKL, SL, LCE and report Average MAE and Average Accuracy aggregated over learning-rate and batch-size grids (Table 3).

CAPE delivers the best accuracy–error trade-off on both datasets. On IMDB, CAPE reduces error to 25.63 MAE, a 28% − 37% reduction relative to NTKL and LCE and 17% relative to SL, while increasing Average Accuracy to 68% compared to the best baseline (SL) with 53% average accuracy. On SVHN, CAPE attains 31.64 MAE, 14%, 40%, and 59% lower than NTKL, SL, and LCE, respectively—with an accuracy of 51 % compared to the best baseline (NTKL) of 49%. These improvements persist despite pronounced distribution

[1]`https://github.com/genericgitrepos/CAPE`

shift—including a cross-modality case for IMDB—suggesting that CAPE's short, batch-local probes capture optimization geometry that transfers beyond the datasets used to fit the meta-regressor.

Compared to the in distribution corpora used to train the meta-regressor (MNIST, Fashion-MNIST, CIFAR-10/100), SVHN and IMDB introduce substantial distribution shift. SVHN is drawn from street-view imagery and IMDB constitutes a cross-modality case, so the ground-truth convergence counts T^* tend to occupy a larger (160 steps) and more variable scale. Because MAE is measured in steps, this scale inflation naturally yields larger absolute errors even when relative fit remains competitive. Despite this harder setting, CAPE still delivers the best accuracy error trade-off on both datasets, outperforming NTKL, SL, and LCE under the same evaluation protocol. This suggests that CAPE's short, batch-local probes capture optimization geometry that transfers beyond the datasets used to fit the meta-regressor. A practical limitation is coverage: expanding the meta-dataset to include more heterogeneous domains should further reduce error under larger shifts. Our in-distribution study in the previous section—where the meta-dataset spans multiple datasets—already indicates strong performance when CAPE is trained once on a diverse corpus. This points to a practical path forward: expand meta-dataset diversity to strengthen out-of-distribution robustness while retaining CAPE's low-cost, short-probe workflow.

6 Conclusion and Limitation

In this paper, we presented CAPE, a lightweight predictive framework that estimates the number of training steps required for deep neural networks to reach a target accuracy *without* full training. Under stratified cross-fold evaluation, CAPE maintains single-digit absolute error (3–9 steps) across architectures and datasets using only a single mini-batch probe at initialization. The stability of the error across true horizons from 15 to 60 steps demonstrates a reliable, architecture-agnostic predictor for advance scheduling and cost-aware planning. Across all settings, CAPE delivers consistent improvements over competitive baselines, reinforcing its utility for selecting models and allocating compute without warm-up training. The approach generalizes across MLP, CNN, RNN, and Transformer families and maintains its advantage on held-out datasets, indicating that the probe captures transferable optimization signals. By integrating structural and dynamical cues—such as gradient-norm statistics, NTK-trace proxies, and dataset-level parameters—CAPE enables practical, *ex-ante* convergence estimation for model selection, resource budgeting, and scheduling in real-world workflows.

Hardware and Runtime Generalization. While CAPE is designed to be hardware-agnostic by focusing on step count rather than wall-clock time, practical deployment still requires mapping predicted steps to compute cost. Exploring runtime estimation or integrating hardware-aware features could help bridge this gap for end-to-end cost prediction.

Coverage of the meta-dataset. CAPE's accuracy depends on the diversity of the meta-training corpus. While it generalizes to held-out datasets, broader coverage—e.g., additional modalities, long-sequence tasks, and more aggressive augmentations—should further reduce error under larger shifts.

References

Steven Adriaensen, Herilalaina Rakotoarison, Samuel Müller, and Frank Hutter. Efficient bayesian learning curve extrapolation using prior-data fitted networks. *Advances in Neural Information Processing Systems*, 36:19858–19886, 2023.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *International conference on machine learning*, pp. 242–252. PMLR, 2019.

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *Advances in neural information processing systems*, 32, 2019.

Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining neural scaling laws. *Proceedings of the National Academy of Sciences*, 121(27):e2311878121, 2024.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The journal of machine learning research*, 13(1):281–305, 2012.

Enric Boix-Adserà and Etai Littwin. Tight conditions for when the ntk approximation is valid. *Transactions on Machine Learning Research*, 2023.

Jiaxin Chen, Xiao-Ming Wu, Yanke Li, Qimai Li, Li-Ming Zhan, and Fu-lai Chung. A closer look at the training strategy for modern meta-learning. *Advances in neural information processing systems*, 33:396–406, 2020.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Eugenio Clerico and Benjamin Guedj. A note on regularised ntk dynamics with an application to pac-bayesian training. *Transactions on Machine Learning Research*, 2024.

Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, volume 15, pp. 3460–8, 2015.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, G Heigold, S Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

Aleksandr Dremov, Alexander Hägele, Atli Kosson, and Martin Jaggi. Training dynamics of the cooldown stage in warmup-stable-decay learning rate scheduler. *Transactions on Machine Learning Research*, 2025.

Simon Du, Jason Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *International conference on machine learning*, pp. 1675–1685. PMLR, 2019.

Cheng Gao, Yuan Cao, Zihao Li, Yihan He, Mengdi Wang, Han Liu, Jason Klusowski, and Jianqing Fan. Global convergence in training large-scale transformers. *Advances in Neural Information Processing Systems*, 37:29213–29284, 2024.

X Yu Geoffrey, Yubo Gao, Pavel Golikov, and Gennady Pekhimenko. Habitat: A {Runtime-Based} computational performance predictor for deep neural network training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 503–521, 2021.

Pulkit Gopalani, Samyak Jha, and Anirbit Mukherjee. Global convergence of sgd for logistic loss on two layer neural nets. *Transactions on Machine Learning Research*, 2024.

Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. In *International Conference on Learning Representations*, 2018.

Jiechao Guan, Yong Liu, and Zhiwu Lu. Fine-grained analysis of stability and generalization for modern meta learning algorithms. *Advances in Neural Information Processing Systems*, 35:18487–18500, 2022.

James Harrison, Luke Metz, and Jascha Sohl-Dickstein. A closer look at learned optimization: Stability, robustness, and inductive biases. *Advances in neural information processing systems*, 35:3758–3773, 2022.

Ali Hassani, Steven Walton, Nikhil Shah, Abulikemu Abuduweili, Jiachen Li, and Humphrey Shi. Escaping the big data paradigm with compact transformers. *arXiv preprint arXiv:2104.05704*, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Byeongho Heo, Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Junsuk Choe, and Seong Joon Oh. Rethinking spatial dimensions of vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 11936–11945, 2021.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

Kaiyi Ji, Jason D Lee, Yingbin Liang, and H Vincent Poor. Convergence of meta-learning with task-specific adaptation over partial parameters. *Advances in Neural Information Processing Systems*, 33:11490–11500, 2020.

Weisen Jiang, James Kwok, and Yu Zhang. Effective meta-regularization by kernelized proximal regularization. *Advances in Neural Information Processing Systems*, 34:26212–26222, 2021.

Daniel Justus, John Brennan, Stephen Bonner, and Andrew Stephen McGough. Predicting the computational cost of deep learning models. In *2018 IEEE international conference on big data (Big Data)*, pp. 3873–3882. IEEE, 2018.

Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning curve prediction with bayesian neural networks. In *International conference on learning representations*, 2017.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images.(2009), 2009.

Akshay Kumar and Jarvis Haupt. Early directional convergence in deep homogeneous neural networks for small initializations. *Transactions on Machine Learning Research*, 2025, 2025.

Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32, 2019.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18 (185):1–52, 2018.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P11-1015`.

Pierre Marion and Raphaël Berthier. Leveraging the two-timescale regime to demonstrate convergence of neural networks. *Advances in Neural Information Processing Systems*, 36:64996–65029, 2023.

Fangzhou Mu, Yingyu Liang, and Yin Li. Gradients as features for deep representation learning. In *8th International Conference on Learning Representations, ICLR 2020*, 2020.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 7. Granada, 2011.

Jack Parker-Holder, Vu Nguyen, and Stephen J Roberts. Provably efficient online hyperparameter optimization with population-based bandits. *Advances in neural information processing systems*, 33:17200–17211, 2020.

Alireza Pourali, Arian Boukani, and Hamzeh Khazaei. PreNeT: Leveraging computational features to predict deep neural network training time. In *Proceedings of the 16th ACM/SPEC International Conference on Performance Engineering*, ICPE '25, pp. 81–91, 2025.

Tiancheng Qin, S Rasoul Etesami, and Cesar A Uribe. Faster convergence of local sgd for over-parameterized models. *Transactions on Machine Learning Research*, 2022.

Basri Ronen, David Jacobs, Yoni Kasten, and Shira Kritchman. The convergence rate of neural networks for learned functions of different frequencies. *Advances in Neural Information Processing Systems*, 32, 2019.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Jihoon Tack, Jongjin Park, Hankook Lee, Jaeho Lee, and Jinwoo Shin. Meta-learning with self-improving momentum target. *Advances in Neural Information Processing Systems*, 35:6318–6332, 2022.

