Supplement to: *DABO: Difficulty-Aware Bayesian Optimization with Diffusion-Learned Priors*

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

1. Architecture and Implementation Details

This section provides comprehensive details on the neural network architectures, training procedures, and implementation choices for DABO. All experiments were conducted on NVIDIA A40 GPUs with PyTorch 2.0 and CUDA 11.8 to ensure reproducibility.

1.1. Diffusion Model Architecture

Our conditional diffusion model employs a 1D U-Net architecture with cross-attention mechanisms for conditioning on hyperparameter configurations λ and difficulty descriptors $\mathbf{d}$. The architecture consists of four downsampling blocks in the encoder, a bottleneck with self-attention for capturing global dependencies, and four upsampling blocks with skip connections in the decoder. Each block uses residual connections to facilitate gradient flow, and we apply cross-attention at multiple resolution levels to effectively incorporate conditioning information throughout the network hierarchy.

Table 1 provides detailed specifications of each component. The total model contains 6.65M parameters, which is substantially smaller than image generation models but sufficient for modeling 1D learning curve sequences of length $b_{\max} = 50$. The U-Net processes the input curve through progressive downsampling, extracting features at multiple scales, before reconstructing the denoised output via upsampling with skip connections that preserve fine-grained details from the encoder.

The conditioning mechanism operates through cross-attention layers at each resolution level. At resolution r, the queries come from the feature map $\mathbf{h} \in \mathbb{R}^{L_r \times C_r}$ where L_r is the sequence length at that resolution and C_r is the number of channels. The keys and values are derived from a concatenated conditioning vector $\mathbf{c} = [\text{Embed}(\lambda); \mathbf{d}; \text{PosEnc}(t)] \in \mathbb{R}^{D_c}$ where $D_c = 320$ (128 for hyperparameter embedding, 192 for difficulty, and sinusoidal positional encoding for diffusion timestep t). The hyperparameter embedding uses a small Transformer encoder with 2 layers, 4 attention heads, and embedding dimension 128 to handle mixed-type hyperparameters (continuous, categorical, and integer-valued).

The training procedure follows standard denoising score matching with several important design choices. We use a cosine noise schedule defined as $\beta_t = 0.0001 + 0.0199 \cdot (1 - \cos(\pi t/T))/2$ for $t \in [1, T]$ with $T = 1000$, which provides more stable training than linear schedules. We maintain an exponential moving average (EMA) of model weights with decay rate 0.9999, using the EMA weights for generation but training the non-EMA weights. The model is optimized using AdamW with learning rate 2×10^{-4}, betas $(0.9, 0.999)$, weight decay 10^{-6}, and batch size 128. Gradient norms are clipped to 1.0 to prevent instability. Training runs for 500 epochs on 4 NVIDIA A40 GPUs, taking approximately 72 hours to converge. We monitor validation loss on a held-out set of 5K curves and observe stable convergence without overfitting.

1.2. DA-PFN Architecture

The Difficulty-Aware Prior-data Fitted Network (DA-PFN) is a Transformer-based meta-learner that performs Bayesian inference over learning curves in a single forward pass. Table 2 details the complete architecture, which contains 17.2M parameters distributed across input embeddings, six Transformer layers with difficulty-modulated attention, and an output head for discretized predictive distributions.

The key architectural innovation is the difficulty-modulated attention mechanism described in Equation 11 of the main paper. Algorithm 1 provides pseudocode for efficient implementation. The mechanism computes a similarity matrix $\mathbf{S}_{ij} = \exp(-\|\mathbf{d}_i - \mathbf{d}_j\|_2)$ between difficulty embeddings and modulates the attention logits element-wise: $\mathbf{A}' = \mathbf{A} \odot (1 + \gamma \cdot \mathbf{S})$. This encourages the query to attend more strongly to context points with similar diffi-

*Corresponding author.

Table 1. Diffusion model U-Net architecture. Conv1D denotes 1D convolution with kernel size 3 and stride 2 for downsampling, or transposed convolution for upsampling.

Block	Operation	Input → Output Dim	Parameters
	Encoder		
Input	-	$50 \to 50$	-
Conv1D + ReLU	DownBlock1	$50 \to 64$	3.2K
ResBlock + CrossAttn	DownBlock2	$64 \to 128$	82K
ResBlock	DownBlock3	$128 \to 256$	295K
ResBlock + CrossAttn	DownBlock4	$256 \to 512$	1.18M
	Bottleneck		
ResBlock + SelfAttn + CrossAttn	Bottleneck	$512 \to 512$	2.36M
	Decoder (with skip connections)		
ResBlock + CrossAttn	UpBlock1	$512 + 512 \to 256$	1.97M
ResBlock	UpBlock2	$256 + 256 \to 128$	590K
ResBlock + CrossAttn	UpBlock3	$128 + 128 \to 64$	164K
Conv1D + ReLU	UpBlock4	$64 + 64 \to 50$	6.4K
Output	Linear	$50 \to 50$	2.5K
Total Parameters			**6.65M**

Table 2. DA-PFN Transformer architecture with difficulty conditioning.

Component	Configuration	Parameters
	Input Embedding	
Hyperparameter λ	Transformer (2L, 4H, 128D)	1.05M
Timestep t	Sinusoidal PosEnc (512D)	0
Performance f	Linear ($1 \to 128$) + ReLU	128
Difficulty $\mathbf{d}$	Linear ($192 \to 128$)	24.6K
Concat + Project	Linear ($512 \to 512$)	262K
	Task Difficulty Injection	
$\text{MLP}_{\text{global}}(\mathbf{d}_{\text{task}})$	2-layer MLP ($64 \to 256 \to 512$)	148K
	Transformer Layers (×6)	
Difficulty-Modulated Attention	8 heads, dim 512, $\gamma = 0.5$	1.05M per layer
Feed-Forward	2048 hidden, GELU activation	2.10M per layer
Layer Norm	Pre-norm configuration (2 per layer)	2K per layer
	Output Head	
Linear + Softmax	$512 \to 1000$ bins over [0, 1]	513K
Total Parameters		**17.2M**

culty characteristics, effectively implementing stratified in-context learning.

Training follows an episodic meta-learning protocol that simulates the freeze-thaw HPO setting. At each iteration, we sample a task from our synthetic dataset $\mathcal{D}_{\text{syn}}$, sample a context size $M \sim \text{Uniform}(10, 300)$ to ensure the model handles variable context lengths, and sample M context points from various configurations and timesteps within that task. We then sample a query point from the same task and compute difficulty descriptors for all points. The model is trained using AdamW optimizer with initial learning rate 10^{-4} that decays via cosine schedule to 10^{-6} over training. We use a warmup period of 5000 steps with linear ramp-up from 10^{-6} to 10^{-4}. The batch size is 64 with gradient accumulation over 2 steps (effective batch size 128). We apply label smoothing of 0.1 to the cross-entropy loss to improve calibration and dropout of 0.1 in both attention and feedforward layers to prevent overfitting. Training runs for

