DiffusionGrow: Continual Learning for Diffusion Language Models

Anonymous ACL submission

Abstract

Masked diffusion language models are increasingly useful as general denoisers, but adapting them to specialized domains can erode the pretrained denoising behavior needed for general generation. Existing parameter-efficient methods such as LoRA route both domain and retention examples through the same adapted computation, which can shift the original denoiser. We propose DIFFUSIONGROW, a function-preserving growth framework that adds timestep-conditioned trainable branches to selected layers while keeping the pretrained computation frozen and accessible through learned gates. The expanded model is exactly equivalent to the pretrained denoiser at initialization, yet gains capacity for domain specialization. For LLaDA-family 8B models, DG+LoRA combines frozen pretrained branches with low-rank trainable growth. Across eight effective-batch-1 adaptation settings spanning two backbones and four domains, DG+LoRA improves matched LoRA baselines, lowering average retention loss from 7.99 to 6.77, shifting average forgetting from -0.21 to -1.43, and reducing average validation loss from 6.06 to 5.80. These results show that preserving an explicit pretrained path can make continual adaptation more stable for large masked diffusion language models.

1 Introduction

Adapting a pretrained language model to a new domain requires a careful balance between plasticity and preservation. The model should acquire domain-specific behavior, but it should not overwrite the pretrained capabilities that support general generation. While this stability–plasticity problem is well studied for autoregressive LMs and parameter-efficient fine-tuning (Ramasesh et al., 2021; Wu et al., 2024b; Hu et al., 2022; Dettmers et al., 2023), it is less understood for masked diffusion language models, whose generation depends on denoising behavior across many corruption levels. Unlike autoregressive models, masked diffusion language models denoise partially corrupted sequences over a range of noise levels (Austin et al., 2021; Sahoo et al., 2024; Nie et al., 2025b). Adaptation can therefore damage not only clean-text predictions, but also the intermediate denoising behavior used during iterative generation.

A common practical solution is to keep the pretrained backbone frozen and train a small set of additional parameters, such as adapters or LoRA modules. These methods are efficient and preserve the original model at initialization, but after training they still route both domain and retention examples through the same adapted computation. As a result, the parameters that improve the target domain can also shift general-domain denoising behavior. Retention losses and replay data can mitigate this drift, but they do not by themselves give the model a structural way to keep the pretrained computation available while learning new domain-specific behavior.

We propose **DiffusionGrow**, a function-preserving growth framework for masked diffusion language models. The central idea is to expand selected layers so that adaptation receives new capacity while the original denoising computation remains explicitly available. Each grown layer contains a frozen pretrained branch and one or more trainable branches, combined by timestep-conditioned gates. All branches are initialized as exact copies of the pretrained layer, so the expanded model computes exactly the same function as the original denoiser at initialization. During adaptation, the frozen branch remains available as an anchor for retention, while the trainable branches provide capacity for domain specialization.

This design adapts function-preserving model growth (Chen et al., 2016; Rusu et al., 2016;

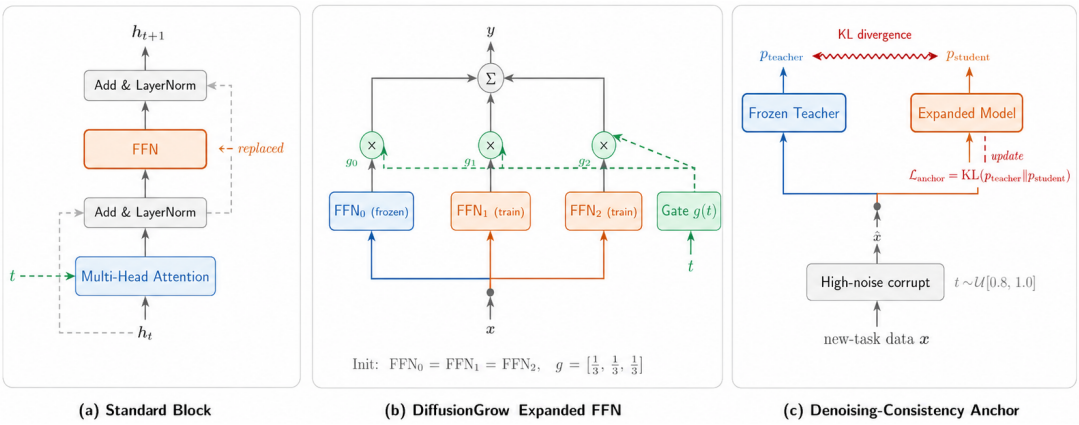


Figure 1: Overview of DiffusionGrow. **(a)** A standard masked diffusion transformer block. **(b)** DiffusionGrow expands the FFN into multiple branches (one frozen, others trainable) weighted by a timestep-conditioned simplex gate. At initialization, all branches are identical, so the model is exactly preserved. **(c)** The denoising-consistency anchor regularizes the expanded model toward the frozen teacher on high-noise corruptions.

Adila et al., 2026) to the masked diffusion setting. The timestep-conditioned gates are important: the model can choose different mixtures of frozen and adapted computation depending on the corruption level. This matches the structure of diffusion language modeling, where early high-noise denoising and later low-noise refinement may require different degrees of specialization and preservation. DiffusionGrow therefore turns retention from a purely loss-level constraint into an architectural property: the pretrained denoising path remains explicitly available throughout adaptation.

We instantiate this idea in **DG+LoRA**, a parameter-efficient variant of DiffusionGrow for large masked diffusion language models. Instead of fully training newly grown branches, DG+LoRA equips the trainable branches with low-rank updates while keeping the original branch frozen. This preserves the memory efficiency of LoRA-style adaptation while adding an explicit function-preserving route for the pretrained model. We evaluate adaptation using target-domain validation loss together with held-out general-domain retention loss and forgetting, which measure whether the adapted denoiser preserves pretrained behavior.

Our contributions are:

1. We study continual domain adaptation for masked diffusion language models, focusing on the tradeoff between target-domain improvement and pretrained denoising retention.
2. We propose DiffusionGrow, a timestep-conditioned function-preserving growth method that keeps the original pretrained computation frozen and exactly preserves the denoiser at initialization.
3. We introduce DG+LoRA, a parameter-efficient variant for large masked diffusion LMs, and evaluate it across LLaDA-family backbones and multiple domains.

2 Related Work

Diffusion language models. Discrete diffusion on text traces to structured denoising in discrete state spaces (Austin et al., 2021). Continuous diffusion approaches demonstrated controllable generation (Li et al., 2022), while latent diffusion improved practicality (Lovelace et al., 2023). Score-based methods sharply improved discrete diffusion quality (Lou et al., 2024). Sahoo et al. (2024) proposed a simple masked diffusion recipe with a Rao Blackwellized objective; Nie et al. (2025a) demonstrated favorable scaling trends; and Nie et al. (2025b) extended masked diffusion into the LLM regime with response-only fine-tuning.

Forgetting and continual learning. Elastic Weight Consolidation (EWC; Kirkpatrick et al., 2017), Learning without Forgetting (LwF; Li and Hoiem, 2018), and replay methods (Lopez-Paz and Ranzato, 2017; Rolnick et al., 2019) operate within fixed capacity. Ramasesh et al. (2021) emphasized the representational nature of forgetting. Recent surveys and empirical studies reframe forgetting across continual pretraining, domain adaptation, and alignment (Wu et al., 2024b; Liao et al., 2025). For diffusion LMs, forgetting may distort the pretrained mapping across corruption levels, motivating methods that change *where* new information is

stored.

Function-preserving growth. Net2Net (Chen et al., 2016) introduced width- and depth-preserving transformations. Progressive neural networks (Rusu et al., 2016) freeze old columns and add new ones. Wu et al. (2024a) showed block expansion improves LLMs; Mazzawi et al. (2024) studied pretrained initializations for larger models. Most directly, Adila et al. (2026) proposed duplicating feed-forward network (FFN) capacity with compensatory scaling for causal LMs. Our work adapts this insight to masked diffusion with timestep-conditioned gates and attention-layer growth.

Mixture of experts. Sparse Mixture-of-Experts (MoE) models (Shazeer et al., 2017; Fedus et al., 2022; Lepikhin et al., 2021) route inputs to specialized expert subnetworks. DiffusionGrow's gated branches resemble a dense MoE where all experts are evaluated but weighted by the gate. The key difference is initialization: MoE experts are trained from scratch or randomly initialized, while DiffusionGrow branches are exact copies of the pretrained FFN, guaranteeing function preservation. The frozen branch further distinguishes our approach: it provides a permanent, unmodifiable expert that anchors the pretrained computation.

Parameter-efficient fine-tuning. LoRA (Hu et al., 2022), adapters (Houlsby et al., 2019), and QLoRA (Dettmers et al., 2023) adapt models with few trainable parameters but modify the original computation graph. DiffusionGrow is complementary: it structurally preserves the original denoiser while adding new capacity.

3 Method

3.1 Problem Setup

Let $x = (x_1, \ldots, x_n)$ be a token sequence. A masked diffusion LM defines a corruption process replacing tokens with a mask symbol over timesteps $t \in \{1, \ldots, T\}$. The pretrained denoiser f_{θ_0} returns token logits:

$$f_{\theta_0}(x_t, t, c) \in \mathbb{R}^{n \times |\mathcal{V}|}, \tag{1}$$

where x_t is the corrupted input, c is optional context, and $\mathcal{V}$ is the vocabulary. The masked-diffusion training objective is:

$$\mathcal{L}_{\text{diff}}(\theta) = \mathbb{E}_{x,t,x_t}\left[\frac{-1}{|M_t|}\sum_{i \in M_t} \log p_\theta(x_i | x_t, t, c)\right], \tag{2}$$

where M_t denotes masked positions. The feed-forward block in layer ℓ is $F_\ell(h) = V_\ell\, \phi(U_\ell h + b_\ell) + d_\ell$.