Bhavya Vasudeva, Puneesh Deora, and Christos Thrampoulidis. Implicit bias and fast convergence rates for self-attention. *Transactions on Machine Learning Research*, 2024.

Nikhil Vyas, Yamini Bansal, and Preetum Nakkiran. Empirical limitations of the ntk for understanding scaling laws in deep learning. *Transactions on Machine Learning Research*, 2023.

Ruosi Wan, Zhanxing Zhu, Xiangyu Zhang, and Jian Sun. Spherical motion dynamics: Learning dynamics of normalized neural network using sgd and weight decay. *Advances in Neural Information Processing Systems*, 34:6380–6391, 2021.

Haonan Wang, Wei Huang, Ziwei Wu, Hanghang Tong, Andrew J Margenot, and Jingrui He. Deep active learning by leveraging training dynamics. *Advances in Neural Information Processing Systems*, 35:25171–25184, 2022.

Mingze Wang and Chao Ma. Early stage convergence and global convergence of training mildly parameterized neural networks. *Advances in Neural Information Processing Systems*, 35:743–756, 2022.

Simon Weissmann, Sara Klein, Waïss Azizian, and Leif Döring. Almost sure convergence of stochastic gradient methods under gradient domination. *Transactions on Machine Learning Research*, 2025.

Mengzhou Xia, Antonios Anastasopoulos, Ruochen Xu, Yiming Yang, and Graham Neubig. Predicting performance for natural language processing tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Feiyang Ye, Baijiong Lin, Zhixiong Yue, Pengxin Guo, Qiao Xiao, and Yu Zhang. Multi-objective meta learning. *Advances in Neural Information Processing Systems*, 34:21338–21351, 2021.

Luca Zancato, Alessandro Achille, Avinash Ravichandran, Rahul Bhotika, and Stefano Soatto. Predicting training time without training. *Advances in Neural Information Processing Systems*, 33:6136–6146, 2020.

Arber Zela, Julien Niklas Siems, Lucas Zimmer, Jovita Lukasik, Margret Keuper, and Frank Hutter. Surrogate nas benchmarks: Going beyond the limited search spaces of tabular nas benchmarks. In *International Conference on Learning Representations*, 2020.

Qi Zhang, Yi Zhou, and Shaofeng Zou. Convergence guarantees for rmsprop and adam in generalized-smooth non-convex optimization with affine noise variance. *Transactions on machine learning research*, 2025.

Yilang Zhang, Alireza Sadeghi, and Georgios Giannakis. Meta-learning universal priors using non-injective change of variables. *Advances in Neural Information Processing Systems*, 37:111965–111993, 2024a.

Yu-Ming Zhang, Jun-Wei Hsieh, Xin Li, Ming-Ching Chang, Chun-Chieh Lee, and Kuo-Chin Fan. Mote-nas: multi-objective training-based estimate for efficient neural architecture search. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, pp. 100845–100869, 2024b.

Pan Zhou, Xiaotong Yuan, Huan Xu, Shuicheng Yan, and Jiashi Feng. Efficient meta learning via minibatch proximal update. *Advances in Neural Information Processing Systems*, 32, 2019.

Zining Zhu, Soroosh Shahtalebi, and Frank Rudzicz. Predicting fine-tuning performance with probing. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11534–11547, 2022.

Difan Zou, Yuan Cao, Dongruo Zhou, and Quanquan Gu. Gradient descent optimizes over-parameterized deep relu networks. *Machine learning*, 109:467–492, 2020.

A Appendix

A.1 Proof of Proposition 1: Asymptotic Consistency

Proposition 1 (Asymptotic Consistency). Assume that:

- The probing feature extractor $\phi(f_\theta, \mathcal{D}_{\text{probe}})$ produces bounded feature vectors $\mathbf{z} \in \mathbb{R}^k$,
- The ground-truth mapping $h^*(\mathbf{z}) := T^*$ is measurable,
- The hypothesis class $\mathcal{G}$ contains a sequence of functions that can approximate h^* arbitrarily well (i.e., $\mathcal{G}$ is dense in $L^2(P_Z)$),
- The training examples $(\mathbf{z}^{(j)}, T^{*(j)})$ are i.i.d. samples from a fixed distribution $\mathcal{D}_\mathcal{M}$.

Then, the empirical risk minimizer

$$g_M \in \arg\min_{g \in \mathcal{G}} \frac{1}{M} \sum_{j=1}^{M} \left(\log g(\mathbf{z}^{(j)}) - \log T^{*(j)} \right)^2$$

satisfies

$$\lim_{M\to\infty} \mathbb{E}_{(\mathbf{z},T^*)} \left[(\log g_M(\mathbf{z}) - \log T^*)^2 \right] = \inf_{g\in\mathcal{G}} \mathbb{E}_{(\mathbf{z},T^*)} \left[(\log g(\mathbf{z}) - \log T^*)^2 \right]$$

That is, g_M is a consistent estimator of the log-step predictor.