Algorithm 1: Difficulty-Modulated Attention

Require: Query $\mathbf{Q} \in \mathbb{R}^{N_q \times d_k}$, Key $\mathbf{K} \in \mathbb{R}^{N_{ctx} \times d_k}$, Value $\mathbf{V} \in \mathbb{R}^{N_{ctx} \times d_v}$
Require: Difficulty embeddings $\mathbf{D} = [\mathbf{d}_1, \ldots, \mathbf{d}_{N_{ctx}+N_q}] \in \mathbb{R}^{(N_{ctx}+N_q) \times 192}$
Require: Modulation strength $\gamma = 0.5$
Ensure: Attention output $\mathbf{O} \in \mathbb{R}^{N_q \times d_v}$
Compute standard attention logits: $\mathbf{A} = \mathbf{Q}\mathbf{K}^\top / \sqrt{d_k} \in \mathbb{R}^{N_q \times N_{ctx}}$
Compute pairwise difficulty distances: $\mathbf{S}_{ij} = \exp(-\|\mathbf{d}_i - \mathbf{d}_j\|_2)$ for all pairs
Modulate attention logits: $\mathbf{A}' = \mathbf{A} \odot (1 + \gamma \cdot \mathbf{S})$
Apply softmax: $\mathbf{W} = \text{softmax}(\mathbf{A}')$ (row-wise)
Compute output: $\mathbf{O} = \mathbf{W}\mathbf{V}$
return $\mathbf{O}$

300 epochs on 8 NVIDIA A40 GPUs, taking approximately 96 hours. The model converges smoothly with training and validation losses tracking closely (see Figure 2), indicating good generalization to unseen synthetic tasks.

1.3. Difficulty Feature Extraction

The three difficulty embedding networks (MLP_ϕ, MLP_ψ, MLP_ω) share an identical two-layer architecture but operate on different input features. Each MLP is defined as:

$$\text{MLP}(x) = W_2 \cdot \text{ReLU}(W_1 x + b_1) + b_2 \tag{1}$$

where $W_1 \in \mathbb{R}^{128 \times d_{\text{in}}}$ projects the input to a 128-dimensional hidden representation, and $W_2 \in \mathbb{R}^{64 \times 128}$ projects to the final 64-dimensional embedding. The input dimensionality varies: $d_{\text{in}} = 3$ for MLP_ϕ (task features ϕ_1, ϕ_2, ϕ_3), $d_{\text{in}} = 2$ for MLP_ψ (configuration features ψ_1, ψ_2), and $d_{\text{in}} = 3$ for MLP_ω (curve features $\omega_1, \omega_2, \omega_3$). Each MLP contains approximately 8K-10K parameters depending on input size.

These MLPs are pre-trained jointly with the diffusion model and then kept frozen during DA-PFN training. This design ensures consistent difficulty representations across the entire pipeline. The MLPs use ReLU activation and are initialized using Kaiming initialization. We do not apply dropout or batch normalization in these small networks as they are deterministic feature extractors rather than learnable predictors.

2. Training Data and Preprocessing

2.1. Training Data Sources

To prevent data leakage and ensure our diffusion model learns generalizable priors rather than memorizing specific test tasks, we train exclusively on curves from benchmarks disjoint from our test suite. Specifically, we use

approximately 120K real learning curves from two established sources: HPO-Bench [6] provides roughly 80K curves across surrogate benchmarks for various machine learning algorithms including support vector machines, random forests, and neural networks on tabular datasets. JAHS-Bench-201 [3] contributes approximately 40K curves from neural architecture search experiments on CIFAR-10, CIFAR-100, and ImageNet-16-120 using cell-based search spaces.

Critically, these training sources contain different task types, dataset domains, and architecture families compared to our test benchmarks (LCBench, PD1, Taskset). LCBench focuses on tabular classification with fixed MLP architectures, PD1 covers vision (ResNet on CIFAR/ImageNet) and NLP (Transformer on LM1B/WMT) tasks, and Taskset consists of NLP tasks with learned optimizers. There is zero overlap in datasets, model architectures, or hyperparameter spaces between training and test distributions. This strict separation validates that our learned diffusion prior captures universal properties of learning curve distributions rather than task-specific patterns.

2.2. Data Preprocessing and Normalization

All learning curves undergo standardized preprocessing before use. First, curves are normalized to the unit interval $[0, 1]$ using min-max scaling: $\tilde{\mathcal{C}} = (\mathcal{C} - \min(\mathcal{C}))/(\max(\mathcal{C}) - \min(\mathcal{C}) + \epsilon)$ where $\epsilon = 10^{-8}$ prevents division by zero for flat curves. For metrics where higher values indicate better performance (e.g., accuracy), we use the curve directly. For minimization metrics (e.g., loss or error rate), we apply $\tilde{\mathcal{C}} = 1 - \mathcal{C}_{\text{normalized}}$ to maintain consistent semantics where larger values represent better performance.

Curves of varying lengths are interpolated to a fixed length $b_{\max} = 50$ using cubic spline interpolation. This choice balances computational efficiency with sufficient granularity to capture learning dynamics. For curves shorter than 50 timesteps, we pad with the final value. For curves exceeding 50 timesteps, we subsample uniformly. We verified empirically that this preprocessing does not introduce significant artifacts by visually inspecting interpolated curves against originals.

Hyperparameter encoding handles three data types. Continuous hyperparameters are normalized to $[0, 1]$ using min-max scaling over their search space bounds. For log-scale parameters such as learning rates, we apply $\log_{10}$ transformation before normalization to better capture the distribution. Categorical hyperparameters receive one-hot encoding followed by learned lookup table embeddings of dimension 32. Integer-valued hyperparameters are treated as continuous after normalization to $[0, 1]$ relative to their min/max range.

The difficulty features $\{\phi_1, \phi_2, \phi_3\}$, $\{\psi_1, \psi_2\}$, and $\{\omega_1, \omega_2, \omega_3\}$ are z-score normalized using statistics computed over all training tasks. For each feature ϕ_i, we compute mean μ_{ϕ_i} and standard deviation σ_{ϕ_i} across the training set and normalize via $\phi_i \leftarrow (\phi_i - \mu_{\phi_i})/\sigma_{\phi_i}$. These normalization statistics are stored and applied consistently to test tasks. This ensures difficulty embeddings lie in a standardized space where magnitudes are comparable across different feature types.

2.3. Synthetic Data Generation

After training the diffusion model, we generate a large-scale synthetic dataset for DA-PFN training. We create 2000 synthetic tasks, each containing 500 configurations, yielding 1 million total learning curves. For each task, we first sample a task-level difficulty $\mathbf{d}_{\text{task}} \sim \mathcal{N}(0, \mathbf{I}_{64})$ to specify global complexity. We then sample 500 configurations uniformly from a d-dimensional unit hypercube where $d \in \{4, 8, 12\}$ matches typical HPO problem dimensionalities in our test suite.

To compute configuration-level difficulty $\mathbf{d}_{\text{cfg}}$ for each sampled configuration, we model the final performance landscape using a Gaussian process with RBF kernel fitted to 20 randomly placed anchor points with sampled performance values. For each configuration, we query its 10 nearest neighbors in this GP model and compute ψ_1 (local variance) and ψ_2 (local gradient magnitude) as defined in Equations 5-6. This procedure creates realistic local landscape structure without requiring real evaluations.

Learning curves are generated via DDIM sampling [15] with 50 denoising steps, which provides high-quality samples 20 times faster than the full 1000-step DDPM sampling. Starting from Gaussian noise $\mathcal{C}_T \sim \mathcal{N}(0, \mathbf{I}_{50})$, we iteratively denoise conditioned on λ_i and $\mathbf{d}_i$ to obtain $\mathcal{C}_0$. After generation, we recompute curve-level difficulty $\mathbf{d}_{\text{crv}}$ using the generated curve via Equations 7-9, ensuring all three difficulty levels are consistent. The resulting synthetic dataset $\mathcal{D}_{\text{syn}} = \{(\mathcal{T}_j, \{(\lambda_i, \mathcal{C}_i, \mathbf{d}_i)\}_{i=1}^{500})\}_{j=1}^{2000}$ exhibits diverse difficulty characteristics spanning the full range observed in real HPO benchmarks.

