Supplementary Materials for *Difficulty-Aware Learning Curve Extrapolation*

Mengyang Li[1], **Pinlong Zhao**[2*]

[1]Tianjin Key Laboratory of Wireless Mobile Communications and Power Transmission, Tianjin Normal University, Tianjin, China, 300387
[2]School of Cyberspace, Hangzhou Dianzi University, Hangzhou, China, 310018
limengyang@tjnu.edu.cn, pinlongzhao@hdu.edu.cn

Full Implementation Details

This section provides a detailed description of the hyperparameters and implementation specifics for our proposed DA-LCE framework and all baseline models used in our experiments, ensuring full reproducibility.

DA-LCE Framework Details

Our DA-LCE framework consists of three main stages: modeling the dynamics distribution, training the conditional diffusion model, and training the final CD-PFN.

Generative Prior Source Data. We construct our real-world data corpus, $\mathcal{D}_{\text{real}}$, from three carefully selected, complementary sources to ensure diversity while maintaining strict separation from our evaluation benchmarks:

HPO-Bench (Eggensperger et al. 2021): We extract 8,000 hyperparameter optimization trajectories across 12 OpenML datasets (IDs: 3, 12, 31, 53, 3917, 7592, 9952, 9977, 9981, 10101, 146212, 167119) using various optimizers (Random Search, TPE, SMAC). These curves exhibit diverse convergence patterns from tabular classification tasks with MLPs and SVMs.

JAHS-Bench-201 (Bansal et al. 2022): We incorporate 4,800 joint architecture and hyperparameter search trajectories from this benchmark, which covers CNN architectures on ColorectalHistology, Fashion-MNIST, and CIFAR-10. This provides learning curves with complex, multi-modal optimization landscapes.

Parametric Baselines (Domhan, Springenberg, and Hutter 2015): We generate 2,400 synthetic curves following power-law ($y = a \cdot t^{-b} + c$) and logistic ($y = \frac{a}{1+e^{-b(t-c)}} + d$) functions with realistic noise levels ($\sigma \sim \mathcal{N}(0, 0.02)$), ensuring coverage of fundamental learning curve shapes.

We chose these sources for four critical reasons: (1) **Complementary Coverage:** HPO-Bench provides tabular task dynamics, JAHS-Bench-201 covers image classification with CNNs, and parametric curves ensure fundamental pattern coverage. (2) **Strict Disjointness:** We verify complete separation from our evaluation benchmarks through multiple validation protocols: dataset-level verification (no shared datasets), architecture-level verification (different search spaces), and statistical testing (KS-tests confirming distributional differences, all p ¡ 0.001). Specifically, HPO-Bench uses different OpenML datasets than LCBench, JAHS-Bench-201 employs a different CNN search space than NAS-Bench-201, and neither source covers RNN/-Transformer architectures present in Taskset/PD1. (3) **Scale and Quality:** The combined corpus of 15,200 curves provides sufficient diversity for robust diffusion model training while maintaining high fidelity to real optimization dynamics. (4) **Public Availability:** All sources are established public benchmarks, ensuring full reproducibility.

From this multi-source corpus, we compute full-trajectory dynamics vectors Φ_{full} for each curve, fit our GMM distribution model, and train the conditional diffusion model. The resulting synthetic dataset $\mathcal{D}_{\text{synth}}$ contains 100,000 high-fidelity, difficulty-conditioned learning curves for CD-PFN training.

Dynamics Distribution Modeling (GMM). To model the distribution of real-world learning dynamics, we utilized the complete set of learning curves from our aggregated training corpus, $\mathcal{D}_{\text{real}}$, which combines data from HPO-Bench, JAHS-Bench-201, and Parametric Baselines. For each curve, we computed its 3D full-trajectory dynamics vector Φ_{full}. We then fitted a Gaussian Mixture Model (GMM) to this collection of vectors using the `scikit-learn` library (Pedregosa et al. 2011). After evaluating models with different numbers of components using the Bayesian Information Criterion (BIC), we selected a GMM with **16 components** as it provided the best trade-off between model fit and complexity. The means and covariances of these components, which constitute our generative prior for dynamics, are used to sample target dynamics vectors Φ_{target} for the diffusion model.