3.2 Noise-Conditioned Function-Preserving Growth

Growth in DiffusionGrow is a *one-time architectural transformation applied before training*, not a dynamic decision made during optimization. The practitioner specifies two hyperparameters: which layers to grow (the set $\mathcal{S}$, controlled by a fraction parameter such as "top 50%" or "all layers") and how many branches per layer (K, typically 2). The selected layers are then expanded, all parameters are frozen or marked trainable, and training proceeds on this fixed architecture. As we show in Section 4, *which* layers to grow matters far more than *how many* branches to add: growing all layers with $K = 2$ outperforms growing 50% with $K = 4$.

For each selected layer $\ell \in \mathcal{S}$, we replace the FFN with a multi-branch module:

$$\widetilde{F}_\ell(h, t) = \sum_{j=1}^{K} \alpha_{\ell,j}(t)\, V_{\ell,j}\, \phi(U_{\ell,j} h + b_{\ell,j}) + d_\ell, \tag{3}$$

where $\alpha_\ell(t) = (\alpha_{\ell,1}(t), \ldots, \alpha_{\ell,K}(t))$ lies on the probability simplex for every t, implemented via a linear projection of the timestep embedding followed by softmax. All branches are initialized as exact copies of the pretrained FFN. Branch $j = 1$ is frozen; branches $j \geq 2$ are trainable.

Theorem 1 (Function preservation)**.** *If all branch parameters are exact copies of the pretrained FFN at initialization and $\alpha_\ell(t)$ lies on the simplex for every t, then $\widetilde{F}_\ell(h, t) = F_\ell(h)$ for all h and t. Consequently, the expanded model is identical to the pretrained denoiser for every input (x_t, t, c).*

Proof sketch. At initialization, every branch computes $u = V_\ell \phi(U_\ell h + b_\ell)$. Then $\widetilde{F}_\ell(h, t) = (\sum_j \alpha_{\ell,j}(t)) \cdot u + d_\ell = u + d_\ell = F_\ell(h)$. Full proof in Appendix A.1. □

Lemma 2 (Diffusion objective preservation)**.** *Under Theorem 1, the expanded model preserves the masked-diffusion objective: $\mathcal{L}_{\text{diff}}(\widetilde{\theta}) = \mathcal{L}_{\text{diff}}(\theta_0)$ for any timestep distribution $p(t)$ and corruption kernel q_t. This extends to response-only supervised fine-tuning (SFT) objectives.*

Table 1: Quantized qualitative retention examples on WikiText probes. **Bold** = matches the masked truth token.

Model	Dataset	Context ([MASK] masked)	Truth	Pretrained	LoRA-r5	DG+LoRA
8B-Instruct	Coding	...a large crustacean with a body [MASK] up to 60 cm...	length	**length**	measuring	**length**
8B-Instruct	Patent	...the claw of H. americanus is orange or [MASK]...	red	**red**	brown	**red**
8B-Instruct	Curated Math	...Shikamaru is assigned the [MASK] of locating two members...	task	**task**	mission	**task**
8B-Instruct	Economics	...Returning to Australia in [MASK] 1971, Headlam...	June	**June**	May	**June**
LLaDA-1.5	Coding	...most hotels [MASK] outside of downtown...	moved	**moved**	relocated	**moved**
LLaDA-1.5	Patent	...playing mostly as a [MASK] tailback to the fullback position...	backup	**backup**	blocking	**backup**
LLaDA-1.5	Curated Math	...at the [MASK] of the Mobile and Ohio Railroad...	intersection	**intersection**	junction	**intersection**
LLaDA-1.5	Economics	...a brief slowdown of the [MASK] at the end of the Depression...	economy	**economy**	industry	**economy**

Proof sketch. Identical logits ⇒ identical softmax ⇒ identical per-position cross-entropy ⇒ identical objective in expectation. See Appendix A.1. □

Theorem 3 (Post-training drift bound)**.** *Let L_ϕ be the Lipschitz constant of activation ϕ, and let $\delta_\ell = \max_j \|(\Delta U_{\ell,j}, \Delta V_{\ell,j})\|_F$ be the maximal branch deviation from initialization after training. For any input h with $\|h\| \le B_h$, the per-layer output drift is bounded:*

$$\|\widetilde{F}_\ell(h,t) - F_\ell(h)\| \le C_\ell \cdot \delta_\ell \cdot B_h, \qquad (4)$$

where C_ℓ depends on L_ϕ, the pretrained weight norms, and K. The frozen branch ensures that when gates route entirely to $j = 1$, the drift is exactly zero.

Proof sketch. Triangle inequality over the gated mixture, bounding each branch's deviation from the pretrained computation. Full derivation in Appendix A.1. □

Proposition 4 (Recoverability of the pretrained denoiser)**.** *The pretrained model lies in the parameter space of DiffusionGrow. Setting $\alpha_{\ell,1}(t) = 1$ for all t recovers the original FFN exactly, regardless of trainable branch values.*

3.3 Training Enhancements

We find two enhancements significantly improve gate learning:

Gate temperature annealing. We compute $\alpha_\ell(t) = \mathrm{softmax}(\mathrm{logits}/\tau)$ with τ linearly annealed from $\tau_0 = 2.0$ (near-uniform routing early in training) to $\tau_1 = 0.5$ (sharp specialization by end). This allows branches to gradually differentiate rather than committing to routing patterns before they have learned useful features.

Separate gate learning rate. Gate networks are small but critical for routing quality. We use a $5\times$ learning rate multiplier for gate parameters, enabling faster convergence of the routing mechanism relative to branch weights.

3.4 Extending Growth to Attention Projections

The basic DiffusionGrow formulation only grows FFN layers, while LoRA adapts both attention and FFN projections. To close this gap, we extend gated branching to attention layers via *GatedLoRAProjection*, a drop-in replacement for any linear projection (Q, K, V, output) that adds $K-1$ LoRA branches with a shared simplex gate:

$$\widetilde{W}(h,t) = \sum_{j=1}^{K} \alpha_j(t)\,(Wh + \Delta W_j h)\,, \qquad (5)$$

where $\Delta W_1 = 0$ for the frozen base and $\Delta W_j = B_j A_j$ for $j \ge 2$, thereby preserving the original function at initialization. The block's original attention implementation (including rotary embeddings and flash attention) remains unmodified.

Naming of DiffusionGrow variants. We use the following names throughout the experiments. **DG** denotes the full DiffusionGrow, in which selected FFN layers are duplicated into one frozen, pretrained branch and one or more trainable full FFN branches. **DG+LoRA** is the parameter-efficient variant used for large LLaDA-family models: the

original branch remains frozen, while each trainable branch is adapted using LoRA-style low-rank updates rather than full FFN duplication. **v2** refers to the training enhancements introduced in Section 3.3: gate temperature annealing and a larger learning rate for gate parameters. **Attn** means that the same gated growth idea is also applied to attention projections through GatedLoRAProjection, as described in Section 3.4. Thus, **DG+LoRA v2 + Attn** means parameter-efficient grown branches with LoRA updates, v2 gate training, and gated attention-projection growth.

3.5 Denoising-Consistency Anchor

To regularize the adapted model without replaying pretraining data, we add a KL divergence penalty between the frozen teacher and adapted model on high-noise corruptions:

$$\mathcal{L}_{\text{anc}}(\theta) = \mathbb{E}_{x,t,x_t}\left[\frac{1}{|M_t|}\sum_{i\in M_t}\text{KL}(p_{\theta_0}\|p_\theta)\right], \quad (6)$$

where t is sampled from a high-noise distribution $p_{\text{anc}}(t)$ biased toward large mask ratios. The total objective is $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{task}} + \lambda\mathcal{L}_{\text{anc}}$.

4 Experiments

4.1 A First Look: Quantized Token-Level Retention

Before reporting aggregate metrics, we provide a small token-level view of how quantized 8B adaptations affect pretrained predictions on held-out general-domain contexts. We mask a single token in each context and compare the pretrained model, a parameter-matched LoRA baseline, and DG+LoRA. Table 1 shows representative cases from the two LLaDA-family backbones and four adaptation domains.

The examples illustrate the retention behavior targeted by our method. In each case, LoRA changes the pretrained prediction to a plausible but different token (e.g., "measuring" instead of "length", "mission" instead of "task", or "industry" instead of "economy"), while DG+LoRA preserves the pretrained answer. These examples are not intended as aggregate accuracy measurements; rather, they show the kind of token-level drift that can occur even when the replacement is semantically reasonable. The main quantitative evidence is reported in Sections 4.3 and 4.4, where we evaluate retention loss, forgetting, and target-domain validation loss across models and domains.

Table 2: Retention-aware joint training at LLaDA-8B for 5,000 steps with $\lambda_{\text{ret}} = 0.5$. Top: feasible PubMed baselines under the same domain + WikiText retention objective. Growth-based methods achieve negative forgetting while LoRA remains positive even at higher rank. Bottom: DG+LoRA vs. LoRA on additional domains. Together with PubMed, this covers four target domains. DG+LoRA consistently improves forgetting and retention behavior; target validation loss is competitive and improves in most domains.