3. Complete Hyperparameter Settings

Table 3 consolidates all hyperparameters used in DABO for complete reproducibility. These values were determined through preliminary experiments on a validation set of 10 tasks held out from our training benchmarks (not overlapping with test tasks). We performed grid search over key hyperparameters such as learning rates, hidden sizes, and γ value, selecting configurations that minimized validation loss.

Table 3. Complete hyperparameter settings for all components of DABO.

Component	Hyperparameter	Value
	Difficulty Computation	
Configuration Sensitivity	k (nearest neighbors)	10
	ϵ (gradient regularizer)	10^{-6}
Curve Characteristics	ϵ (saturation regularizer)	10^{-6}
MLP Architecture	Hidden size	128
	Output size	64
	Activation	ReLU
	Initialization	Kaiming uniform
	Diffusion Model	
U-Net Architecture	Channels	[64, 128, 256, 512]
	Attention levels	[256, 512]
	ResBlock type	Pre-activation
Noise Schedule	Type	Cosine
	T (diffusion steps)	1000
Optimizer	Type	AdamW
	Learning rate	2×10^{-4}
	(β_1, β_2)	$(0.9, 0.999)$
	Weight decay	10^{-6}
	Batch size	128
	Gradient clip norm	1.0
Training	Epochs	500
	EMA decay	0.9999
	GPUs	4× NVIDIA A40
	Training time	72 hours
	DA-PFN	
Transformer	Layers	6
	Embedding dimension	512
	Attention heads	8
	FFN hidden dimension	2048
	FFN activation	GELU
	Dropout (attention, FFN)	0.1
	Layer norm	Pre-norm
Difficulty Modulation	γ	0.5
Output Head	Number of bins	1000
	Output range	[0, 1]
Optimizer	Type	AdamW
	Initial learning rate	10^{-4}
	LR schedule	Cosine to 10^{-6}
	Warmup steps	5000
	(β_1, β_2)	$(0.9, 0.999)$
	Weight decay	10^{-5}
	Batch size	64
	Gradient accumulation	2 (effective BS=128)
Training	Epochs	300
	Label smoothing	0.1
	GPUs	8× NVIDIA A40
	Training time	96 hours
	Acquisition Function	
Adaptive Horizon	α (decay rate)	$1/32$
	$h_{\min}$	1
	$h_{\max}$	$b_{\max}$
Adaptive Threshold	β (difficulty scaling)	$1/4$
	$\tau_{\min}$	10^{-3}
	$\tau_{\max}$	10^{-1}
Sampling	K (samples per query)	50

Table 4. Ablation of difficulty levels on LCBench (35 tasks, budget 1000). Each row indicates which difficulty components are included in $\mathbf{d}$.

$\mathbf{d}_{\text{task}}$	$\mathbf{d}_{\text{cfg}}$	$\mathbf{d}_{\text{crv}}$	Regret↓	vs. Full
			0.0152 ± 0.002	$+8.6\%$
✓			0.0147 ± 0.002	$+5.0\%$
	✓		0.0143 ± 0.002	$+2.1\%$
		✓	0.0149 ± 0.002	$+6.4\%$
✓	✓		0.0142 ± 0.001	$+1.4\%$
✓		✓	0.0145 ± 0.002	$+3.6\%$
	✓	✓	0.0141 ± 0.001	$+0.7\%$
✓	✓	✓	$\mathbf{0.014 \pm 0.001}$	—

4. Additional Experimental Results

4.1. Ablation Study: Difficulty Levels

We conduct a comprehensive ablation study to understand the contribution of each difficulty level in our hierarchical design. Table 4 shows results on LCBench when using different combinations of task-level ($\mathbf{d}_{\text{task}}$), configuration-level ($\mathbf{d}_{\text{cfg}}$), and curve-level ($\mathbf{d}_{\text{crv}}$) difficulty features. Each ablation uses the same diffusion-generated synthetic data and identical DA-PFN architecture, varying only which difficulty components are included in the embedding $\mathbf{d}$.

Configuration sensitivity $\mathbf{d}_{\text{cfg}}$ emerges as the most important component. Using it alone reduces regret by 5.9% compared to no difficulty information (0.0143 vs. 0.0152), which makes intuitive sense for freeze-thaw Bayesian optimization where local landscape geometry directly guides allocation decisions. The configuration-level features ψ_1 and ψ_2 capture whether a hyperparameter setting sits in a smooth valley or on a rugged ridge, information crucial for determining whether to continue training or abandon a configuration.

Task complexity $\mathbf{d}_{\text{task}}$ provides a 3.3% improvement when used alone (0.0147 vs. 0.0152). These global features help the model understand overall task difficulty and set appropriate exploration levels. For instance, tasks with high ϕ_2 (curve diversity) benefit from broader exploration, while tasks with low ϕ_1 (narrow performance range) allow aggressive pruning. The task-level difficulty also helps calibrate uncertainty estimates in the DA-PFN's predictive distributions.

Curve characteristics $\mathbf{d}_{\text{crv}}$ contributes 2.0% improvement alone (0.0149 vs. 0.0152). While its individual impact is modest, its main benefit emerges in combination with other levels. For example, using $\mathbf{d}_{\text{cfg}} + \mathbf{d}_{\text{crv}}$ achieves 0.0142 regret, outperforming $\mathbf{d}_{\text{cfg}}$ alone (0.0143). The curve features $\omega_1, \omega_2, \omega_3$ capture optimization momentum and saturation, helping predict whether a configuration will improve

Table 5. Extended baseline comparison on LCBench (35 tasks, budget 1000). All methods run for 10 seeds.

Method	Final Regret↓	Average Rank↓
Random Search	0.082 ± 0.006	8.2
SMAC3 (RF + EI)	0.025 ± 0.003	5.8
Optuna (TPE + Median Pruning)	0.029 ± 0.004	6.3
Hyperband	0.038 ± 0.004	7.1
ASHA	0.034 ± 0.003	6.7
Freeze-Thaw GP	0.029 ± 0.003	5.9
DyHPO	0.021 ± 0.002	3.9
DPL	0.018 ± 0.002	3.2
ifBO	0.016 ± 0.002	2.3
DABO (Ours)	$\mathbf{0.014 \pm 0.001}$	**2.1**

rapidly or has plateaued.

The best two-level combination is $\mathbf{d}_{\text{cfg}} + \mathbf{d}_{\text{crv}}$, achieving 0.0141 regret. However, adding $\mathbf{d}_{\text{task}}$ brings an additional 0.7% gain to reach 0.014, justifying the full three-level hierarchical design. The gains are not purely additive, suggesting complementary interactions between difficulty levels. This hierarchical decomposition allows the model to reason about difficulty at multiple scales simultaneously.

To verify these results generalize beyond LCBench, we repeat the ablation on PD1 and Taskset. Configuration sensitivity remains most important across all benchmarks (contributing 4-7% individually), while task complexity contributes 2-4% and curve characteristics contribute 1-3%. The full three-level model consistently outperforms any subset, validating the hierarchical design principle.