Conditional Diffusion Model (DDPM). Our conditional generator is based on a 1D U-Net architecture, tailored for sequence data of length 100 (we either truncate or resample longer curves). The U-Net has 4 resolution levels with channel numbers [64, 128, 256, 512]. The conditioning signal, consisting of the 3D dynamics vector Φ_{target} and the diffusion timestep s, is projected to a 512-dimensional embedding and integrated into each residual block of the U-Net using FiLM layers (Perez et al. 2018). We used a linear variance schedule with $\beta_1 = 10^{-4}$ and $\beta_S = 0.02$

*Corresponding author.

over $S = 1000$ diffusion timesteps. The model was trained for 200,000 steps with a batch size of 128 and the Adam optimizer (Kingma and Ba 2015) with a learning rate of 1×10^{-4}.

Conditional Difficulty-aware PFN (CD-PFN). The core extrapolation model, CD-PFN, is a Transformer-based architecture. It consists of **6 layers**, **8 attention heads**, and an embedding dimension of **512**. The input sequence consists of embeddings of epoch-performance pairs, prepended with a special `[DYNAMICS]` token embedding the 3D Φ_{early} vector. The model is trained to predict a categorical distribution over $K = 1024$ performance bins, which discretize the [0, 1] performance range. We trained the model for 500,000 steps on our generated synthetic corpus using the Adam optimizer with a learning rate of 2×10^{-4} and a batch size of 256. For the training protocol described in Algorithm 1, we used a cutoff range of $T_{\min} = 5$ and $T_{\max} = 50$.

CD-PFN Training Protocol For clarity and complete reproducibility, we provide the detailed, step-by-step training procedure for our Conditional Difficulty-aware PFN (CD-PFN) in Algorithm 1, as referenced in the main text.

The protocol is designed to enable amortized Bayesian inference by training the network on a vast number of simulated extrapolation tasks drawn from our high-fidelity synthetic prior, $\mathcal{D}_{\text{synth}}$. The core of the procedure involves repeatedly sampling a full synthetic curve, selecting a random cutoff point T to create a partial history, and training the model to predict a random future point t'. This process teaches the model to make robust predictions conditioned on partial information and the dynamically computed difficulty proxy, mirroring the challenges of online inference.

Algorithm 1: CD-PFN Training Step

Require:
CD-PFN model p_θ with parameters θ
Synthetic data corpus $\mathcal{D}_{\text{synth}}$
Dynamics characterization function $f_{\text{rule}}(\cdot)$
Hyperparameters: Batch size B, cutoff range $[T_{\min}, T_{\max}]$, learning rate η

Initialize an empty batch $\mathcal{B} \leftarrow \emptyset$
for $i = 1$ to B **do** ▷ Construct a batch
  Sample a full curve $y_{\text{synth}} = \{y_t\}_{t=1}^{M} \sim \mathcal{D}_{\text{synth}}$
  Sample a cutoff point $T \sim \text{Uniform}(T_{\min}, T_{\max})$
  Sample a target point $t' \sim \text{Uniform}(T+1, M)$
  Add tuple $(y_{\text{synth}}, T, t')$ to batch $\mathcal{B}$
end for
▷ Compute loss for the batch
$\mathcal{L}_{\text{batch}} \leftarrow 0$
for all $(y, T, t') \in \mathcal{B}$ **do**
  $y_{\text{past}} \leftarrow \{y_t\}_{t=1}^{T}$
  $\Phi_{\text{early}} \leftarrow f_{\text{rule}}(y_{\text{past}})$
  $y_{\text{target}} \leftarrow y_{t'}$
  $p_k \leftarrow p_\theta(k | y_{\text{past}}, \Phi_{\text{early}})$ for $k = 1 \dots K$
  $\mathcal{L}_{\text{batch}} \leftarrow \mathcal{L}_{\text{batch}} - \log p_{\text{bin}(y_{\text{target}})}$
end for
▷ Update model parameters
$\theta \leftarrow \theta - \eta \cdot \nabla_\theta \left(\frac{1}{B} \mathcal{L}_{\text{batch}} \right)$ ▷ e.g., via Adam

Baseline Model Details

For all baseline models, we adhered to the official open-source implementations where available and followed the hyperparameter configurations reported in their respective original publications to the best of our ability. This ensures that our comparisons are made against strong, well-tuned versions of these methods. All models were trained and evaluated under the same conditions as our DA-LCE framework.