Method	Tr.%	Val ↓	Fgt ↓
PubMed baselines (all with joint training)			
LoRA r=8	0.3	7.185	+0.153
LoRA r=16	0.5	7.183	+0.149
LoRA r=32	1.1	7.140	+0.144
Adapters (b=64)	0.4	6.654	−0.098
Random grow + LoRA	0.2	**6.621**	−0.080
Uniform grow + LoRA	0.2	6.623	−0.064
DG+LoRA (ours)	∼0.3	6.626 ±.004	**−0.077**±.002
Multi-domain: DG+LoRA vs. LoRA			
Patent: LoRA	∼0.5	**6.139**	+0.387
Patent: DG+LoRA	∼0.3	6.364	**−0.055**
Chemistry: LoRA	∼0.5	4.676	+0.386
Chemistry: DG+LoRA	∼0.3	**4.660**	**−0.093**
Legal: LoRA	∼0.5	6.456	+0.328
Legal: DG+LoRA	∼0.3	**6.374**	**−0.118**

All methods receive the same domain + WikiText retention loss. DG+LoRA adapts 50% of FFN layers; LoRA adapts all attention+FFN layers. "Random grow" initializes branches randomly; "Uniform grow" uses fixed uniform gates. Full FT, EWC, and LwF require all 8B parameters trainable, exceeding 32GB GPU memory.

4.2 Setup

We evaluate parameter-efficient continual adaptation on two publicly released LLaDA-family masked diffusion backbones, `GSAI-ML/LLaDA-8B-Instruct` and `GSAI-ML/LLaDA-1.5`, both loaded with NF4 4-bit quantization and bfloat16 compute; the pretrained backbone is frozen and only adapter and gate parameters train. We compare a LoRA baseline (rank 5, $\alpha = 10$, dropout 0.05; 13.76M trainable parameters) against **DG+LoRA** (grow rank 16, $\alpha = 32$, grow fraction 0.5, $K = 2$ branches; 12.58M trainable parameters) at a closely matched trainable-parameter budget. All 8B runs use a joint domain retention loss $\mathcal{L} = \mathcal{L}_{\text{domain}} + 0.5\,\mathcal{L}_{\text{wikitext}}$ for 5000 micro-steps with sequence length 256, AdamW (weight decay 0.01), constant learning rate 3×10^{-5}, max gradient norm 1.0, and seed 42. Retention training and evaluation use WikiText-103; the four target domains are coding (codeparrot-clean), patent

(BigPatent), curated math (filtered OpenWebMath), and economics (BIS speeches). Forgetting is defined as post-adaptation retention loss minus pretrained retention loss; lower is better for validation, retention, and forgetting. The full ablation protocol, dataset construction details, and MDLM-OWT 170M ablations appear in Appendices A.17, A.16, and A.4. We include an autoregressive reference comparison in Section 4.5. All 8B experiments run on a single NVIDIA RTX 5090 (32GB).

4.3 Main Result: Growth Uses Retention Supervision Better Than LoRA

Manual routing first verifies that the frozen branch can recover pretrained behavior. We now show that learned gates can recover this behavior automatically when trained with a joint objective that includes both domain adaptation and retention (Figure 2).

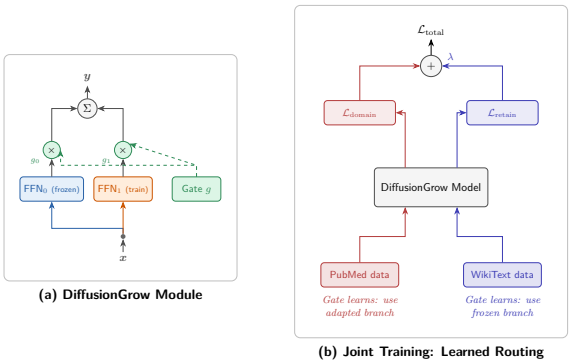
We train both LoRA r=16 and DG+LoRA r=16 (50% layers) on LLaDA-8B for 5,000 steps with a joint loss:

$$\mathcal{L} = \mathcal{L}_{\text{domain}} + \lambda_{\text{ret}} \cdot \mathcal{L}_{\text{WikiText}},$$

where $\lambda_{\text{ret}} = 0.5$. Both methods receive the *same* retention supervision. The question: does DG's frozen branch help when both methods are explicitly trained to retain?

Table 2 presents joint training results. The top panel compares all feasible methods on PubMed. Growth-based methods (random grow, uniform grow, DG+LoRA) all achieve val loss around 6.62 and negative forgetting, substantially outperforming LoRA at any rank: even LoRA r=32 (1.1% trainable, val 7.140) cannot match growth methods at 0.2% trainable (val 6.62). Increasing LoRA rank from 8 to 32 barely helps (7.185→7.140) and forgetting remains positive at all ranks (+0.144 to +0.153). The gap between DG+LoRA, random growth, and uniform growth is small at 8B with joint training, indicating that the growth architecture itself matters more than initialization or gate design. Adapters (val 6.654, fgt −0.098) fall between LoRA and growth methods.

The bottom panel extends DG+LoRA vs. LoRA across three additional domains. LoRA exhibits positive forgetting in all four domains (+0.328 to +0.387), while DG+LoRA achieves negative forgetting everywhere (−0.055 to −0.118). DG wins on val loss in three of four domains; patent is the exception, where LoRA's broader weight modification adapts better (6.139 vs. 6.364) despite worse retention. The joint loss resembles replay (Rolnick et al., 2019), but replay with LoRA merely slows forgetting. Growth-based methods exploit the retention signal more effectively because the frozen or copied branch provides a reference computation that can serve retention without interfering with domain adaptation.



(a) DiffusionGrow Module

(b) Joint Training: Learned Routing

Figure 2: Joint training: frozen branch (FFN$_0$) and trainable branch (FFN$_1$) mixed by a learned gate. Domain loss routes to adapted branch; retention loss routes to frozen branch.

4.4 Effective-Batch-1 Adaptation Across Backbones and Domains

To test whether DG+LoRA's joint-training advantage persists under stricter optimization constraints and across additional 8B-scale backbones, we run a controlled effective-batch-1 matrix. We define effective batch size as per-device batch size multiplied by gradient accumulation steps; thus, effective-batch-1 uses per-device batch size 1 and gradient accumulation 1 (no gradient accumulation). We evaluate two diffusion LMs, LLaDA-8B-Instruct and LLaDA-1.5, across four domains: coding (codeparrot-clean), patent (BigPatent), curated math, and economics. All runs use NF4 quantization of the frozen backbone, learning rate 3×10^{-5}, seed 42, and 5,000 steps, with parameter-matched LoRA (rank 5) versus DG+LoRA as the only varying axis.

Table 3 shows DG+LoRA improves all three metrics on every (backbone, domain) cell. Validation-loss gains are largest on coding (−0.38 nats) and curated math (−0.61 nats); retention loss improves by 1.4-1.9 nats outside coding; forgetting becomes substantially more negative everywhere, with DG+LoRA reducing it by 0.7 to 1.8 relative

Table 3: Effective-batch-1 joint adaptation across two 8B diffusion backbones and four domains. Effective batch size is per-device batch size multiplied by gradient accumulation steps; here all runs use per-device batch size 1 and gradient accumulation 1 (no gradient accumulation), with an NF4 frozen backbone, LR=3×10^{-5}, seed 42, and 5,000 steps. DG+LoRA improves all three metrics on every matched backbone-domain pair.

Backbone	Domain	Method	Val loss ↓	Retention loss ↓	Forgetting ↓
LLaDA-8B-Instruct	Coding	LoRA r=5	4.3734	7.0404	−0.6935
LLaDA-8B-Instruct	Coding	DG+LoRA	**3.9928**	**6.9635**	**−0.7704**
LLaDA-8B-Instruct	Patent	LoRA r=5	6.4363	8.0912	−0.0629
LLaDA-8B-Instruct	Patent	DG+LoRA	**6.4124**	**6.6958**	**−1.4583**
LLaDA-8B-Instruct	Curated math	LoRA r=5	6.1706	8.2631	−0.1142
LLaDA-8B-Instruct	Curated math	DG+LoRA	**5.5670**	**6.4946**	**−1.8826**
LLaDA-8B-Instruct	Economics	LoRA r=5	7.2572	8.4881	+0.0082
LLaDA-8B-Instruct	Economics	DG+LoRA	**7.2230**	**6.9191**	**−1.5607**
LLaDA-1.5	Coding	LoRA r=5	4.3701	7.0231	−0.7273
LLaDA-1.5	Coding	DG+LoRA	**3.9965**	**6.9553**	**−0.7952**
LLaDA-1.5	Patent	LoRA r=5	6.4452	8.0116	−0.1954
LLaDA-1.5	Patent	DG+LoRA	**6.4076**	**6.6995**	**−1.5075**
LLaDA-1.5	Curated math	LoRA r=5	6.1819	8.3326	−0.0409
LLaDA-1.5	Curated math	DG+LoRA	**5.5662**	**6.4854**	**−1.8881**
LLaDA-1.5	Economics	LoRA r=5	7.2499	8.6319	+0.1374
LLaDA-1.5	Economics	DG+LoRA	**7.2216**	**6.9124**	**−1.5820**

to LoRA. The pattern is essentially identical on LLaDA-8B-Instruct and LLaDA-1.5, indicating the advantage is architectural rather than backbone-specific. The magnitude of the forgetting gap here is larger than in the joint-training comparison at the standard-batch setting (Table 2), suggesting LoRA's retention is particularly fragile in the effective-batch-1 setting. A multi-seed re-run on the LLaDA-8B-Instruct $\times$ curated math cell (seeds $\{42, 123, 456\}$) preserves DG+LoRA's retention and forgetting advantage (mean retention 6.70 vs. 7.71; forgetting -1.54 vs. -0.54); full numbers are reported in Appendix Table 13.

4.5 Autoregressive Reference Comparison

To contextualize the retention-adaptation tradeoff beyond masked diffusion backbones, Table 4 includes a within-scale autoregressive reference using Qwen2.5-7B on the same coding and curated-math domains. AR and masked diffusion losses are not directly comparable because the model families differ in objective, tokenization, corruption process, and likelihood; we therefore use this table as within-family contextualization rather than as an architectural ranking.