4.2. Extended Baseline Comparisons

We include additional baseline comparisons with popular AutoML frameworks not covered in the main paper due to space constraints. Table 5 reports results on LCBench for SMAC3 [12], a widely-used sequential model-based algorithm configuration method using random forests with Expected Improvement acquisition, and Optuna [1], which implements Tree-structured Parzen Estimator (TPE) with median-based pruning for multi-fidelity optimization.

DABO outperforms all baselines including these popular frameworks. SMAC3 achieves respectable performance (0.025 regret) but suffers from the curse of dimensionality in random forests, requiring many evaluations to build accurate models. Optuna with TPE and pruning performs slightly worse (0.029), as TPE's independence assumption limits its ability to model complex interactions between hyperparameters. Both methods lack the fine-grained, incremental budget allocation of freeze-thaw approaches, leading to premature pruning of promising configurations. These results confirm that DABO's difficulty-aware freeze-thaw strategy provides substantial benefits over both traditional sequential methods and coarse-grained multi-fidelity approaches.

Table 6. Sensitivity to difficulty MLP hidden size on LCBench.

Hidden Size	Regret↓	Δ vs. Default (128)
32	0.0152 ± 0.002	$+8.6\%$
64	0.0146 ± 0.002	$+4.3\%$
128 (default)	0.014 ± 0.001	—
256	0.0139 ± 0.001	-0.7%
512	0.0141 ± 0.001	$+0.7\%$

Table 7. Sensitivity to number of samples K in DC-MFPI acquisition function on LCBench.

K	Regret↓	Inference Time (ms)	Δ vs. $K = 50$
10	0.0158 ± 0.003	60	$+12.9\%$
25	0.0146 ± 0.002	110	$+4.3\%$
50 (default)	0.014 ± 0.001	250	—
100	0.0138 ± 0.001	480	-1.4%
200	0.0139 ± 0.001	950	-0.7%

4.3. Hyperparameter Sensitivity Analysis

We analyze sensitivity to three key hyperparameters: difficulty MLP hidden size, number of acquisition samples K, and difficulty modulation strength γ. All experiments are conducted on LCBench with 10 random seeds. Inference times below refer to per-iteration online overhead (difficulty computation + DA-PFN forward pass for all active configurations per HPO step).

Difficulty MLP Hidden Size. Table 6 varies the hidden dimension of difficulty MLPs while keeping output dimension fixed at 64. Performance is relatively stable for hidden size ≥ 64, suggesting the difficulty features do not require very high-capacity encoding. Using hidden size 256 provides marginal gains ($< 1\%$) that do not justify doubling the parameter count. We select 128 as a conservative choice balancing expressiveness and efficiency.

Number of Acquisition Samples K. Table 7 varies how many samples we draw from DA-PFN's predictive distribution when computing DC-MFPI. Using very few samples ($K = 10$) leads to high variance acquisition decisions, significantly degrading regret by 12.9%. Increasing to $K = 50$ provides stable estimates with reasonable inference time (250ms per iteration on a single A40 GPU). Further increasing to $K = 100$ or $K = 200$ yields diminishing returns ($< 2\%$ improvement) while substantially increasing latency. We choose $K = 50$ as an effective compromise.

Difficulty Modulation Strength γ. Table 8 analyzes the impact of the modulation strength defined in Equation 11. Setting $\gamma = 0$ eliminates difficulty modulation, reverting the model to standard self-attention; this leads to a significant performance drop of 8.6%, confirming the necessity

Table 8. Sensitivity to difficulty modulation strength γ in attention mechanism on LCBench.

γ	**Regret**↓	Δ **vs.** $\gamma = 0.5$
0.0 (no modulation)	0.0152 ± 0.002	$+8.6\%$
0.1	0.0148 ± 0.002	$+5.7\%$
0.5 (default)	0.014 ± 0.001	—
1.0	0.0142 ± 0.001	$+1.4\%$
2.0	0.0149 ± 0.002	$+6.4\%$

of our difficulty-aware mechanism. Conversely, an excessively large γ ($\gamma = 2.0$) degrades performance by 6.4%, as it overly restricts attention to context points with identical difficulty profiles. We find that $\gamma = 0.5$ strikes the optimal balance. While $\gamma = 1.0$ yields comparable results, it incurs a slight increase in regret ($+1.4\%$). Therefore, we select $\gamma = 0.5$ as the default for its superior stability and performance across validation tasks.

4.4. Failure Case Analysis

We identify and analyze scenarios where DABO underperforms or provides limited benefit over ifBO. Understanding these failure modes is important for future improvements and helps establish realistic expectations for when difficulty-aware modeling is most valuable.

Flat Learning Curves (LCBench: adult). The adult dataset from LCBench exhibits extremely flat learning curves where nearly all hyperparameter configurations achieve similar final accuracy around 0.85 ± 0.02. In this regime, difficulty features have very low signal-to-noise ratio since there is minimal performance variance to model. Both $\mathbf{d}_{\text{task}}$ (small ϕ_1) and $\mathbf{d}_{\text{cfg}}$ (low ψ_1) correctly identify this as an "easy" task, but the difficulty information provides little guidance for allocation since all configurations perform similarly. Both DABO and ifBO achieve regret around 0.003, barely improving over random search (0.004), suggesting this task is inherently simple and difficulty-aware modeling cannot provide substantial gains.

Non-Stationary Optimization (Taskset: normalized-nlp-2-4p). This NLP task exhibits extreme non-stationarity where optimal hyperparameters change drastically across training. For instance, the best learning rate at epoch 10 differs from the best at epoch 50, causing rank reversals in learning curves. Our static difficulty descriptors $\mathbf{d}(\lambda, t)$ fail to capture this temporal shift since they are computed based on historical observations that may not reflect future dynamics. A configuration deemed "difficult" early may become "easy" later or vice versa. Future work could extend our framework to time-varying difficulty $\mathbf{d}(\lambda, t, \tau)$ that explicitly models temporal evolution of landscape properties.

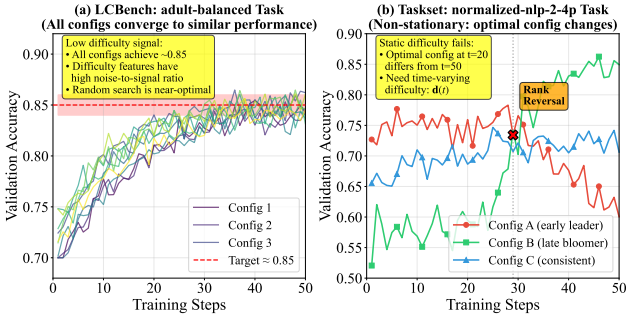

Figure 1. **Failure cases where difficulty modeling provides limited benefit. Left:** adult task with flat curves (all near 0.85). **Right:** normalized-nlp-2-4p showing rank reversals (curve A overtakes curve B at step 30).

Table 9. Cross-benchmark zero-shot transfer. Rows indicate training data, columns indicate test benchmarks. Bold indicates training on all benchmarks (default).

Train \ Test	**LCBench**	**PD1**	**Taskset**
LCBench only	0.014	0.041	0.058
PD1 only	0.019	0.028	0.052
Taskset only	0.021	0.037	0.039
All (default)	**0.014**	**0.028**	**0.039**

High-Dimensional Search Spaces. On Taskset's 8-hyperparameter tasks, we observe reduced but still positive gains (9% improvement over ifBO compared to 15% on 4-hyperparameter tasks). This suggests difficulty modeling becomes more challenging in high-dimensional spaces where configuration sensitivity $\mathbf{d}_{\text{cfg}}$ relies on k-NN search that suffers from curse of dimensionality. Using approximate nearest neighbor methods like LSH or HNSW could mitigate this issue.