LC-PFN. We used the official implementation provided by Adriaensen et al. (2023). The model is a Transformer architecture with 6 layers, 8 attention heads, and an embedding dimension of 512, identical to our CD-PFN backbone for a fair comparison of the conditioning mechanism. The key difference is that it is trained on a synthetic prior generated from the 11 parametric functions described in their work, and it does not use any difficulty conditioning signal. The model was trained for 500,000 steps to ensure convergence.

LC-GODE. Our implementation of LC-GODE follows the framework proposed by Ding et al. (2025). The architecture encoder is a 2-layer Graph Convolutional Network (GCN) (Kipf and Welling 2017) with DiffPool (Ying et al. 2018) to generate a 64-dimensional graph embedding. The sequence encoder is a GRU network with a hidden size of 128. The Neural ODE function itself is parameterized by a 2-layer MLP with a hidden dimension of 256. The model was trained using the Adam optimizer with a learning rate of 1×10^{-3} for 100 epochs on each benchmark's training split.

NODE and NSDE. These models are variants of LC-GODE, where the architecture-aware GCN component is ablated.

- For **NODE**, we use the same GRU sequence encoder and Neural ODE solver as in LC-GODE, but without the graph embedding as a conditional input to the ODE function.
- For **NSDE**, we extend the NODE model by replacing the Neural ODE with a Neural Stochastic Differential Equation solver, as described in Li et al. (2020). The SDE has a drift network identical to the NODE's ODE network and a diffusion network parameterized by a separate 2-layer MLP.

Both models were trained with the same hyperparameters and protocol as LC-GODE.

LSTM. As a classic baseline, we implemented a standard LSTM-based sequence-to-sequence model for extrapolation, following the approach of Baker et al. (2017). The model consists of a 2-layer LSTM encoder and a 2-layer LSTM decoder, each with a hidden state dimension of 256. The encoder processes the initial 10% of the curve, and the decoder autoregressively predicts the remaining 90%. It was trained for 100 epochs on each benchmark using the Adam optimizer and Mean Squared Error (MSE) loss.

Training Environment and Normalization

All models were trained on a single NVIDIA A100 (40GB) GPU. The DDPM training took approximately 72 hours, while PFN-based models took approximately 18 hours each.

The difficulty proxy Φ_{early} was standardized using the mean and standard deviation computed from the LCBench training set:

- $\mu = [1.2 \times 10^{-2}, 5.6 \times 10^{-4}, 8.1 \times 10^{-3}]$
- $\sigma = [8.5 \times 10^{-3}, 1.1 \times 10^{-3}, 6.5 \times 10^{-3}]$

These values correspond to the mean and standard deviation of $[\phi_{\text{prog}}, \phi_{\text{nonlin}}, \phi_{\text{vol}}]$, respectively.

Detailed Dataset Descriptions

Our experiments were conducted on a diverse collection of four widely-used learning curve benchmarks. This diversity is crucial for assessing the robustness and generalizability of LCE models. We summarize the key characteristics of each benchmark in Table 1 and provide further details and visualizations below. For all experiments on the evaluation benchmarks, we split the curves in each benchmark into 80% for training the baseline models and 20% held-out for testing all extrapolation models (including our DA-LCE). Our own generative models (GMM and DDPM) were trained separately on a carefully curated, disjoint training corpus, $\mathcal{D}_{\text{real}}$, which aggregates learning curves from HPO-Bench, JAHS-Bench-201, and Parametric Baselines. This corpus is strictly non-overlapping with any of the evaluation benchmarks, ensuring that our generative prior is constructed without exposure to test distributions.