The AR rows show the expected behavior of mature autoregressive adaptation: domain validation loss improves with only small retention changes. Within the masked diffusion family, DG+LoRA consistently improves the adaptation-retention tradeoff over LoRA. On coding, DG+LoRA gives the best LDM validation and retention losses for both 8B-Instruct and LLaDA-1.5. On curated math, LoRA-r5 reduces forgetting only mildly and worsens validation loss relative to the quantized pretrained baseline, whereas DG+LoRA improves validation loss and substantially reduces retention loss. This supports the view that function-preserving growth is especially useful when stable adaptation requires preserving pretrained denoising behavior.

5 Discussion

Evaluation lessons. Our multi-level evaluation reveals gaps between metrics. Aggregate perplexity suggests LoRA and DG are comparable at 170M, where both show negative forgetting. At 8B, however, the qualitative examples in Table 1 show that LoRA can shift pretrained predictions to plausible alternatives, such as "length"→"measuring", "task"→"mission", and "economy"→"industry", while DG+LoRA preserves the pretrained token. This supports the retention-loss trends in Table 2 and the effective-batch-1 results in Table 3. Meanwhile, downstream accuracy (MMLU/ARC) tells a different story: LoRA improves reasoning more than DG (Appendix A.5), revealing a tradeoff invisible to any single metric. These findings support evaluation protocols that combine retention loss, target-domain validation loss, downstream tasks, and token-level probes.

Architectural insights. Growth breadth dominates depth (0.9-nat gap in Table 5). Gate temperature annealing and separate gate LR are orthogonal improvements that collectively make DG surpass

Table 4: Domain-wise continual adaptation comparison between autoregressive (AR) baselines and masked diffusion language models (LDMs). AR and LDM losses are not directly comparable; this table is provided as within-family contextualization.

Family	Model	Dataset	Method	Val Loss ↓	Retention Loss ↓	Forgetting ↓
AR	Qwen2.5-7B	coding	Pretrained	1.1819	2.4453	0.0000
AR	Qwen2.5-7B	coding	LoRA domain-only	**1.1133**	2.4287	-0.0166
AR	Qwen2.5-7B	coding	Replay LoRA	1.1308	**2.3413**	**-0.1039**
AR	Qwen2.5-7B	coding	O-LoRA	1.1561	2.4343	-0.0110
LDM	8B-Instruct	coding	Pretrained NF4	4.6296	7.7339	0.0000
LDM	8B-Instruct	coding	LoRA-r5	4.3734	7.0404	-0.6935
LDM	8B-Instruct	coding	DG+LoRA	**3.9928**	**6.9635**	**-0.7704**
LDM	LLaDA-1.5	coding	Pretrained NF4	4.6410	7.7505	0.0000
LDM	LLaDA-1.5	coding	LoRA-r5	4.3701	7.0231	-0.7273
LDM	LLaDA-1.5	coding	DG+LoRA	**3.9965**	**6.9559**	**-0.7952**
AR	Qwen2.5-7B	curated_math	Pretrained	1.5931	2.4453	0.0000
AR	Qwen2.5-7B	curated_math	LoRA domain-only	**1.5203**	2.4228	-0.0224
AR	Qwen2.5-7B	curated_math	Replay LoRA	1.5342	**2.3422**	**-0.1031**
AR	Qwen2.5-7B	curated_math	O-LoRA	1.5780	2.4327	-0.0125
LDM	8B-Instruct	curated_math	Pretrained NF4	5.8035	8.3773	0.0000
LDM	8B-Instruct	curated_math	LoRA-r5	6.1706	8.2631	-0.1142
LDM	8B-Instruct	curated_math	DG+LoRA	**5.5670**	**6.4946**	**-1.8826**
LDM	LLaDA-1.5	curated_math	Pretrained NF4	5.8131	8.3735	0.0000
LDM	LLaDA-1.5	curated_math	LoRA-r5	6.1819	8.3326	-0.0409
LDM	LLaDA-1.5	curated_math	DG+LoRA	**5.5662**	**6.4854**	**-1.8881**

LoRA (Table 6). The frozen branch is an architectural advantage only with joint training; without retention supervision, LoRA wins (Appendix A.11).

6 Conclusion

We introduced DIFFUSIONGROW, a function-preserving growth framework for retention-aware continual adaptation in masked diffusion language models. Instead of routing all examples through the same adapted computation, DiffusionGrow keeps the pretrained denoising path frozen and reachable while adding trainable capacity for domain specialization. In LLaDA-8B joint training, DG+LoRA achieves negative forgetting across the tested domains under the same retention objective where matched LoRA baselines remain in positive-forgetting territory. In an effective-batch-1 cross-backbone evaluation spanning two 8B diffusion backbones and four domains, DG+LoRA improves retention loss, forgetting, and validation loss in every matched setting, reducing average retention loss from 7.99 to 6.77 and shifting average forgetting from -0.21 to -1.43. These results support function-preserving growth as a stable and parameter-efficient route to continual adaptation in large masked diffusion language models.

Limitations

The frozen branch advantage requires joint training with retention supervision (Table 2); with domain-only training, LoRA wins (Appendix A.11). At 8B, memory restricts growth to 25-50% of layers. Multi-seed analysis covers PubMed and one effective-batch-1 curated-math setting; the remaining domains use seed 42. LoRA outperforms DG on patent val loss despite DG's better retention. All 8B experiments use $\lambda_{\text{ret}} = 0.5$. Our evaluation protocol uses perplexity and token-level accuracy, which may not capture all dimensions of generation quality (e.g., coherence, factuality, or harmful content). The downstream evaluation covers only MMLU and ARC (128 examples each); broader benchmark coverage would strengthen conclusions.

Ethical considerations

This work studies how to evaluate and preserve language-model capabilities during domain adaptation. All experiments use publicly available datasets and pretrained checkpoints. No human subjects were involved. Improved domain adaptation could be misused for harmful specialization; our evaluation protocol can help detect such shifts. AI tools were used only for language polishing; all scientific content was developed and verified by the authors.

References

Dyah Adila, Hanna Mazzawi, Benoit Dherin, and Xavier Gonzalvo. 2026. Grow, don't overwrite: Fine-tuning without forgetting. *Preprint*, arXiv:2603.08647.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. 2021. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. 2016. Net2net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representations*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*, volume 36, pages 10088–10115. NeurIPS 2023 oral.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2021. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto. 2022. Diffusion-LM improves controllable text generation. In *Advances in Neural Information Processing Systems*.

Zhizhong Li and Derek Hoiem. 2018. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947.

Chonghua Liao, Ruobing Xie, Xingwu Sun, Haowen Sun, and Zhanhui Kang. 2025. Exploring forgetting in large language model pre-training. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2112–2127, Vienna, Austria. Association for Computational Linguistics.

David Lopez-Paz and Marc'Aurelio Ranzato. 2017. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*.

Aaron Lou, Chenlin Meng, and Stefano Ermon. 2024. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 32819–32848.

Justin Lovelace, Varsha Kishore, Chao Wan, Eliot Shekhtman, and Kilian Q. Weinberger. 2023. Latent diffusion for language generation. In *Advances in Neural Information Processing Systems*.

Hanna Mazzawi, Javier Gonzalvo, Michael Wunder, Sammy Jerome, and Benoit Dherin. 2024. Deep fusion: Efficient network training via pre-trained initializations. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 35225–35239.

Shen Nie, Fengqi Zhu, Chao Du, Tianyu Pang, Qian Liu, Guangtao Zeng, Min Lin, and Chongxuan Li. 2025a. Scaling up masked diffusion models on text. In *The Thirteenth International Conference on Learning Representations*.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2025b. Large language diffusion models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Vinay Venkatesh Ramasesh, Ethan Dyer, and Maithra Raghu. 2021. Anatomy of catastrophic forgetting: Hidden representations and task semantics. In *International Conference on Learning Representations*.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy P. Lillicrap, and Gregory Wayne. 2019. Experience replay for continual learning. In *Advances in Neural Information Processing Systems*.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive neural networks. *Preprint*, arXiv:1606.04671.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. 2024. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*, volume 37.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*.

Chengyue Wu, Yukang Gan, Yixiao Ge, Zeyu Lu, Jiahao Wang, Ye Feng, Ying Shan, and Ping Luo. 2024a. LLaMA pro: Progressive LLaMA with block expansion. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6518–6537.

Tongtong Wu, Linhao Luo, Yuan-Fang Li, Shirui Pan, Thuy-Trang Vu, and Gholamreza Haffari. 2024b. Continual learning for large language models: A survey. *Preprint*, arXiv:2402.01364.

A Appendix

A.1 Proof of Function Preservation

Proof of Theorem 1 (Function Preservation). We must show that the expanded FFN $\widetilde{F}_\ell(h,t)$ computes the same output as the original $F_\ell(h)$ for all hidden states $h \in \mathbb{R}^d$ and all timesteps t.