Proof. Let $\mathcal{Z} = \mathbb{R}^k \times \mathbb{R}_{>0}$, where each sample $(\mathbf{z}, T^*) \sim \mathcal{D}_\mathcal{M}$. Define the per-example loss function as

$$L(g; \mathbf{z}, T^*) = (\log g(\mathbf{z}) - \log T^*)^2$$

assuming $g(\mathbf{z}) > 0$ to ensure the logarithm is well-defined.

Define the population risk:

$$\mathcal{R}(g) := \mathbb{E}_{(\mathbf{z},T^*)\sim\mathcal{D}_\mathcal{M}} \left[L(g; \mathbf{z}, T^*) \right]$$

and the empirical risk over M samples:

$$\hat{\mathcal{R}}_M(g) := \frac{1}{M} \sum_{j=1}^{M} L(g; \mathbf{z}^{(j)}, T^{*(j)})$$

Let $g_M \in \arg\min_{g\in\mathcal{G}} \hat{\mathcal{R}}_M(g)$. We aim to show that

$$\mathcal{R}(g_M) \to \inf_{g\in\mathcal{G}} \mathcal{R}(g) \quad \text{as } M \to \infty$$

Step 1: Uniform Convergence. If the loss class $\mathcal{L}_\mathcal{G} := \{(\mathbf{z}, T^*) \mapsto L(g; \mathbf{z}, T^*) \mid g \in \mathcal{G}\}$ has finite pseudo-dimension or bounded covering number, then by the uniform law of large numbers:

$$\sup_{g\in\mathcal{G}} \left| \hat{\mathcal{R}}_M(g) - \mathcal{R}(g) \right| \xrightarrow{P} 0 \quad \text{as } M \to \infty$$

This holds if g is Lipschitz (e.g., MLPs with bounded weights), and both $\log T^*$ and $\log g(\mathbf{z})$ are bounded, e.g., via clipping or regularization.

Step 2: Approximation Error. By assumption, there exists $g^* \in \mathcal{G}$ such that $\mathcal{R}(g^*) = \inf_{g\in\mathcal{G}} \mathcal{R}(g)$. Therefore, $\mathcal{G}$ can approximate the Bayes optimal predictor arbitrarily closely.

Step 3: Conclude Consistency. From uniform convergence and richness of $\mathcal{G}$, we have:

$$\lim_{M\to\infty} \mathcal{R}(g_M) = \inf_{g\in\mathcal{G}} \mathcal{R}(g)$$

which implies consistency of the meta-regressor:

$$\lim_{M\to\infty} \mathbb{E}_{(\mathbf{z},T^*)} \left[(\log g_M(\mathbf{z}) - \log T^*)^2 \right] = \inf_{g\in\mathcal{G}} \mathcal{R}(g)$$

A.2 Supporting Lemma: Generalization of Empirical Risk Minimization

Lemma 1 (ERM Convergence). Let $\mathcal{G}$ be a hypothesis class of real-valued functions mapping from $\mathbb{R}^k$ to $\mathbb{R}_{>0}$, and assume that:

- The per-sample loss is given by $L(g; \mathbf{z}, T^*) = (\log g(\mathbf{z}) - \log T^*)^2$,
- The examples $(\mathbf{z}, T^*)$ are i.i.d. from a fixed distribution $\mathcal{D}_{\mathcal{M}}$,
- The function class $\mathcal{G}$ has finite VC-dimension or bounded Rademacher complexity,
- $\log g(\mathbf{z})$ and $\log T^*$ are uniformly bounded almost surely.

Then, the empirical risk minimizer

$$g_M = \arg\min_{g \in \mathcal{G}} \frac{1}{M} \sum_{j=1}^{M} \left(\log g(\mathbf{z}^{(j)}) - \log T^{*(j)} \right)^2$$

satisfies with high probability:

$$\left| \mathbb{E}\left[L(g_M; \mathbf{z}, T^*) \right] - \inf_{g \in \mathcal{G}} \mathbb{E}\left[L(g; \mathbf{z}, T^*) \right] \right| \leq \epsilon(M)$$

where $\epsilon(M) \to 0$ as $M \to \infty$.