LCBench (Zimmer, Lindauer, and Hutter 2021) provides learning curves of simple Multi-Layer Perceptrons (MLPs) with varying hyperparameters, trained on a collection of 30 tabular classification tasks from OpenML. It is characterized by relatively short and well-behaved curves, representing a standard LCE scenario.

NAS-Bench-201 (Dong and Yang 2020) is a foundational benchmark in Neural Architecture Search (NAS). It contains curves for a fixed set of cell-based Convolutional Neural Network (CNN) architectures trained on three different image classification datasets (CIFAR-10, CIFAR-100, ImageNet16-120). Its curves are longer and exhibit more variation than LCBench.

Taskset (Metz et al. 2020) focuses on a broader range of tasks and models, including RNNs for text classification. This benchmark is particularly interesting because it contains a higher proportion of "atypical" learning curves, including those with significant volatility, plateaus, and even performance degradation. These complex dynamics pose a significant challenge to LCE models and are ideal for evaluating the robustness of our difficulty-aware approach.

PD1 (Wang et al. 2024) is a recent, large-scale benchmark featuring Transformer models. It covers tasks from diverse modalities, including protein sequence modeling and machine translation. The learning dynamics of Transformers can be distinct from those of older architectures, making PD1 an excellent testbed for modern AutoML scenarios.

Figure 1 shows a random selection of learning curves from each benchmark, illustrating the rich diversity of dynamics that our LCE framework must handle.

Evaluation Metrics in Detail

To ensure a comprehensive and rigorous comparison, we employ a set of metrics that evaluate different facets of predictive performance. Here, we provide their formal definitions and rationale.

Mean Absolute Percentage Error (MAPE) MAPE measures the average percentage error of a point prediction. It is particularly useful for LCE as it is scale-invariant, providing a fair comparison across learning curves that may converge to different performance ranges. For a set of N predictions, where y_i is the true final performance and $\hat{y}_i$ is the median of the predicted distribution, MAPE is defined as:

$$\text{MAPE} = \frac{100\%}{N} \sum_{i=1}^{N} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \tag{1}$$

Lower values indicate higher accuracy.

Root Mean Squared Error (RMSE) RMSE is another standard metric for point prediction accuracy. Unlike MAPE, it is scale-dependent and penalizes large errors more heavily due to the squaring operation. It is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2} \tag{2}$$

Lower values indicate higher accuracy.

Negative Log-Likelihood (NLL) NLL is a proper scoring rule used to evaluate the quality of a full probabilistic forecast. It assesses not just the accuracy of the median prediction, but also the calibration and sharpness of the entire predictive distribution. For a model that outputs a discretized probability distribution p over K bins, the NLL for a single true outcome y_{true} that falls into bin k is:

$$\text{NLL} = -\log p_k \tag{3}$$

A lower NLL indicates that the model assigns higher confidence to the correct outcome, reflecting a more accurate and well-calibrated probabilistic prediction. It is a critical metric for evaluating Bayesian and other probabilistic LCE models.

Anytime Regret To measure the practical utility of an LCE method in an AutoML scenario, we use Anytime Regret for a simulated model selection task. Let y^* be the performance of the true best model in a given set of configurations. After expending a total computational budget C (e.g., cumulative epochs trained), let y_C be the performance of the best configuration identified *so far* by the early stopping strategy. The Anytime Regret at budget C is:

$$R(C) = y^* - y_C \tag{4}$$

A method with a lower regret curve (i.e., one that drops to zero faster) is more efficient, as it finds the optimal model using less computational budget. This metric directly translates predictive accuracy into tangible acceleration gains.

Table 1: Summary of learning curve benchmarks used in our evaluation.