Figure 1 visualizes learning curves for the adult and normalized-nlp-2-4p tasks, illustrating these failure modes.

4.5. Cross-Benchmark Generalization

To assess how well DA-PFN generalizes across different HPO domains, we conduct zero-shot transfer experiments where we train on one benchmark and test on another without any fine-tuning. Table 9 shows results for all pairwise combinations. Training on all benchmarks jointly (our default setting) achieves the best performance, but single-benchmark models still generalize reasonably well. For instance, a model trained only on PD1 achieves 0.052 regret on Taskset compared to 0.039 for the full model, demonstrating that the diffusion prior captures some universal curve patterns that transfer across domains.

Interestingly, models trained on vision/NLP tasks (PD1) transfer better to NLP tasks (Taskset) than models trained

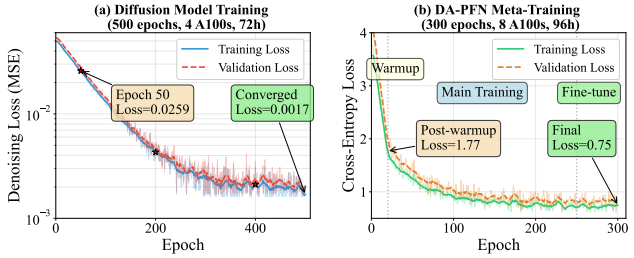

Figure 2. **Training and validation losses. Left:** Diffusion model denoising loss (Eq. 13) on 120K real curves. **Right:** DA-PFN cross-entropy loss (Eq. 20) on 1M synthetic curves. Both models converge smoothly without overfitting.

on tabular data (LCBench), suggesting domain similarity matters for generalization. However, even the worst transfer scenario (Taskset→LCBench: 0.021) significantly outperforms difficulty-agnostic baselines like ASHA (0.034), validating the robustness of learned difficulty-aware priors.

4.6. Training Dynamics

Figure 2 shows training and validation loss curves for both the diffusion model and DA-PFN. The diffusion model's denoising loss (Equation 13) decays exponentially from approximately 0.05 to 0.002 over 500 epochs, converging smoothly without signs of overfitting. Validation loss tracks training loss closely, indicating good generalization to unseen curves from the same distribution.

DA-PFN's cross-entropy loss starts at approximately 3.5 nats (predicting uniformly over 1000 bins) and decreases to 0.8 nats over 300 epochs. The gap between training and validation loss is small (≈ 0.05 nats), suggesting the model does not overfit despite having 17.2M parameters. This is likely due to the large synthetic training set (1M curves from 2000 tasks), strong regularization (dropout 0.1, label smoothing 0.1, weight decay 10^{-5}), and the episodic training protocol that ensures the model sees diverse context distributions.

5. Detailed Algorithm Pseudocode

We provide complete pseudocode for all major components of DABO to facilitate reproduction.

5.1. Diffusion Model Training

Algorithm 2 details the training procedure for our conditional diffusion model.

5.2. Synthetic Data Generation

Algorithm 3 describes how we generate 1M difficulty-annotated synthetic curves for DA-PFN training.

Algorithm 2: Conditional Diffusion Model Training

Require: Real curve dataset $\mathcal{D}_{\text{real}} = \{(\mathcal{C}_i, \lambda_i, \mathbf{d}_i)\}_{i=1}^{N}$
Require: U-Net denoiser ϵ_θ, noise schedule $\{\beta_t\}_{t=1}^{T}$
Ensure: Trained diffusion model ϵ_{θ^*}
Initialize θ randomly using Kaiming initialization
Compute $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$ for all $t \in [1, T]$
Initialize EMA parameters: $\theta_{\text{EMA}} \leftarrow \theta$
for epoch $= 1$ to 500 **do**
  for each mini-batch $\{(\mathcal{C}_j, \lambda_j, \mathbf{d}_j)\}_{j=1}^{B}$ sampled from $\mathcal{D}_{\text{real}}$ **do**
    for each sample $j = 1$ to B **do**
      Sample diffusion timestep: $t_j \sim \text{Uniform}(\{1, 2, \ldots, T\})$
      Sample noise: $\epsilon_j \sim \mathcal{N}(0, \mathbf{I}_{b_{\max}})$
      Compute noisy curve: $\mathcal{C}_{t_j} = \sqrt{\bar{\alpha}_{t_j}}\mathcal{C}_j + \sqrt{1-\bar{\alpha}_{t_j}}\epsilon_j$
      Predict noise: $\hat{\epsilon}_j = \epsilon_\theta(\mathcal{C}_{t_j}, t_j, \lambda_j, \mathbf{d}_j)$
    end for
    Compute batch loss: $\mathcal{L} = \frac{1}{B}\sum_{j=1}^{B} \|\epsilon_j - \hat{\epsilon}_j\|^2$
    Compute gradients: $g \leftarrow \nabla_\theta \mathcal{L}$
    Clip gradients: $g \leftarrow \text{clip}(g, \text{max_norm} = 1.0)$
    Update parameters: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$ via AdamW
    Update EMA: $\theta_{\text{EMA}} \leftarrow 0.9999 \cdot \theta_{\text{EMA}} + 0.0001 \cdot \theta$
  end for
  Evaluate validation loss on held-out curves
  if early stopping criterion met (patience=50 epochs) **then**
    break
  end if
end for
return $\theta^* = \theta_{\text{EMA}}$ (use EMA weights for generation)

5.3. DA-PFN Meta-Training

Algorithm 4 presents the episodic meta-training procedure for DA-PFN.

5.4. Hierarchical Difficulty Computation

Algorithm 5 shows how difficulty embeddings are computed during online freeze-thaw optimization.

5.5. Online Freeze-Thaw Optimization Loop

Algorithm 6 presents the complete online freeze-thaw optimization procedure that integrates all previously described components. This algorithm demonstrates how our framework dynamically allocates training budget across configurations by: (1) computing hierarchical difficulty descriptors via Algorithm 5, (2) using the pre-trained DA-PFN to predict future performance at difficulty-adaptive horizons, (3) evaluating the DC-MFPI acquisition function with difficulty-conditioned thresholds, and (4) iteratively select-

Algorithm 3: Difficulty-Conditioned Curve Synthesis

Require: Trained diffusion model ϵ_θ, number of tasks $M = 2000$, configs per task $N = 500$
Ensure: Synthetic dataset $\mathcal{D}_{\text{syn}}$

