Asymmetric Flow Models

Hansheng Chen **Jan Ackermann** **Minseo Kim** **Gordon Wetzstein** **Leonidas Guibas**
Stanford University
https://hanshengchen.com/asymflow

Figure 1: **AsymFLUX.2 klein generations.** AsymFlow finetunes FLUX.2 klein into a pixel-space flow model, producing highly realistic images with rich visual styles and fine detail.

Abstract

Flow-based generation in high-dimensional pixel spaces is difficult because velocity prediction requires modeling high-dimensional noise, even when data has strong low-rank structure. We present *Asymmetric Flow Modeling* (AsymFlow), a rank-asymmetric velocity parameterization that restricts noise prediction to a low-rank subspace while keeping data prediction full-dimensional. From this asymmetric prediction, AsymFlow analytically recovers the full-dimensional velocity without changing the network architecture or training/sampling procedures. On ImageNet 256×256, AsymFlow achieves a leading 1.57 FID, outperforming prior DiT/JiT-like pixel diffusion models by a large margin. AsymFlow also provides the first-ever route for finetuning pretrained latent flow models into pixel-space models: aligning the low-rank pixel subspace to the latent space gives a seamless initialization that preserves the latent model's high-level semantics and structure, so finetuning mainly improves low-level mismatches rather than relearning pixel generation. We show that the pixel AsymFlow model finetuned from FLUX.2 klein 9B establishes a new state of the art for pixel-space text-to-image generation, beating its latent base on HPSv3, DPG-Bench, and GenEval while qualitatively showing substantially improved visual realism.

Preprint.

1 Introduction

Recent progress in diffusion-based image and video generation [5, 6, 18, 32, 63, 72] has been driven by combining scalable transformer architectures [7, 15, 48] with flow matching objectives [1, 40, 42]. Most state-of-the-art systems operate in compressed lower-dimensional latent spaces learned by autoencoders [51], which is highly scalable but delegates fine detail to a fixed decoder that the generative model cannot control. This limitation motivates a return to high-dimensional generation, including direct pixel-space generation [2, 9, 10, 27, 35, 45, 46, 64, 71].

However, moving to high-dimensional spaces exposes a bottleneck in velocity prediction. The velocity target $\boldsymbol{u} = \boldsymbol{\epsilon} - \boldsymbol{x}_0$ consists of both data and noise components. To predict it accurately, the network must extract the noise from the input and pass it through its internal features. This is straightforward in latent spaces, where the noise dimension is small relative to the network width. In pixel space, however, the per-patch noise dimension can pollute the network's internal states, creating a bottleneck [75]. Classical pixel diffusion models used U-Net architectures [14, 21, 28, 52, 54], whose skip connections naturally route noise from input to output. Modern scalable transformers lack these pathways, so recent methods either reintroduce architectural bypasses, such as U-ViT-like transformers [4, 11, 17, 22, 23] or decoder heads [10, 45, 62, 64, 71, 75], which complicates the otherwise simple transformer recipe, or switch to predicting clean data $\boldsymbol{x}_0$ directly [35, 46, 58], which is numerically ill-conditioned at low noise levels [28, 55].

We introduce *Asymmetric Flow Modeling* (AsymFlow), a new parameterization for high-dimensional flow modeling that avoids both of these compromises. AsymFlow parameterizes the two velocity components asymmetrically: the data component remains full-dimensional, while the noise component is restricted to a low-rank subspace. The full-dimensional velocity is recovered analytically, so standard flow matching training and sampling remain unchanged. In this view, standard $\boldsymbol{x}_0$-prediction and $\boldsymbol{u}$-prediction are special cases of AsymFlow, corresponding to zero and full rank of this noise subspace, respectively. Between these endpoints, AsymFlow can choose an intermediate rank that keeps velocity prediction in an important subspace while avoiding full-rank noise prediction.

In addition, AsymFlow makes it possible to build large-scale pixel generators by finetuning pretrained latent flow models. The key observation is that latent and pixel spaces are not disconnected: a latent model can be mathematically lifted into a low-rank pixel model whose samples inherit the semantics and structure of the latent generator. This turns latent-to-pixel adaptation into a correction problem, where finetuning keeps the high-level content and only needs to close the low-level projection gap between low-rank pixel outputs and full-rank pixel targets. To our knowledge, this is the first practical path for turning existing large-scale latent flow models themselves into strong pixel generators.