At initialization, every branch $j \in \{1,\dots,K\}$ has parameters $(U_{\ell,j}^{(0)}, V_{\ell,j}^{(0)}, b_{\ell,j}^{(0)}) = (U_\ell, V_\ell, b_\ell)$. Each branch therefore computes the same vector:

$$u(h) = V_\ell \phi(U_\ell h + b_\ell).$$

Substituting into the grown FFN:

$$\begin{aligned}\widetilde{F}_\ell(h,t) &= \sum_{j=1}^{K} \alpha_{\ell,j}(t) \cdot V_{\ell,j}^{(0)} \phi(U_{\ell,j}^{(0)} h + b_{\ell,j}^{(0)}) + d_\ell \\ &= \sum_{j=1}^{K} \alpha_{\ell,j}(t) \cdot u(h) + d_\ell \\ &= \left(\sum_{j=1}^{K} \alpha_{\ell,j}(t)\right) u(h) + d_\ell.\end{aligned}$$

Since $\alpha_\ell(t)$ lies on the simplex, $\sum_{j=1}^{K} \alpha_{\ell,j}(t) = 1$ for all t. Therefore:

$$\widetilde{F}_\ell(h,t) = u(h) + d_\ell = F_\ell(h).$$

For the full model, write the transformer as a composition $f_{\theta_0} = G_L \circ \cdots \circ G_1$ where each G_ℓ consists of attention, layer norm, and FFN. Replacing the FFN in layer $\ell \in \mathcal{S}$ with $\widetilde{F}_\ell$ yields a layer $\widetilde{G}_\ell$ satisfying $\widetilde{G}_\ell(z,t) = G_\ell(z,t)$ for all z, t (since only the FFN changes and it is pointwise equal). By composition, the full expanded model equals the pretrained model for every input (x_t, t, c). □

Proof of Lemma 2 (Objective Preservation). From Theorem 1, $\widetilde{f}_\theta(x_t,t,c) = f_{\theta_0}(x_t,t,c)$ for all inputs at initialization. Therefore:

$$\begin{aligned}p_{\widetilde{\theta}}(x_i \mid x_t, t, c) &= \mathrm{softmax}(\widetilde{f}_\theta(x_t,t,c)_i) \\ &= \mathrm{softmax}(f_{\theta_0}(x_t,t,c)_i) \\ &= p_{\theta_0}(x_i \mid x_t, t, c)\end{aligned}$$

for every token position i. The masked-diffusion objective is:

$$\mathcal{L}_{\text{diff}}(\theta) = \mathbb{E}_{x,t,x_t}\left[\frac{-1}{|M_t|}\sum_{i \in M_t} \log p_\theta(x_i | x_t, t, c)\right].$$

Since the per-position log-probabilities are identical pointwise, the expectations are equal: $\mathcal{L}_{\text{diff}}(\widetilde{\theta}) = \mathcal{L}_{\text{diff}}(\theta_0)$.

For response-only SFT, the objective restricts M_t to response tokens and conditions on prompt context c, but does not alter the fact that both models produce identical logits. The argument applies verbatim. □

Proof of Theorem 3 (Post-Training Drift Bound). After training, let $\Delta U_{\ell,j} = U_{\ell,j} - U_\ell$ and $\Delta V_{\ell,j} = V_{\ell,j} - V_\ell$ denote parameter deviations for branch j. Define:

$$\delta_\ell = \max_{j \geq 2} \max(\|\Delta U_{\ell,j}\|_F, \|\Delta V_{\ell,j}\|_F).$$

The grown FFN output is:

$$\widetilde{F}_\ell(h,t) = \sum_{j=1}^{K} \alpha_{\ell,j}(t) \cdot V_{\ell,j} \phi(U_{\ell,j} h + b_{\ell,j}) + d_\ell.$$

The drift $\|\widetilde{F}_\ell(h,t) - F_\ell(h)\|$ equals:

$$\begin{aligned}&\Big\| \textstyle\sum_j \alpha_{\ell,j}(t) V_{\ell,j} \phi(U_{\ell,j} h + b_{\ell,j}) \\ &\qquad - V_\ell \phi(U_\ell h + b_\ell) \Big\|.\end{aligned}$$

Since branch $j = 1$ is frozen, only trainable branches contribute. By triangle inequality:

$$\begin{aligned}&\leq \textstyle\sum_{j \geq 2} \alpha_{\ell,j}(t) \Big\| V_{\ell,j} \phi(U_{\ell,j} h + b_{\ell,j}) \\ &\qquad - V_\ell \phi(U_\ell h + b_\ell) \Big\|.\end{aligned}$$

For each $j \geq 2$, adding and subtracting $V_\ell \phi(U_{\ell,j} h + b_{\ell,j})$ and applying the Lipschitz property of ϕ (constant L_ϕ):

$$\begin{aligned}&\leq \|\Delta V_{\ell,j}\| \cdot \|\phi(U_{\ell,j} h + b_{\ell,j})\| \\ &\quad + \|V_\ell\| \cdot L_\phi \cdot \|\Delta U_{\ell,j}\| \cdot \|h\| \\ &\leq C_\ell \cdot \delta_\ell \cdot B_h,\end{aligned}$$

where $C_\ell = L_\phi(2\|V_\ell\| + \|U_\ell\| + \delta_\ell)$. Since $\sum_{j \geq 2} \alpha_{\ell,j}(t) \leq 1$, the bound follows.

When the gate routes entirely to the frozen branch ($\alpha_{\ell,1}(t) = 1$), all trainable branches have zero weight, so the drift is exactly zero regardless of δ_ℓ. □

Proof of Proposition 4. Setting $\alpha_{\ell,1}(t) = 1$ for all t gives:

$$\begin{aligned}\widetilde{F}_\ell(h,t) &= 1 \cdot V_\ell \phi(U_\ell h + b_\ell) + \textstyle\sum_{j \geq 2} 0 \cdot (\cdots) + d_\ell \\ &= F_\ell(h),\end{aligned}$$

since the frozen branch parameters $(U_{\ell,1}, V_{\ell,1}, b_{\ell,1}) = (U_\ell, V_\ell, b_\ell)$ are never modified. This holds for any values of the trainable branch parameters, confirming that the pretrained denoiser is always recoverable within the DiffusionGrow parameter space. □

A.2 DiffusionGrow Training Algorithm

Algorithm 1 summarizes the full procedure. The architecture is expanded once (initialization phase), then training proceeds on the fixed grown model (training phase). No further growth or architectural changes occur during training.

Algorithm 1 DiffusionGrow training

Require: Pretrained denoiser f_{θ_0}, layers $\mathcal{S}$, branch count K, data $\mathcal{D}$, anchor weight λ, temperature schedule $\tau_0 \to \tau_1$, gate LR multiplier γ

Initialization:
for each layer $\ell \in \mathcal{S}$ **do**
  Copy FFN into K branches; freeze branch 1
  Init gate network $\to$ uniform $\alpha_\ell(t)$
  (Optional) Add GatedLoRAProjection to Q/K/V/O with $B=0$ init
end for
Freeze all untouched backbone parameters

Training:
for step $s = 1, \ldots, S$ **do**
  $\tau \leftarrow \tau_0 + (\tau_1 - \tau_0) \cdot s/S$ ▷ Anneal temp.
  Sample (x, c) from $\mathcal{D}$
  Compute $\mathcal{L}_{\text{task}}$ with gates at temperature τ
  Compute $\mathcal{L}_{\text{anc}}$ on high-noise corruptions
  Update branches and gates (gate LR $= \gamma \cdot$ base LR)
end for
return adapted model $\widetilde{f}_\theta$

A.3 Additional Experimental Details

Hyperparameters. All MDLM experiments: AdamW optimizer, weight decay 0.01, cosine LR schedule with 3% warmup. LoRA: rank 16, α=32. Adapters: bottleneck dimension 64. DiffusionGrow: $K = 2$ branches, anchor weight $\lambda = 0.5$, $t_{\text{anchor}} \sim \mathcal{U}[0.8, 1.0]$. DG v2: τ annealed $2.0 \to 0.5$, gate LR $5\times$. DG+LoRA v2: rank 32, α=64.

Datasets. PubMed abstracts streamed from `ccdv/pubmed-summarization` (50K train, 2K val). arXiv abstracts from `TimSchopf/arxiv_categories` filtered to CS/Math/Statistics (50K train, 2K val). Patent descriptions from `big_patent` category A (50K train, 2K val, truncated to 8192 characters). Chemistry textbook passages from `jablonkagroup/LibreText_Chemistry` (50K train, 2K val). Legal documents from `umarbutler/open-australian-legal-corpus` (50K train, 2K val, truncated to 8192 characters). WikiText-103 and OpenWebText held-out shards (2K examples each) for retention. All datasets are publicly available on HuggingFace without gating.

Table 5: Core ablations on MDLM-OWT / PubMed (1500 steps). Layer coverage has the largest effect.

Layers	K	Gate	Tr.%	Val ↓	Fgt ↓
Top 25%	2	Unif.	7.7	5.482	−.223
Top 25%	2	Learn	7.7	5.452	−.234
Top 25%	2	+Anc$_u$	7.7	5.475	−.221
Top 25%	2	+Anc$_h$	7.7	5.463	−.235
Top 50%	2	+Anc$_h$	14.3	5.058	−.566
Top 50%	4	+Anc$_h$	33.4	5.011	−.593
All	2	+Anc$_h$	25.0	**4.564**	**−.941**

Anc$_u$ = uniform-t anchor, Anc$_h$ = high-noise anchor ($\lambda = 0.5$). K = branches. Fgt = forgetting.

Code domain. Python code from codeparrot-clean, 5000 steps, same hyperparameters as PubMed. Results: Full FT 2.200 (fgt −1.109), LoRA 2.534 (−0.969), DG 2.614 (−0.835), DG+LoRA 2.782 (−0.779). All negative forgetting confirms 170M model is too small for catastrophic forgetting.

A.4 170M Ablations, v2 Enhancements, and Manual Routing

Layer coverage dominates. Growing all layers with 2 branches (val loss 4.564) dramatically outperforms the top 50% (5.058) or top 25% (5.452), a 0.9-nat gap that substantially exceeds all other design choices. This suggests domain adaptation signal is distributed across layers rather than concentrated in the deepest ones. The practical recommendation is clear: allocate growth budget to breadth across layers.

Learned gates consistently improve over uniform. At 25% coverage, learned gates reduce val loss from 5.482 to 5.452 and improve forgetting from -0.223 to -0.234. The effect is modest at low coverage but compounds with full-layer growth.

More branches yield diminishing returns. Increasing from 2 to 4 branches at 50% coverage improves val loss from 5.058 to 5.011 (0.047 nats) at the cost of $2.3\times$ the trainable parameters (14.3%→33.4%). Growing all layers with 2 branches (25% params, val 4.564) is strictly preferable.