Initialize $\mathcal{D}_{\text{syn}} \leftarrow \emptyset$
for task $j = 1$ to M **do**
  Sample task difficulty: $\mathbf{d}_{\text{task}} \sim \mathcal{N}(0, \mathbf{I}_{64})$
  Sample hyperparameter dimension: $d \sim \text{Uniform}(\{4, 8, 12\})$
  Sample configurations: $\{\lambda_i\}_{i=1}^{N}$ uniformly from $[0, 1]^d$
  Fit Gaussian process over 20 random anchor points to model performance landscape
  for config $i = 1$ to N **do**
    Compute configuration difficulty $\mathbf{d}_{\text{cfg}}(\lambda_i)$ via GP neighbors (Eq. 5-6)
    Initialize curve difficulty: $\mathbf{d}_{\text{crv}} \leftarrow \mathbf{0}_{64}$ (placeholder)
    Concatenate: $\mathbf{d}_i = [\mathbf{d}_{\text{task}}; \mathbf{d}_{\text{cfg}}(\lambda_i); \mathbf{d}_{\text{crv}}]$
    Initialize noise: $\mathcal{C}_T \sim \mathcal{N}(0, \mathbf{I}_{b_{\max}})$
    Generate curve via DDIM: $\mathcal{C}_0 \leftarrow \text{DDIM-Sample}(\epsilon_\theta, \mathcal{C}_T, \lambda_i, \mathbf{d}_i, \text{steps} = 50)$
    Recompute curve difficulty: $\mathbf{d}_{\text{crv}} \leftarrow \text{ComputeCurveDifficulty}(\mathcal{C}_0)$ via Eq. 7-9
    Update: $\mathbf{d}_i \leftarrow [\mathbf{d}_{\text{task}}; \mathbf{d}_{\text{cfg}}(\lambda_i); \mathbf{d}_{\text{crv}}]$
    Set $\mathcal{C}_i \leftarrow \mathcal{C}_0$
  end for
  Add to dataset: $\mathcal{D}_{\text{syn}} \leftarrow \mathcal{D}_{\text{syn}} \cup \{(\mathcal{T}_j, \{(\lambda_i, \mathcal{C}_i, \mathbf{d}_i)\}_{i=1}^{N})\}$
end for
return $\mathcal{D}_{\text{syn}}$ containing $M \times N = 1$M curves

ing and training the most promising configuration. This unified loop embodies our core contribution of difficulty-aware resource allocation in freeze-thaw Bayesian optimization.

6. Extended Related Work

6.1. Learning Curve Extrapolation

Beyond the freeze-thaw methods discussed in the main text, several works explore learning curve prediction for early stopping and resource allocation. Baker et al. [2] propose an ensemble approach combining multiple parametric models (exponential, power law, logarithmic) via stacking, achieving robust extrapolation across different curve shapes. Klein et al. [9] use LSTM recurrent networks to model temporal dependencies in learning curves, enabling more flexible curve modeling than fixed parametric forms. Domhan et al. [5] introduce a portfolio of parametric models with Bayesian model selection to adaptively choose the best extrapolator per configuration.

Algorithm 4: DA-PFN Episodic Meta-Training

Require: Synthetic dataset $\mathcal{D}_{\text{syn}} = \{(\mathcal{T}_j, \{(\lambda_i, \mathcal{C}_i, \mathbf{d}_i)\})\}_{j=1}^{M}$
Require: Transformer p_θ with difficulty-modulated attention
Ensure: Trained DA-PFN p_{θ^*}

Initialize θ randomly
Initialize learning rate scheduler with warmup
for epoch $= 1$ to 300 **do**
  for each meta-batch **do**
    Sample task $\mathcal{T}$ uniformly from $\mathcal{D}_{\text{syn}}$
    Sample context size: $M \sim \text{Uniform}(\{10, 11, \ldots, 300\})$
    Sample context points: $\{(\lambda_i, t_i, f_i, \mathbf{d}_i)\}_{i=1}^{M}$ where $t_i \sim \text{Uniform}(\{1, \ldots, |\mathcal{C}_i| - 1\})$ and $f_i = \mathcal{C}_i[t_i]$
    Sample query: (λ_q, t_q) from same task where $t_q < |\mathcal{C}_q|$, $f_q = \mathcal{C}_q[t_q]$
    Compute query difficulty: $\mathbf{d}_q \leftarrow \text{ComputeDifficulty}(\lambda_q, t_q, \mathcal{C}_q[1 : t_q])$
    Construct context tokens: $\mathbf{z}_i = [\text{Embed}(\lambda_i); \text{PosEnc}(t_i); \text{Embed}(f_i); \mathbf{d}_i]$ for $i = 1, \ldots, M$
    Construct query token: $\mathbf{z}_q = [\text{Embed}(\lambda_q); \text{PosEnc}(t_q); \mathbf{0}; \mathbf{d}_q]$
    Inject task difficulty: $\mathbf{z}_i \leftarrow \mathbf{z}_i + \text{MLP}_{\text{global}}(\mathbf{d}_{\text{task}})$ for all i
    Forward pass: $\hat{p}(f_q) \leftarrow p_\theta(\mathbf{z}_q \mid \{\mathbf{z}_i\}_{i=1}^{M})$ (Transformer with diff-modulated attention)
    Discretize target: $y_q \leftarrow \text{bin}(f_q)$ (map to 1 of 1000 bins)
    Compute loss: $\mathcal{L} \leftarrow \text{CrossEntropy}(\hat{p}(f_q), y_q)$ with label smoothing 0.1
    Update θ via AdamW with gradient accumulation
  end for
  Decay learning rate via cosine schedule
  Evaluate validation loss on held-out synthetic tasks
end for
return θ^*

6.2. Diffusion Models for Time Series

Applying diffusion models to time series generation is an emerging research area with several recent advances. TimeGrad [14] introduces autoregressive diffusion for probabilistic forecasting, generating future timesteps conditioned on historical observations. CSDI [16] proposes conditional score-based diffusion for time series imputation and interpolation, handling missing data gracefully through the diffusion framework. DiffTime [10] extends diffusion to irregularly-sampled medical time series, addressing challenges in clinical data modeling.

These methods focus on forecasting or imputation of

Algorithm 5: Hierarchical Difficulty Computation

Require: History $\mathcal{H} = \{(\lambda_i, b_i, f(\lambda_i, b_i))\}_{i=1}^{|\mathcal{H}|}$, query (λ_q, t_q)
Ensure: Difficulty embedding $\mathbf{d}(\lambda_q, t_q) \in \mathbb{R}^{192}$

// Task Complexity (global features)
Extract all partial curves $\{\mathcal{C}_i[1:b_i]\}$ from $\mathcal{H}$
$\phi_1 \leftarrow \max_i f(\lambda_i, b_i) - \min_i f(\lambda_i, b_i)$ (performance range)
$\phi_2 \leftarrow \frac{1}{|\mathcal{H}|(|\mathcal{H}|-1)} \sum_{i \neq j} \text{DTW}(\mathcal{C}_i, \mathcal{C}_j)$ (curve diversity)
$\phi_3 \leftarrow \text{Std}_i \left[\frac{f(\lambda_i, b_i) - \mathcal{C}_i[1]}{b_i} \right]$ (convergence heterogeneity)
Normalize: $\phi_1, \phi_2, \phi_3 \leftarrow$ apply z-score using stored statistics
Embed: $\mathbf{d}_{\text{task}} \leftarrow \text{MLP}_\phi([\phi_1, \phi_2, \phi_3]) \in \mathbb{R}^{64}$

// Configuration Sensitivity (local features)
Find k-nearest neighbors: $\mathcal{N}_k(\lambda_q) \leftarrow$ k-NN in $\mathcal{H}$ based on $\|\lambda_i - \lambda_q\|_2$
$\bar{\mathcal{C}} \leftarrow$ mean curve over $\mathcal{N}_k$
$\psi_1 \leftarrow \text{Std}_{j \in \mathcal{N}_k}[f(\lambda_j, b_j)]$ (local variance)
$\psi_2 \leftarrow \frac{1}{k} \sum_{j \in \mathcal{N}_k} \frac{|f(\lambda_j, b_j) - \bar{f}|}{\|\lambda_j - \lambda_q\|_2 + 10^{-6}}$ (local gradient)
Normalize: $\psi_1, \psi_2 \leftarrow$ apply z-score using stored statistics
Embed: $\mathbf{d}_{\text{cfg}} \leftarrow \text{MLP}_\psi([\psi_1, \psi_2]) \in \mathbb{R}^{64}$