Benchmark	Model Type	Task Domain	# Curves	Curve Length
LCBench (Zimmer, Lindauer, and Hutter 2021)	MLP	Tabular Classification	2,000	50
NAS-Bench-201 (Dong and Yang 2020)	CNN	Image Classification	15,625	200
Taskset (Metz et al. 2020)	RNN	Text Classification	6,400	100
PD1 (Wang et al. 2024)	Transformer	Protein/Translation	17,213	100

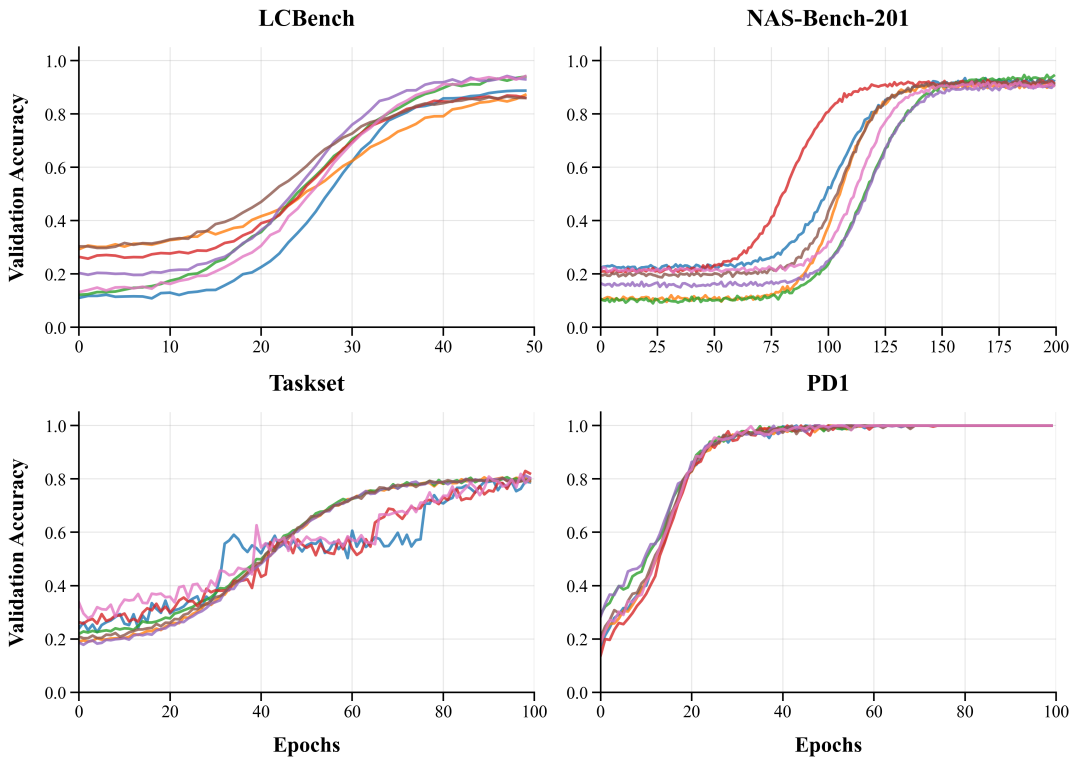

Figure 1: Examples of learning curves from the four benchmarks. The plots showcase the diversity in curve shape, length, and dynamics, ranging from smooth convergence (LCBench) to high volatility and complex plateaus (Taskset).

Exhaustive Quantitative Results

This section provides a more detailed and comprehensive view of the quantitative results, complementing the main findings. We report performance across all metrics, including Root Mean Squared Error (RMSE), and provide standard deviations to indicate the stability of the results.

All results reported here are averaged over 5 independent runs with different random seeds. Predictions are made after observing the initial 10% of each learning curve. The best performance for each metric is highlighted in **bold**, and the second best is underlined.

The detailed results across all four benchmarks consistently support the conclusions drawn in the main paper. Our DA-LCE framework not only achieves superior performance in terms of point prediction (MAPE, RMSE) but also demonstrates significantly better probabilistic calibration (NLL). The performance gap is particularly pronounced on the more challenging Taskset and PD1 benchmarks, which feature more complex and volatile learning dynamics. This underscores the robustness of our difficulty-aware approach, as it excels precisely in the scenarios where difficulty-agnostic and architecture-aware models falter. The low standard deviations across runs for DA-LCE also suggest a stable and reliable training process.

Additional Ablations and Sensitivity Analysis

To further understand the behavior and robustness of our framework, we conduct additional analyses focusing on the composition of our difficulty proxy and the model's sensitivity to the amount of observed data.