Table 6: Impact of v2 training enhancements on MDLM-OWT / PubMed (2000 steps). Each enhancement provides additive gains.

Method	Tr.%	Val ↓	Fgt ↓
LoRA (baseline)	1.4	4.394	−.985
DG-Full (prev.)	25.0	4.507	−.984
+ anneal	25.0	4.408	−1.039
+ anneal + gate LR	25.0	4.337	−1.075
+ anneal + gate LR + attn	25.3	**4.248**	**−1.137**
DG+LoRA (prev.)	0.9	4.645	−.888
DG+LoRA v2 (r=32)	1.7	4.472	−1.008

Anneal: τ 2.0→0.5. Gate LR: 5× multiplier. Attn: GatedLoRAProjection on Q/K/V/O. Forgetting is PubMed-only (Table 12 averages PubMed+arXiv).

Table 7: Feature isolation: attention growth vs. other architectural extensions on MDLM-OWT / PubMed (2000 steps).

Method	Tr.%	Val ↓	Fgt ↓
DG+LoRA + Attn	1.7	4.423	−1.001
DG+LoRA + DynBranch	1.7	4.576	−.903
DG+LoRA + Both	2.2	4.414	−1.006
DG+LoRA + ProgLoRA	1.7	4.540	−.931
DG v2 + Attn (full)	25.3	**4.248**	**−1.137**
DG+LoRA v2 + Attn	1.7	4.425	−1.001
DG+LoRA v2 + Attn + Dyn	2.2	4.415	−1.005

DynBranch = dynamic branch addition mid-training. ProgLoRA = progressive rank growth (8→32). Attn = attention growth.

The anchor's effect is subtle at 170M. The high-noise anchor provides marginal improvement at low layer coverage (rows 2 vs. 4: forgetting −0.234 vs. −0.235). We expect greater impact at larger scales where genuine forgetting emerges.

Table 6 isolates the contribution of each v2 enhancement. Starting from DG-Full (4.507), gate temperature annealing provides 0.099 nats (4.507→4.408), the 5× gate LR adds 0.071 nats (4.408→4.337), and attention growth adds 0.089 nats (4.337→4.248). Each improvement is additive, together yielding a 0.259-nat improvement over the original DG-Full and surpassing LoRA by 0.146 nats. Retention also improves monotonically, from −0.984 (DG-Full) to −1.137 (full v2 + attn), indicating that better routing produces not only better adaptation but also stronger knowledge preservation. The DG+LoRA hybrid also benefits substantially from v2 improvements: rank-32 with annealing and gate LR improves from 4.645 (0.9% params) to 4.472 (1.7% params), a 0.173-nat gain that brings it within 0.078 nats of LoRA while maintaining better retention (−1.008 vs. −0.985).

Table 7 compares three architectural extensions: attention growth, dynamic branch addition, and progressive LoRA rank growth. Attention growth is the most impactful feature, improving DG+LoRA from 4.576 to 4.423 (−0.153 nats). Dynamic branching provides modest benefit when combined with attention growth (4.423→4.414), while progressive LoRA rank growth (8→32) is least effective (4.540). The key insight is that standard DiffusionGrow only grows FFN layers, while LoRA adapts both attention and FFN projections. Extending growth to attention projections gives DiffusionGrow access to the same adaptation surface as LoRA. The full DG v2 + Attn variant (25.3% params) substantially outperforms DG+LoRA v2 + Attn (1.7% params) at 4.248 vs. 4.425, indicating that full branch duplication provides meaningful additional capacity over LoRA-based branches at 170M scale.

Table 8: Downstream accuracy after PubMed joint training at LLaDA-8B (128 examples each). LoRA's broader weight modification improves reasoning tasks more than DG's conservative frozen-branch approach.

Method	MMLU ↑	ARC ↑
Pretrained	30.5	30.5
LoRA (joint)	**36.7**	**32.8**
DG+LoRA (joint)	32.0	31.3

From manual to learned routing. With domain-only training, LoRA outperforms DiffusionGrow at every matched parameter budget because the gate does not learn to use the frozen branch (Appendix A.11). Entropy regularization, anchor loss, and manual gate override improve DG's retention but do not surpass LoRA at matched budgets (Appendix A.12). At 8B with manual gate override, DG recovers pretrained behavior by routing to the frozen branch (100% token recovery rate, Appendix A.12), but manual routing requires the operator to decide when to use it. The joint loss in Section 4.3 is what enables the gate to route automatically.

A.5 Downstream Task Evaluation

Table 8 shows downstream task performance. LoRA improves MMLU from 30.5% to 36.7% (+6.2pp) and ARC from 30.5% to 32.8% (+2.3pp), while DG+LoRA shows smaller gains (MMLU +1.5pp, ARC +0.8pp). This reveals a genuine

Table 9: LoRA rank sweep with joint training at LLaDA-8B on PubMed (5000 steps, $\lambda_{\text{ret}} = 0.5$). Increasing rank from 8 to 32 provides diminishing returns on adaptation and does not resolve forgetting. Growth methods included for comparison.

Method	Tr.%	Val ↓	Fgt ↓
LoRA r=8	0.3	7.185	+0.153
LoRA r=16	0.5	7.183	+0.149
LoRA r=32	1.1	7.140	+0.144
Adapters (b=64)	0.4	6.654	−0.098
Random grow + LoRA	0.2	6.621	−0.080
Uniform grow + LoRA	0.2	6.623	−0.064
DG+LoRA (ours)	∼0.3	6.626	−0.077

tradeoff between retention (where DG excels, Table 2) and task transfer (where LoRA excels). DG's frozen branch conservatively preserves the pretrained computation, which maintains perplexity but limits how much domain knowledge propagates to new reasoning tasks. Practitioners should choose based on their priority: perplexity-based retention and safety guarantees (DG) or maximum task transfer (LoRA).

A.6 LoRA Rank Sweep with Joint Training

Table 9 presents a LoRA rank sweep alongside all growth baselines, all trained with the same joint domain + WikiText retention objective at LLaDA-8B. We vary LoRA rank from 8 to 32, spanning a 4× range in trainable parameters (0.3% to 1.1%).

The rank sweep reveals two findings. First, increasing LoRA rank provides diminishing returns: doubling from r=8 to r=16 barely changes val loss (7.185→7.183), and quadrupling to r=32 yields only a 0.045-nat improvement (7.185→7.140). Forgetting decreases slightly (+0.153→+0.144) but remains positive at all ranks. Second, even LoRA r=32 at 1.1% trainable parameters cannot match any growth method at 0.2% trainable parameters (val ∼6.62). The gap between LoRA r=32 (7.140) and the worst growth variant (6.626) is 0.51 nats, larger than the total improvement from quadrupling LoRA's rank.

This suggests the limitation is architectural, not capacity-related. LoRA modifies the original weight matrix $W + \Delta W$, so domain adaptation and retention compete within the same parameters regardless of rank. Growth methods add separate capacity alongside the original computation, allowing the two objectives to be satisfied independently. The frozen or copied branch handles retention while the new branch handles domain adaptation. Adapters (val 6.654) fall between LoRA and growth methods, consistent with their hybrid nature: they add new parameters but still modify the residual stream that the original computation flows through.

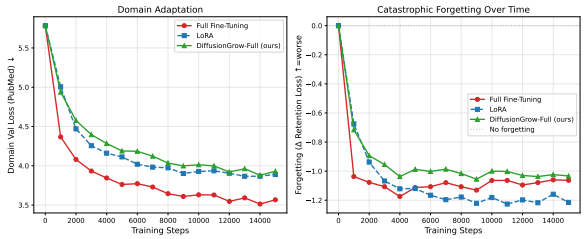


Figure 3: Forgetting dynamics over 15K steps on PubMed (MDLM-OWT 170M). All methods maintain negative forgetting throughout. Full FT peaks at −1.17 (4K) then degrades; LoRA shows best retention (−1.22); DG-Full achieves −1.03.

Table 10: LLaDA-8B domain-only adaptation on PubMed (no retention loss). LoRA dominates; DG shows positive forgetting at 2000 steps. Here — stands for not applicable.

Method	Tr.%	Steps	Val ↓	Fgt ↓
Pretrained	0.0	—	7.629	0.000
LoRA	0.5	500	6.985	−.118
DG (25%)	13.1	500	7.739	−.109
LoRA	0.5	2000	**6.378**	**−.307**
DG (25%)	13.1	2000	7.106	+.609
DG+LoRA (all)	0.3	2000	7.135	+.638
DG+LoRA + Attn	0.3	2000	7.065	+.583

A.7 Extended Training Dynamics

We train Full FT, LoRA, and DG-Full for 15,000 steps (7.5× standard budget) on PubMed (Figure 3). All methods maintain negative forgetting throughout, confirming 170M is too small for catastrophic forgetting. Full FT peaks at −1.174 (4K) then degrades to −1.063 (15K). LoRA achieves best final retention (−1.215). DG-Full reaches −1.033. Code domain (5000 steps) also shows negative forgetting for all methods (−0.779 to −1.109).

A.8 Domain-Only Scaling to LLaDA-8B

Table 10 presents domain-only scaling results. At 2000 steps, LoRA achieves val 6.378 with forgetting −0.307, while DG variants show positive forgetting (+0.58 to +0.64). Memory constraints restrict growth to 25% of layers. Joint training (Table 2) resolves this gap.

Table 11: Efficiency on MDLM-OWT PubMed (2000 steps).