Proof. This follows from standard learning theory results. Since the loss is bounded and Lipschitz in $\log g(\mathbf{z})$, and $\mathcal{G}$ has finite capacity, uniform convergence holds over $\mathcal{L}_{\mathcal{G}} = \{L(g; \cdot, \cdot) : g \in \mathcal{G}\}$:

$$\sup_{g \in \mathcal{G}} \left| \hat{\mathcal{R}}_M(g) - \mathcal{R}(g) \right| \to 0 \quad \text{as } M \to \infty$$

Then, by consistency of ERM under uniform convergence, the excess risk of g_M over the best-in-class predictor vanishes:

$$\mathcal{R}(g_M) - \inf_{g \in \mathcal{G}} \mathcal{R}(g) \to 0$$

A.3 Justification of Probing Features

Our feature set $\mathbf{z}$ is constructed to reflect known factors that influence convergence dynamics in neural networks. Below, we justify the inclusion of each feature:

- **Parameter Count** P: Determines model capacity. Larger networks typically have longer optimization paths unless regularized or trained with adaptive learning rates.
- **Gradient Norm** g^2: Related to the sharpness of the loss surface. Steeper gradients at initialization suggest faster early learning, as shown in gradient descent convergence bounds (see e.g., Du et al. (2019)).
- **NTK Trace Proxy** τ: Serves as a computationally efficient proxy for the Neural Tangent Kernel trace. The NTK governs the linearized training dynamics of overparameterized networks and has been shown to correlate with convergence rates in theory (Jacot et al., 2018).
- **Dataset Size** N: Influences generalization and total gradient noise. Larger datasets generally increase the number of steps required to reach a loss threshold.
- **Learning Rate** η and **Batch Size** B: These hyperparameters directly affect the effective step size and noise scale in stochastic optimization. Their effects are theoretically characterized in SGD convergence analysis.

Thus, our probing feature vector is both computationally lightweight and theoretically grounded in optimization and generalization theory.

A.4 Meta-Model Training and Evaluation Details

In this work, we trained several machine learning regressors on the CAPE-derived meta-dataset, with the objective of predicting the convergence time (T^*) from the feature set $\{\log P, \log B, \log G^2, \log \tau, \log LR, \log N\}$. To ensure a comprehensive comparison, we employed models spanning linear, kernel-based, tree-based, boosting, and neural network families. The configurations used are summarized below.

- **Linear Regression.** A standard least-squares regressor that assumes a linear mapping between features and target. Used as the baseline model without regularization.
- **Ridge Regression.** A linear regressor with L_2 regularization to mitigate overfitting. We set the regularization strength to $\alpha = 1.0$.
- **Random Forest Regressor.** An ensemble of decision trees trained via bootstrap aggregation. We used 300 estimators with maximum depth set to 12. Random seed was fixed to 42 and parallel training was enabled.
- **Support Vector Regression (SVR).** We used the radial basis function (RBF) kernel with parameters $C = 10.0$, $\epsilon = 0.1$, and $\gamma =$ scale. This allows capturing smooth non-linear dependencies in the meta-features.
- **Extreme Gradient Boosting (XGBoost).** Gradient boosted regression trees were configured with 500 estimators, maximum depth of 6, learning rate $\eta = 0.05$, subsample ratio 0.9, and column sampling ratio 0.9.
- **Multilayer Perceptron Regressor (MLPRegressor).** A feed-forward neural network with two hidden layers of sizes $(256, 128)$, ReLU activations, the Adam optimizer, L_2 penalty $\alpha = 10^{-4}$, adaptive learning rate scheduling with initial learning rate 10^{-3}, and a maximum of 500 iterations. Early stopping was enabled to avoid overfitting.

Each model was trained on the same meta-dataset extracted from four benchmark datasets (MNIST, Fashion-MNIST, CIFAR-10, CIFAR-100). Evaluation was performed across grids of learning rates, batch sizes, and tolerance values (ϵ). For each combination, ten trials with fresh data batches were executed to reduce stochastic variance. Results were aggregated at multiple levels (overall, per-dataset, per-learning rate, and per-batch size) and the corresponding outcomes are reported in the Table 4.

Meta-regressor	**MLP**	**CNN**	**RNN**	**Transformer**
Extreme Gradient Boosting (XGBoost)	1.98	2.89	3.79	8.71
Random Forest Regressor	2.27	3.04	3.99	9.08
Multilayer Perceptron Regressor (MLPRegressor)	3.42	4.05	5.69	10.47
Support Vector Regression (SVR)	3.68	7.45	7.48	12.66
Ridge Regression	3.72	8.31	7.98	10.77
Linear Regression	3.82	8.30	7.90	11.12

Table 4: Overall Meta-Regressor Performance across architectures