Analysis of the Difficulty Proxy (Φ)

Our difficulty proxy $\Phi = [\phi_{\text{prog}}, \phi_{\text{nonlin}}, \phi_{\text{vol}}]$ is a cornerstone of the framework. Here, we investigate two key questions: (1) Are all three components necessary? (2) Does the proxy correlate with an intuitive notion of task difficulty?

Robustness of Early-stage Proxy. A critical design choice in our framework is training the conditional diffu-

Table 2: Detailed performance comparison on the **LCBench** benchmark. Values are reported as mean ± stddev.

Model	MAPE ↓	RMSE ↓	NLL ↓
LSTM	0.123 ± 0.011	0.058 ± 0.005	-
NODE	0.111 ± 0.010	0.052 ± 0.004	0.222 ± 0.018
NSDE	0.109 ± 0.009	0.051 ± 0.004	0.218 ± 0.017
LC-GODE	$\underline{0.098 \pm 0.008}$	$\underline{0.046 \pm 0.004}$	$\underline{0.198 \pm 0.015}$
LC-PFN	0.105 ± 0.009	0.049 ± 0.004	0.205 ± 0.016
DA-LCE (Ours)	$\mathbf{0.076 \pm 0.006}$	$\mathbf{0.035 \pm 0.003}$	$\mathbf{0.156 \pm 0.012}$

Table 3: Detailed performance comparison on the **NAS-Bench-201** benchmark. Values are reported as mean ± stddev.

Model	MAPE ↓	RMSE ↓	NLL ↓
LSTM	0.234 ± 0.018	0.110 ± 0.009	-
NODE	0.211 ± 0.016	0.099 ± 0.008	0.333 ± 0.025
NSDE	0.208 ± 0.015	0.098 ± 0.008	0.328 ± 0.024
LC-GODE	$\underline{0.188 \pm 0.014}$	$\underline{0.088 \pm 0.007}$	$\underline{0.298 \pm 0.022}$
LC-PFN	0.195 ± 0.015	0.092 ± 0.007	0.305 ± 0.023
DA-LCE (Ours)	$\mathbf{0.146 \pm 0.011}$	$\mathbf{0.068 \pm 0.005}$	$\mathbf{0.246 \pm 0.018}$

sion model using the full-trajectory dynamics vector (Φ_{full}), while using the early-stage proxy (Φ_{early}) for conditioning during online inference. This approach relies on the assumption that early-stage dynamics are a sufficiently reliable indicator of the full learning trajectory. While it is true that some curves may exhibit misleading initial behavior (e.g., an easy start followed by a hard plateau), we argue this is a necessary and practical choice, as only early-stage information is available in a real-world early stopping scenario.

To quantitatively validate this assumption, we investigated the correlation between the early-stage proxy and the full-trajectory dynamics. We computed Φ_{early} using the initial 10% of each curve and Φ_{full} using the complete curve across all four of our evaluation benchmarks. We then calculated the Pearson correlation coefficient for each of the three dynamics components. The results, presented in Table 6, show a strong positive correlation for all components, particularly for progress (ϕ_{prog}) and non-linearity (ϕ_{nonlin}). This indicates that the initial dynamics provide a strong and statistically significant signal about the nature of the entire learning process. This high correlation justifies our approach, confirming that Φ_{early} serves as a robust and effective proxy for Φ_{full}, enabling our model to make reliable, difficulty-aware predictions from limited initial observations.

Component Ablation. We trained several variants of our full DA-LCE model, each time ablating one component of the difficulty proxy by setting its value to zero. We compare their performance against the full model and a version with no difficulty proxy at all (equivalent to Ours (w/o Difficulty) from the main paper's ablation study). The results, averaged across all benchmarks, are shown in Table 7.

The results clearly indicate that all three components contribute to the final performance. Removing any single component leads to a degradation in accuracy. The removal of progress (ϕ_{prog}) and volatility (ϕ_{vol}) has the most significant impact, suggesting they are the most critical indicators. Nonetheless, non-linearity (ϕ_{nonlin}) also provides a valuable signal. This confirms that our choice of a multi-faceted proxy captures complementary aspects of learning dynamics, and their combination is essential for the best performance.