Method	Tr.%	Mem	Tok/s	Val ↓
Full FT	100.0	5.1 G	**16.2K**	**3.909**
LoRA	1.4	3.2 G	12.7K	4.394
Adapters	1.4	**2.6 G**	15.7K	4.151
Unif. grow	14.3	3.1 G	17.4K	4.963
DG (ours)	14.3	6.6 G	10.1K	5.003

DG's higher memory is due to the frozen teacher copy for the anchor loss. Without the anchor, memory is comparable to uniform growth.

A.9 Efficiency

Table 11 shows that DiffusionGrow's main overhead is the frozen teacher copy for the anchor loss, increasing peak memory from 3.1 GB to 6.6 GB and reducing throughput by $\sim$42%. This can be amortized by computing the anchor every k steps. The DG+LoRA hybrid reduces memory substantially ($\sim$60 MB overhead per layer vs. $\sim$11 GB for full duplication at 8B).

A.10 Visualizations

Figure 4 (left) shows the adaptation-retention Pareto frontier, where DG v2 + Attn achieves the best tradeoff among non-replay methods. Figure 4 (right) shows learned gate weights across layers and timesteps. Figure 5 visualizes the ablation results. The full DiffusionGrow architecture is illustrated in Figure 1.

A.11 Gate Training Strategies at 170M

With domain-only training, the DiffusionGrow gate does not automatically route to the frozen branch for out-of-domain inputs. We tested several strategies to improve retention at MDLM-OWT 170M scale.

Entropy regularization ($\lambda = 0.05$) penalizes low gate entropy, preventing collapse to one branch. **Anchor loss** ($\lambda = 0.3$) adds KL divergence between frozen teacher and adapted model on high-noise corruptions. Results on PubMed (2000 steps):

- LoRA r=16 (1.4%): val=4.352, fgt=-0.994
- LoRA r=256 (18.3%): val=4.032, fgt=-1.161
- DG+LoRA v2 + ent + anchor (1.7%): val=4.568, fgt=-0.946
- Full DG + ent + anchor (25%): val=4.347, fgt=-1.046

At matched parameter budgets, LoRA wins on both axes at 170M scale. The frozen branch provides structural recoverability but does not translate to better forgetting without retention-aware training (which motivates the joint training approach in Section 4.3).

A.12 Inference-Time Controllability Details

DiffusionGrow enables inference-time control over the adaptation-retention tradeoff by adjusting gate routing. At MDLM-170M, sweeping the gate bias from fully adapted to fully frozen produces a smooth tradeoff (domain val 4.445 to 5.821, retention 4.524 to 5.473).

At LLaDA-8B with 2000 domain-only training steps and manual gate override:

- LoRA r=16: fgt=+0.236 (PubMed), +0.914 (arXiv). Permanent.
- DG+LoRA adapted: fgt=+0.111 (PubMed), -0.156 (arXiv).
- DG+LoRA frozen gates: fgt=-0.229 (PubMed), +0.203 (arXiv). Recoverable.

Over 2,000 WikiText-103 masked tokens at 8B, LoRA forgets 10 tokens that the pretrained model predicted correctly. DG with frozen gates recovers all 10 (100% recovery rate). Examples of LoRA's forgetting include biomedical contamination ("cultured" for "captive", "transitions" for "survives") and factual errors ("January" for "July").

These results demonstrate that manual gate override can recover pretrained behavior, but manual routing requires external domain knowledge. The joint training approach (Section 4.3) eliminates this requirement by teaching the gate to route automatically.

A.13 Extended Theoretical Discussion

Why the simplex is sufficient but not necessary. Theorem 1 requires $\sum_j \alpha_{\ell,j}(t) = 1$. This is stronger than strictly necessary: any set of weights summing to 1 suffices, even with negative values. We enforce the simplex constraint (nonnegativity) for three reasons: (1) interpretability, as each weight represents a routing probability; (2) optimization stability, preventing large opposite-signed weights that cancel; and (3) connection to mixture-of-experts literature.

Relationship to LoRA function preservation. LoRA initializes $\Delta W = BA$ with $B = 0$, ensuring zero perturbation at initialization. This is a

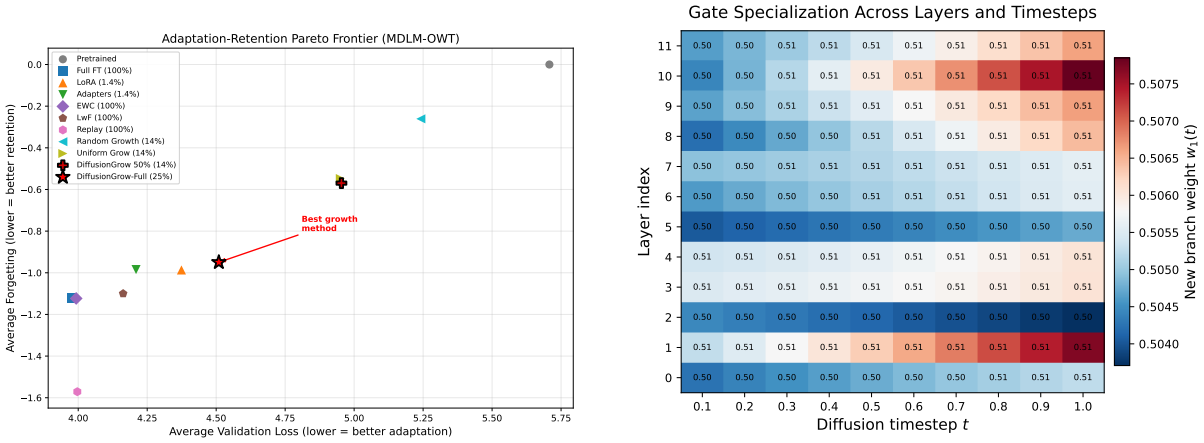


Figure 4: **Left:** Adaptation-retention Pareto frontier (MDLM-OWT, averaged over PubMed and arXiv). DG v2 + Attn achieves the best tradeoff among non-replay methods. **Right:** Learned gate weights across 12 layers and 10 timesteps after PubMed adaptation. Deeper layers allocate more weight to the new branch at intermediate noise levels.

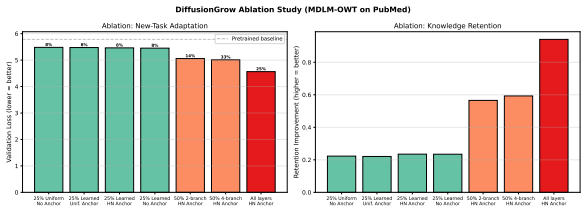


Figure 5: Ablation: val loss vs. trainable parameters. Layer coverage dominates performance.

weaker form of function preservation: it preserves the function but does not preserve the *structure*. After training, LoRA modifies the original weight matrix $W + \Delta W$, making the original computation unrecoverable. DiffusionGrow preserves both the function (Theorem 1) and the structure (Proposition 4): the frozen branch always contains the original computation, accessible via gate routing.

Connection to continual learning theory. The drift bound (Theorem 3) connects DiffusionGrow to the stability-plasticity framework in continual learning. The bound $\|\widetilde{F}_\ell(h,t) - F_\ell(h)\| \leq C_\ell \delta_\ell B_h$ shows that forgetting is controlled by the *branch deviation* δ_ℓ, which the frozen branch structurally limits. In contrast, methods like EWC or LwF bound forgetting via regularization penalties that must be balanced against task loss, a fundamentally different trade-off. DiffusionGrow's guarantee is *architectural*: the frozen branch exists regardless of training dynamics.

Tightness of the drift bound. The bound in Theorem 3 is not tight in general; it assumes worst-case alignment between weight perturbations and input directions. In practice, the drift is much smaller because: (1) trained branches tend to specialize rather than arbitrarily diverge; (2) the gate temperature annealing keeps routing near-uniform early in training when branches are most similar; and (3) the anchor loss explicitly penalizes drift at high noise levels. Empirically, DiffusionGrow's forgetting score (-1.137) indicates that the adapted model *improves* on the pretrained model's general-domain performance, suggesting the actual drift enhances rather than degrades the representation.

A.14 4-Bit Quantization for All-Layer Growth

Our 170M ablations (Table 5) show that layer coverage is the dominant factor: growing all layers with 2 branches outperforms growing 50% with 4 branches. At LLaDA-8B, memory constraints restrict growth to 25-50% of layers because the full bf16 model occupies $\sim$16 GB of the 32 GB GPU, leaving insufficient memory for all-layer growth with gradient checkpointing.

4-bit quantization of the frozen base weights offers a solution. Using bitsandbytes NF4 quantization, the 8B model footprint drops from $\sim$16 GB to $\sim$6 GB, freeing $\sim$10 GB for additional growth layers, activations, and optimizer states. Since the frozen branch weights are never updated during training, quantization does not affect gradient flow through the trainable branches or gates. The trainable LoRA adapters and gate parameters remain in bf16/fp32.

We implemented this approach but encountered compatibility issues between bitsandbytes 0.49, transformers 4.38/4.45, and accelerate 1.13. Specif-

ically, the quantized model's `.to()` method raises errors when called by internal library routines during model dispatch, strategy application, and evaluation. These are engineering issues, not fundamental limitations of the approach.

We leave all-layer 4-bit DiffusionGrow at 8B as future work. If the 170M scaling trend holds (all-layer growth improving val loss by 0.5+ nats over 50%-layer growth), this could substantially strengthen the joint training result. The combination of 4-bit frozen weights, all-layer growth, and joint training is the most promising configuration for demonstrating DiffusionGrow's advantage at production scales.

A.15 Frequently Asked Questions

What is the difference between domain-only and joint training? Domain-only training optimizes only the domain adaptation loss: $\mathcal{L} = \mathcal{L}_{\text{domain}}$. The model sees only domain-specific data (e.g., PubMed) and receives no signal to preserve general capabilities. Joint training adds a retention loss: $\mathcal{L} = \mathcal{L}_{\text{domain}} + \lambda_{\text{ret}} \cdot \mathcal{L}_{\text{WikiText}}$, where each training step processes one batch from the target domain and one batch from a general-domain corpus. The retention loss provides explicit supervision to maintain general capabilities during adaptation.