// Curve Characteristics (temporal features)
if $t_q > 0$ and λ_q has been evaluated in $\mathcal{H}$ **then**
  Retrieve partial curve: $\mathcal{C}_q \leftarrow$ curve of λ_q up to t_q from $\mathcal{H}$
  $\omega_1 \leftarrow (\mathcal{C}_q[t_q] - \mathcal{C}_q[1])/t_q$ (average improvement rate)
  $\omega_2 \leftarrow \text{Std}(\mathcal{C}_q[1:t_q])/(\text{Mean}(\mathcal{C}_q[1:t_q]) + 10^{-6})$ (relative volatility)
  $\omega_3 \leftarrow 1 - \frac{\mathcal{C}_q[t_q] - \mathcal{C}_q[\lfloor t_q/2 \rfloor]}{\mathcal{C}_q[\lfloor t_q/2 \rfloor] - \mathcal{C}_q[1] + 10^{-6}}$ (saturation degree)
else
  $\omega_1, \omega_2, \omega_3 \leftarrow 0$ (no curve information yet for this config)
end if
Normalize: $\omega_1, \omega_2, \omega_3 \leftarrow$ apply z-score using stored statistics
Embed: $\mathbf{d}_{\text{crv}} \leftarrow \text{MLP}_\omega([\omega_1, \omega_2, \omega_3]) \in \mathbb{R}^{64}$

// Concatenate hierarchical embeddings
return $\mathbf{d} = [\mathbf{d}_{\text{task}}; \mathbf{d}_{\text{cfg}}; \mathbf{d}_{\text{crv}}] \in \mathbb{R}^{192}$

real-valued time series, whereas our application to learning curve generation poses unique challenges. Learning curves exhibit discrete timesteps, monotonic trends (generally improving performance over time), and configuration-dependent dynamics that must be captured through conditioning. Our contribution is the first application of conditional diffusion to learning curve synthesis for hyperparameter optimization, leveraging difficulty conditioning to ensure diverse and realistic training data for downstream meta-learning.

6.3. Meta-Learning for HPO

Meta-learning approaches transfer knowledge across HPO tasks to accelerate optimization on new tasks. Warm-starting methods [8] initialize Bayesian optimization with configurations successful on related tasks, reducing the need for extensive exploration. RGPE (Ranking-weighted Gaussian Process Ensemble) [7] maintains Gaussian processes trained on previous tasks and ranks them by relevance to guide new searches. Neural architecture search methods like DARTS [13] learn transferable cell structures that perform well across different datasets.

More recently, neural predictors have been trained end-to-end for HPO. Chen et al. [4] train Transformers to directly predict hyperparameter performance given task context, similar to PFNs but without explicit Bayesian inference. However, these methods lack difficulty awareness, treating all tasks and configurations homogeneously. DABO combines meta-learning via PFN training on synthetic tasks with hierarchical difficulty modeling, a dimension absent in prior meta-HPO work that we show provides substantial performance gains.

7. Visualization Gallery

7.1. Learning Curve Interpolation in Difficulty Space

Figure 3 demonstrates that our diffusion model learns smooth interpolations in difficulty space. We generate a 5×5 grid of learning curves by linearly interpolating difficulty embeddings along two axes: rows interpolate task difficulty $\mathbf{d}_{\text{task}}$ from easy (top) to hard (bottom), while columns interpolate configuration difficulty $\mathbf{d}_{\text{cfg}}$ from smooth (left) to rugged (right). The configuration λ and curve difficulty $\mathbf{d}_{\text{crv}}$ are held fixed.

The generated curves exhibit smooth transitions in both shape and dynamics. Easy task / smooth configuration curves (top-left) show rapid convergence with low variance. Hard task / rugged configuration curves (bottom-right) display slow, noisy convergence with plateaus and fluctuations. Intermediate cells smoothly interpolate between these extremes, confirming the diffusion model has learned a continuous latent manifold structured by difficulty. This interpolation property is crucial for generating diverse training data spanning the full difficulty spectrum.

Algorithm 6: DABO Online Freeze-Thaw Optimization

Require: Pre-trained DA-PFN p_θ, budget B, search space Λ, initial config λ_0
Require: Difficulty MLPs: MLP_ϕ, MLP_ψ, MLP_ω
Ensure: Best configuration λ^* and performance f^*

Initialize history: $\mathcal{H} \leftarrow \emptyset$
Initialize active configurations: $\mathcal{A} \leftarrow \{\lambda_0\}$ with $b_{\lambda_0} = 0$
Initialize budget consumed: $t_{\text{total}} \leftarrow 0$
while $t_{\text{total}} < B$ **do**
  // Step 1: Compute difficulty for all active configurations
  for each $\lambda \in \mathcal{A}$ **do**
    $\mathbf{d}(\lambda, b_\lambda) \leftarrow \text{ComputeDifficulty}(\mathcal{H}, \lambda, b_\lambda)$ // Algorithm 5
    $h(\lambda) \leftarrow \max(1, \lfloor b_{\max} \cdot \exp(-\alpha \|\mathbf{d}_{\text{cfg}}(\lambda)\|_2) \rfloor)$ // Eq. 21
  end for

  // Step 2: Predict future performance using DA-PFN
  Compute adaptive threshold: $T \leftarrow f_{\text{best}} + \tau(\|\mathbf{d}_{\text{task}}\|) \cdot (1 - f_{\text{best}})$ // Eq. 22
  for each $\lambda \in \mathcal{A}$ **do**
    Sample predictions: $\{f_k\}_{k=1}^K \sim p_\theta(\cdot \mid \mathcal{H}, \lambda, b_\lambda + h(\lambda), \mathbf{d}(\lambda, b_\lambda))$
    DC-MFPI$(\lambda) \leftarrow \frac{1}{K} \sum_{k=1}^K \mathbb{I}[f_k > T]$ // Eq. 23
  end for

  // Step 3: Select and train most promising configuration
  $\lambda^+ \leftarrow \arg\max_{\lambda \in \mathcal{A}}$ DC-MFPI(λ)
  Resume training λ^+ from checkpoint at epoch b_{λ^+} for 1 epoch
  Evaluate: $f(\lambda^+, b_{\lambda^+} + 1) \leftarrow$ validation performance
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\lambda^+, b_{\lambda^+} + 1, f(\lambda^+, b_{\lambda^+} + 1))\}$
  $b_{\lambda^+} \leftarrow b_{\lambda^+} + 1$
  $t_{\text{total}} \leftarrow t_{\text{total}} + 1$

  // Step 4: Manage configuration pool
  if $t_{\text{total}} \mod 10 = 0$ **then**
    [] $\lambda_{\text{new}} \sim \text{Uniform}(\Lambda)$, $\mathcal{A} \leftarrow \mathcal{A} \cup \{\lambda_{\text{new}}\}$ with $b_{\lambda_{\text{new}}} = 0$
  end if
  if $b_{\lambda^+} \geq b_{\max}$ **then**
    [] $\mathcal{A} \leftarrow \mathcal{A} \setminus \{\lambda^+\}$
  end if
end while
$\lambda^* \leftarrow \arg\max_{(\lambda,t,f) \in \mathcal{H}} f$
return λ^*, $f^* = \max_{(\lambda,t,f) \in \mathcal{H}} f$

8. Computational Resources and Carbon Footprint

Training DABO requires significant but feasible computational resources. The diffusion model training uses 4 NVIDIA A40 GPUs for 72 hours (288 GPU-hours), while DA-PFN training uses 8 A40 GPUs for 96 hours (768 GPU-hours), totaling 1056 GPU-hours for the complete offline training phase. Using the ML CO2 Impact calculator [11] with A40 TDP of 300W and average US grid carbon intensity of 0.429 kg CO2/kWh, we estimate total carbon emissions of approximately 136 kg CO2 for full model training. This is comparable to other meta-learning approaches and substantially lower than large language model pre-training.