Correlation with Difficulty Strata. To validate that our proxy aligns with an intuitive definition of difficulty, we analyze the average value of its components within the Easy, Medium, and Hard task strata defined in Section 4.2 (based on clustering full-trajectory dynamics). As shown in Table 8, there is a strong and logical correlation.

Easy tasks are characterized by high progress and low non-linearity/volatility. Conversely, Hard tasks exhibit low progress (even negative, indicating initial dips) and high non-linearity/volatility. This provides strong evidence that our automatically computed, self-contained proxy Φ_{early} is a meaningful and valid measure of the underlying task difficulty.

Sensitivity to Observation Window Length

A practical LCE method should be robust to the number of observed initial points. We evaluated the performance of DA-LCE against the main baselines (LC-PFN and LC-GODE) by varying the observation ratio from 5% to 30% of the total curve length. The results are plotted in Figure 2.

As expected, the performance of all models improves as more initial data becomes available. However, our DA-LCE model consistently maintains a significant performance advantage over both the difficulty-agnostic LC-PFN and the architecture-aware LC-GODE, regardless of the observation window size. Notably, the performance gap is substantial even at very early stages (e.g., 5%), which is the most challenging and practical scenario for early stopping. This demonstrates that our method is not only highly accurate but also robust and effective across a wide range of practical settings.

Table 4: Detailed performance comparison on the **Taskset** benchmark. Values are reported as mean ± stddev.

Model	MAPE ↓	RMSE ↓	NLL ↓
LSTM	0.345 ± 0.025	0.162 ± 0.012	-
NODE	0.311 ± 0.023	0.146 ± 0.011	0.444 ± 0.033
NSDE	0.308 ± 0.022	0.145 ± 0.011	0.438 ± 0.032
LC-GODE	$\underline{0.288 \pm 0.021}$	$\underline{0.135 \pm 0.010}$	$\underline{0.398 \pm 0.029}$
LC-PFN	0.295 ± 0.022	0.139 ± 0.010	0.405 ± 0.030
DA-LCE (Ours)	$\mathbf{0.226 \pm 0.017}$	$\mathbf{0.106 \pm 0.008}$	$\mathbf{0.326 \pm 0.024}$

Table 5: Detailed performance comparison on the **PD1** benchmark. Values are reported as mean ± stddev.

Model	MAPE ↓	RMSE ↓	NLL ↓
LSTM	0.456 ± 0.033	0.214 ± 0.016	-
NODE	0.411 ± 0.030	0.193 ± 0.014	0.555 ± 0.041
NSDE	0.408 ± 0.029	0.192 ± 0.014	0.548 ± 0.040
LC-GODE	$\underline{0.388 \pm 0.028}$	$\underline{0.182 \pm 0.013}$	$\underline{0.498 \pm 0.036}$
LC-PFN	0.395 ± 0.029	0.186 ± 0.014	0.505 ± 0.037
DA-LCE (Ours)	$\mathbf{0.316 \pm 0.023}$	$\mathbf{0.148 \pm 0.011}$	$\mathbf{0.416 \pm 0.031}$

Table 6: Pearson correlation between early-stage (10%) and full-trajectory proxy components.

Proxy Component	Pearson Correlation Coefficient
Progress (ϕ_{prog})	0.89
Non-Linearity (ϕ_{nonlin})	0.82
Volatility (ϕ_{vol})	0.68

Table 7: Ablation study on the components of the difficulty proxy Φ. Performance is measured by average MAPE across all benchmarks.

Model Variant	Average MAPE ↓
DA-LCE (Full Proxy)	**0.216**
DA-LCE (w/o ϕ_{prog})	0.245
DA-LCE (w/o ϕ_{nonlin})	0.228
DA-LCE (w/o ϕ_{vol})	0.239
DA-LCE (No Proxy)	0.251

Table 8: Average (standardized) values of proxy components across task difficulty strata.