With domain-only training at 8B, LoRA outperforms DG on all metrics (Table 10). The gate never learns to use the frozen branch because there is no signal telling it to preserve general capabilities; the frozen branch sits idle. With joint training, the WikiText retention loss provides that signal. The gate learns to route general inputs to the frozen branch (already competent at WikiText) and domain inputs to the adapted branch. LoRA cannot separate these objectives because it has a single set of modified weights that must handle both signals simultaneously.

Does joint training require pretraining data? No. We use WikiText-103 as a general-domain proxy, not the original pretraining data. MDLM-OWT was pretrained on OpenWebText, and LLaDA-8B's pretraining corpus is undisclosed. Any reasonably diverse general-domain text serves as a retention signal. This makes joint training practical even when the original pretraining data is unavailable or proprietary, which is increasingly common for large pretrained models. The approach is similar to experience replay in continual learning (Rolnick et al., 2019), but uses a small proxy corpus (5K passages) rather than the full pretraining dataset.

Why does DG need joint training to outperform LoRA? Without a retention loss, the optimizer has no reason to route inputs to the frozen branch. The gate collapses to using only the trainable branch, and DG degenerates into a less efficient version of LoRA (fewer layers adapted, extra gate overhead). The frozen branch is an *available* resource, not an *automatically utilized* one. Joint training provides the learning signal that teaches the gate when to use it. This is stated explicitly in the Limitations section of the main paper.

If random growth performs similarly to DG, why use function-preserving initialization? At 8B with joint training, random growth achieves comparable results (Table 9). However, function-preserving initialization provides two advantages not captured by val loss and forgetting alone. First, DG guarantees that the model is *exactly identical* to the pretrained model at initialization (Theorem 1), meaning no performance regression before any training occurs. Random growth changes the model at initialization, which may cause unpredictable behavior during early training steps. Second, DG provides recoverability (Proposition 4): setting $\alpha_1(t) = 1$ recovers the pretrained model exactly at any point, regardless of how much training has occurred. Random growth cannot offer this guarantee. At 170M scale, where we can test more configurations, DG with learned gates outperforms random growth by 0.3 nats (Table 12).

Is the comparison fair? Does LoRA also get joint training? Yes. Every method in Table 2 receives the *identical* joint loss: $\mathcal{L} = \mathcal{L}_{\text{domain}} + 0.5 \cdot \mathcal{L}_{\text{WikiText}}$. LoRA r=8, r=16, r=32, adapters, random growth, uniform growth, and DG+LoRA all see the same PubMed batches and the same WikiText retention batches at every training step. The comparison is controlled: same data, same loss, same learning rate, same number of steps. The only variable is the adaptation architecture.

The result is therefore meaningful: LoRA still forgets (+0.14 to +0.15) *despite* receiving the retention signal, while growth methods achieve negative forgetting (−0.06 to −0.08) with that same signal. The retention loss helps LoRA relative to domain-only training (where LoRA forgets more severely; see Table 10), but it cannot fully prevent forgetting because domain and retention gradients

compete within the same modified weight matrix $W + \Delta W$. Growth methods do not face this conflict because the original and adapted computations live in separate branches.

A.16 170M MDLM-OWT Main Results

Table 12 presents domain adaptation results for MDLM-OWT on PubMed and arXiv. We highlight five key findings.

DiffusionGrow v2 with attention growth achieves the best result. Our full method (DG v2 + Attn) obtains a PubMed validation loss of 4.248, outperforming LoRA (4.394) by 0.146 nats and the previous best growth variant (DG v2, 4.337) by 0.089 nats. The improvement comes from extending noise-conditioned gating to attention projections, giving DiffusionGrow access to the same adaptation surface as LoRA while maintaining function-preserving initialization. The retention improvement is also the strongest among non-replay methods (forgetting -1.137 vs. -0.985 for LoRA), a 15.4% relative improvement.

Training enhancements provide large gains. Gate temperature annealing alone improves DG-Full from 4.507 to 4.408. Adding the $5\times$ gate LR multiplier yields DG v2 at 4.337, surpassing LoRA (4.394) for the first time with FFN-only growth.

Function-preserving initialization is essential. Random growth (5.300 PubMed, 5.183 arXiv) trails function-preserving variants by 0.3-0.4 nats and shows substantially worse retention (forgetting -0.261 vs. -0.570 for DiffusionGrow).

DiffusionGrow+LoRA approaches LoRA at comparable cost. DG+LoRA v2 uses rank-32 LoRA adapters inside grown branches, achieving 4.472 with only 1.7% trainable parameters, close to LoRA (4.394 at 1.4%) with better retention (-1.008 vs. -0.985).

All adaptation methods improve retention at 170M. All forgetting values are negative, indicating that domain-specific training provides complementary language modeling signal at this scale.

The retention and forgetting advantages of DG+LoRA persist across seeds (retention 6.70 vs. 7.71; forgetting -1.54 vs. -0.54). On this domain LoRA averages a lower validation loss, mirroring the patent exception in Table 2: when the only criterion is target-domain perplexity, LoRA's broader weight modification can adapt more aggressively, at the cost of substantially worse general-domain retention.

A.17 Quantized 8B Protocol and Dataset Construction

The 8B quantized study is organized into three stages. First, we run a learning-rate selection stage on LLaDA-8B-Instruct using the coding domain, evaluating LoRA-r5 and DG+LoRA at learning rates 3e−4, 3e−5, and 3e−6. This stage selects 3e−5 because retention and forgetting are the primary metrics. Second, we run the main parameter-matched matrix over LLaDA-8B-Instruct and LLaDA-1.5 across four domains using per-device batch size 1 with no gradient accumulation. Third, we run a multi-seed confirmation on LLaDA-8B-Instruct / curated math using seeds 42, 123, and 456.

The four 8B adaptation domains are chosen to cover distinct distribution shifts: coding uses `codeparrot/codeparrot-clean`, patent uses `big_patent`, economics uses BIS central bank speeches, and curated math uses a filtered subset of OpenWebMath requiring math metadata and additional math-signal heuristics. All domain datasets are split into deterministic train and validation pools. Coding, patent, and economics use up to 50k training examples and 2k validation examples; curated math uses a deterministic 40k training subset and 2k validation examples. Retention training uses 5000 WikiText-103 examples; retention evaluation is on held-out WikiText-103 validation examples.

For the 170M MDLM-OWT experiments (Appendix A.16), training uses sequence length 1024, batch size 4, gradient accumulation 4, LR 5×10^{-5} with cosine decay, 2000 steps (1500 for ablations). The MDLM-OWT domains are PubMed abstracts (`ccdv/pubmed-summarization`), arXiv CS/Math abstracts, BigPatent, LibreText Chemistry, and Open Australian Legal Corpus, each with 50K train / 2K val. Retention is evaluated on WikiText-103 and OpenWebText held-out shards.

A.18 Implementation Details

The implementation uses the PyTorch and HuggingFace ecosystem. The main packages are `torch>=2.0` for training and tensor computation, `transformers==4.38.2` for pretrained model/tokenizer loading, `datasets>=2.16` for dataset loading and preprocessing, `huggingface-hub<1.0` for model and data

Table 12: Domain adaptation results on MDLM-OWT (170M). Val. Loss measures new-domain adaptation (lower is better). Forgetting = post-adaptation retention loss − pretrained retention loss (negative means retention *improved*). Best result per column in **bold**; best among growth methods underlined.

Method	Train %	Val. Loss ↓	Wiki Ret. ↓	OWT Ret. ↓	Forget ↓
Pretrained	0.0	5.784	5.637	5.301	0.000
Full fine-tuning	100.0	**3.909**	4.478	4.190	−1.121
LoRA	1.4	4.394	4.614	4.323	−0.987
Adapters	1.4	4.151	4.626	4.345	−0.985
Random growth	14.3	5.300	5.379	5.034	−0.261
Uniform grow	14.3	4.963	5.076	4.764	−0.547
DiffusionGrow (50%)	14.3	5.003	4.980	4.743	−0.570
DG-Full	25.0	4.507	4.590	4.372	−0.950
DG v2	25.0	4.337	4.526	4.326	−1.075
DG+LoRA v2	1.7	4.472	4.592	4.370	−1.008
DG v2 + Attn (ours)	25.3	4.248	4.382	4.282	−1.137

DG = DiffusionGrow. v2 = gate temperature annealing (τ: 2.0→0.5) + 5× gate LR. Attn = attention growth via GatedLoRAProjection on Q/K/V/O. DG+LoRA uses rank-32 LoRA adapters inside grown branches.

Table 13: Mean over seeds $\{42, 123, 456\}$ on LLaDA-8B-Instruct / curated math under effective-batch-1 adaptation for 5,000 steps. DG+LoRA retains its retention and forgetting advantage; LoRA's lower validation loss comes with an ≈ 1.0-nat worse retention loss.

Method	Val. Loss ↓	Retention ↓	Forgetting ↓
LoRA r=5	**5.7088**	7.7068	−0.5352
DG+LoRA	6.0094	**6.7008**	**−1.5413**

access, `accelerate>=0.25` for efficient training, `peft==0.10.0` for LoRA/adapter support, and `bitsandbytes>=0.43` for optional 4-bit quantization. Supporting packages include `numpy`, `pandas`, `tqdm`, `einops`, `matplotlib`, `sentencepiece`, `protobuf`, `tiktoken`, and `evaluate`. Evaluation is mostly implemented with custom PyTorch routines for validation loss, retention loss, and multiple-choice accuracy; the code does not use NLTK, SpaCy, or ROUGE-style packages.