Crucially, this cost is amortized over all future HPO tasks. Once trained, DABO can be applied to unlimited HPO problems with only marginal online cost. For a single HPO run with budget 1000 steps, DA-PFN inference adds approximately 4 minutes of GPU time (250ms per iteration), consuming roughly 0.02 kWh and emitting 0.009 kg CO2. This is negligible compared to the actual model training being optimized, which typically consumes hours

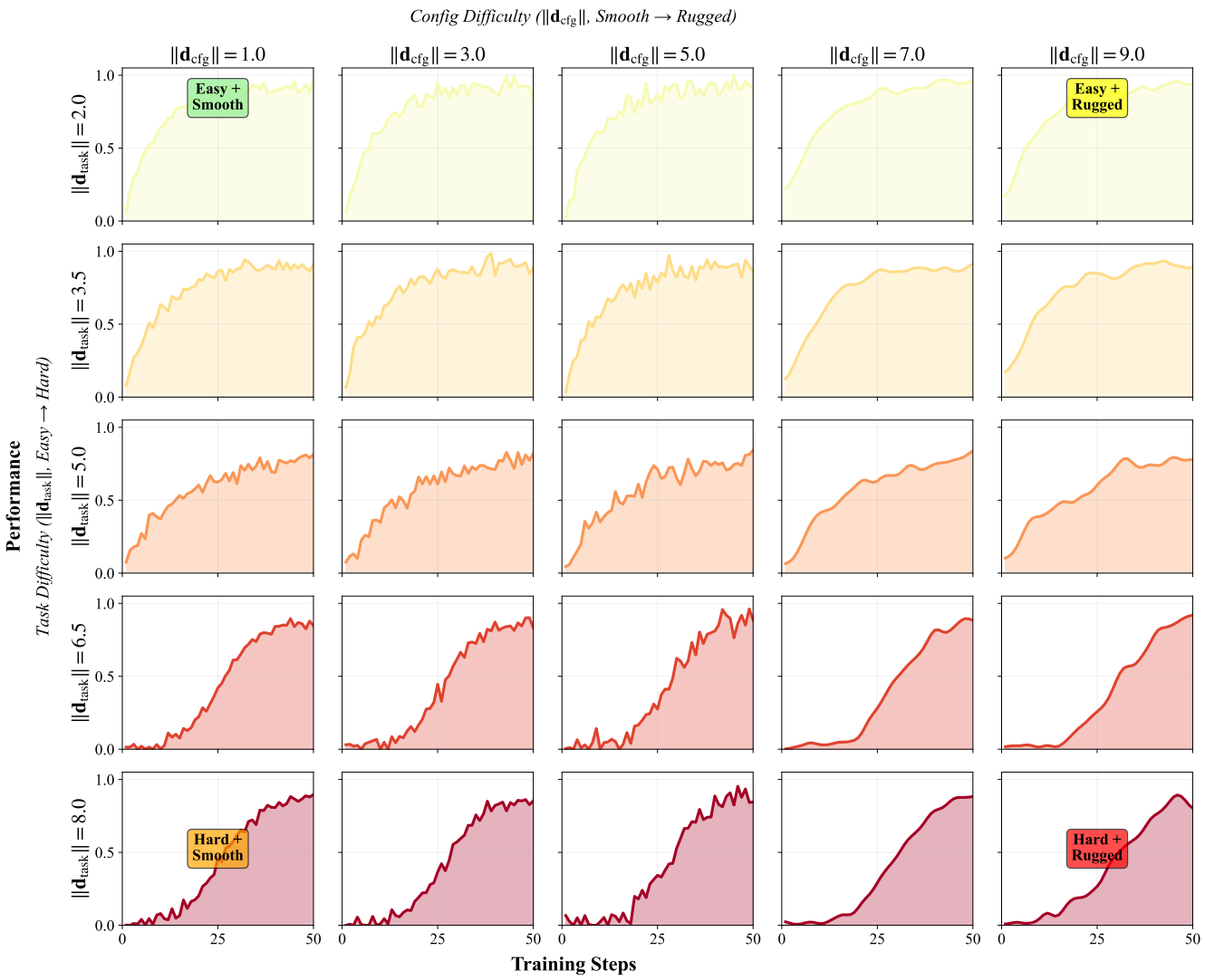

Figure 3. **Diffusion model interpolation in difficulty space.** 5×5 grid varying task difficulty (rows, easy→hard) and configuration difficulty (columns, smooth→rugged). Curves transition smoothly, validating learned difficulty manifold.

to days of compute.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *KDD*, pages 2623–2631, 2019. 5

[2] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. In *ICLR*, 2018. 8

[3] Archit Bansal, Danny Stoll, Maciej Janowski, Arber Zela, and Frank Hutter. Jahs-bench-201: A foundation for research on joint architecture and hyperparameter search. In *NeurIPS*, pages 38788–38802, 2022. 3

[4] Yutian Chen, Xingyou Song, Chansoo Lee, Zi Wang, Richard Zhang, David Dohan, Kazuya Kawakami, Greg Kochanski, Arnaud Doucet, Marc'Aurelio Ranzato, et al. Towards learning universal hyperparameter optimizers with transformers. In *NeurIPS*, pages 32053–32068, 2022. 9

[5] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, pages 3460–3468, 2015. 8

[6] Katharina Eggensperger, Philipp Müller, Neeratyoy Mallik, Matthias Feurer, Rene Sass, Aaron Klein, Noor Awad, Marius Lindauer, and Frank Hutter. HPOBench: A collection of reproducible multi-fidelity benchmark problems for HPO. In *NeurIPS*, 2021. 3

[7] Matthias Feurer, Benjamin Letham, and Eytan Bakshy. Scal-

able meta-learning for Bayesian optimization using ranking-weighted Gaussian process ensembles. In *ICML*, 2018. 9

[8] Matthias Feurer, Benjamin Letham, Frank Hutter, and Eytan Bakshy. Practical transfer learning for bayesian optimization. *arXiv preprint arXiv:1802.02219*, 2018. 9

[9] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast Bayesian optimization of machine learning hyperparameters on large datasets. In *AISTATS*, pages 528–536, 2017. 8

[10] Marcel Kollovieh, Abdul Fatir Ansari, Michael Bohlke-Schneider, Jasper Zschiegner, Hao Wang, and Yuyang Bernie Wang. Predict, refine, synthesize: Self-guiding diffusion models for probabilistic time series forecasting. In *NeurIPS*, pages 28341–28364, 2023. 8

[11] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019. 10

[12] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54): 1–9, 2022. 5

[13] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *ICLR*, 2019. 9

[14] Kashif Rasul, Calvin Seward, Ingmar Schuster, and Roland Vollgraf. Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting. In *ICML*, pages 8857–8868, 2021. 8

[15] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *ICLR*, 2021. 3

[16] Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. CSDI: Conditional score-based diffusion models for probabilistic time series imputation. In *NeurIPS*, pages 24804–24816, 2021. 8