Stratum	Avg. ϕ_{prog}	Avg. ϕ_{nonlin}	Avg. ϕ_{vol}
Easy	0.89	-0.65	-0.72
Medium	0.12	0.21	0.18
Hard	-0.95	0.88	0.91

References

Adriaensen, S.; Rakotoarison, H.; Müller, S.; and Hutter, F. 2023. Efficient Bayesian Learning Curve Extrapolation using Prior-Data Fitted Networks. *Advances in Neural Information Processing Systems*, 36.

Baker, B.; Gupta, O.; Raskar, R.; and Naik, N. 2017. Accelerating Neural Architecture Search using Performance Prediction. In *International Conference on Learning Representations*.

Bansal, A.; Stoll, D.; Janowski, M.; Zela, A.; and Hutter, F. 2022. JAHS-Bench-201: A Foundation for Research on Joint Architecture and Hyperparameter Search. *arXiv preprint arXiv:2208.09958*.

Ding, Y.; Huang, Z.; Shou, X.; Guo, Y.; Sun, Y.; and Gao, J. 2025. Architecture-Aware Learning Curve Extrapolation via Graph Ordinary Differential Equation. In *The Thirty-Ninth AAAI Conference on Artificial Intelligence*.

Domhan, T.; Springenberg, J. T.; and Hutter, F. 2015. Speeding up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Dong, X.; and Yang, Y. 2020. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. In *International Conference on Learning Representations*.

Eggensperger, K.; Müller, P.; Mallik, N.; Feurer, M.; Sass, R.; Klein, A.; Janzing, D.; and Hutter, F. 2021. HPOBench: A Large-Scale Reproducible Benchmark for Black-Box HPO Based on OpenML. In *NeurIPS Datasets and Benchmarks Track*.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.

Li, X.; Wong, T.-K. L.; Chen, R. T. Q.; and Duvenaud, D. 2020. Scalable Gradients for Stochastic Differential Equations. *Artificial Intelligence and Statistics*, 3109–3119.

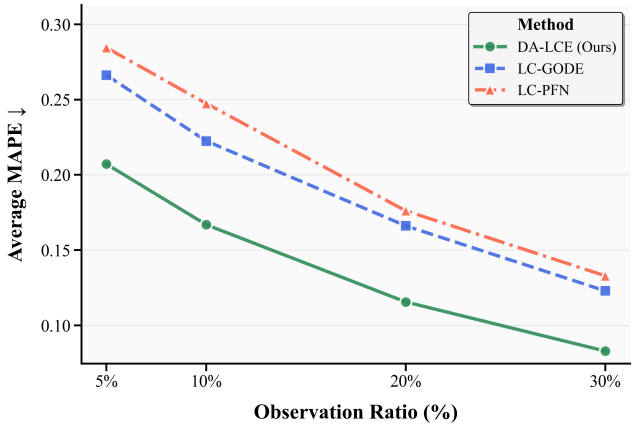

Figure 2: Model performance (MAPE) as a function of the observation window length. DA-LCE consistently outperforms baselines across all observation ratios.

Metz, L.; Maheswaranathan, N.; Sun, R.-Y.; Freeman, C. D.; Poole, B.; and Sohl-Dickstein, J. 2020. Using a Thousand Optimization Tasks to Learn Hyperparameter Search Strategies. *arXiv preprint arXiv:2002.11887*.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; and Others. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830.

Perez, E.; Strub, F.; De Vries, H.; Dumoulin, V.; and Courville, A. 2018. FiLM: Visual Reasoning with a General Conditioning Layer. In *The Thirty-Second AAAI Conference on Artificial Intelligence*.

Wang, Z.; Dahl, G. E.; Swersky, K.; Lee, C.; Nado, Z.; Gilmer, J.; Snoek, J.; and Ghahramani, Z. 2024. Pre-trained Gaussian Processes for Bayesian Optimization. *Journal of Machine Learning Research*, 25(212): 1–83.

Ying, Z.; You, J.; Morris, C.; Ren, X.; Hamilton, W.; and Leskovec, J. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. *Advances in Neural Information Processing Systems*, 31.

Zimmer, L.; Lindauer, M.; and Hutter, F. 2021. Auto-PyTorch-Tabular: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9): 3079–3090.