We evaluate AsymFlow in two settings. On ImageNet 256×256 [12], AsymFlow reaches 1.76 FID with the JiT-H/16 network [35] and 1.57 FID with an additional REPA loss [70], outperforming prior DiT/JiT-like pixel diffusion models by a large margin. For text-to-image generation, our pixel AsymFlow model finetuned from FLUX.2 klein 9B [6] sets a new state of the art in pixel-space generation, beating its latent base on HPSv3 [44], DPG-Bench [25], and GenEval [16] while qualitatively exhibiting substantially improved visual realism.

To summarize, our main contributions are:

- We introduce AsymFlow, a novel rank-asymmetric flow parameterization with full-rank data and low-rank noise for scalable high-dimensional generation.
- We provide the first method of finetuning pretrained latent flow models into pixel models through AsymFlow, using a principled latent-to-pixel lift without architectural modifications.
- We achieve a leading 1.57 FID on ImageNet 256×256 and demonstrate a 9B-scale pixel-space text-to-image model with state-of-the-art performance.

2 Related Work

Recent work mainly addresses the high-dimensional bottleneck in two ways: changing the network architecture so high-dimensional noisy inputs can reach the output more easily, or changing the prediction parameterization to avoid high-dimensional noise prediction.

Hierarchical architectures. One line of work keeps noise or velocity prediction feasible using hierarchical architectures with high-dimensional bypasses. Classical DDPM/ADM-style U-Nets [14,

21, 52] and U-ViT-like hierarchical transformers [4, 11, 17, 22, 23] use skip-connected multi-scale structures, while DDT-like decoder-based designs [65], including RAE, PixNerd, PixelDiT, DiP, and DeCo [10, 45, 62, 64, 71, 75], expose the noisy input to decoder or refiner pathways conditioned on backbone features. These designs are effective, but they complicate the plain transformer recipe that has scaled successfully in large image and video generators [5, 6, 18, 32, 63, 72]. In contrast, AsymFlow enables high-dimensional generation without architectural modification, making it possible to finetune large-scale latent flow models into pixel space for the first time.

Prediction parameterizations. In early diffusion models, hierarchical U-Net-like architectures made $\boldsymbol{\epsilon}$-prediction practical, while $\boldsymbol{x}_0$-prediction was often less favored because of low-noise numerical issues [21, 28, 55]. With the paradigm shift to plain diffusion transformers (DiT) [43, 48, 69], JiT [35] argues that pixel diffusion should predict clean data $\boldsymbol{x}_0$ rather than noise or velocity, and several follow-up pixel methods [46, 58] adopt the same $\boldsymbol{x}_0$-prediction backbone with perceptual or representation-alignment (REPA) losses [70, 73]. k-Diff [27] learns a scalar interpolation between $\boldsymbol{x}_0$- and $\boldsymbol{u}$-prediction, but this isotropic parameterization does not reduce the dimensionality of the noise component and gives results close to JiT. Unlike prior work, AsymFlow treats the prediction target asymmetrically: the data term $\boldsymbol{x}_0$ remains full-dimensional, while the noise term $\boldsymbol{\epsilon}$ is restricted to a low-rank subspace, which retains the benefits of $\boldsymbol{u}$-prediction in a meaningful subspace.

3 Preliminaries

We briefly introduce diffusion models [21, 59, 60] using the flow matching convention [1, 40, 42], then review common prediction parameterizations.

Flow matching. Let $\boldsymbol{x}_0 \in \mathbb{R}^D$ be a data vector of dimension D. A typical flow model defines an interpolation between a data sample and Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, yielding the noisy sample $\boldsymbol{x}_t \coloneqq \alpha_t \boldsymbol{x}_0 + \sigma_t \boldsymbol{\epsilon}$, where $t \in (0, 1]$ denotes diffusion time and $\alpha_t = 1 - t$, $\sigma_t = t$ define the linear flow schedule. Under this construction, generative modeling is achieved by solving a reverse-time SDE or ODE that transports noise to data [41, 61]. In particular, the ODE velocity is given by $\frac{\mathrm{d}\boldsymbol{x}_t}{\mathrm{d}t} = \mathbb{E}_{\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0|\boldsymbol{x}_t)}\left[\frac{\boldsymbol{x}_t - \boldsymbol{x}_0}{t}\right]$, which is the posterior mean of the sample velocity $\boldsymbol{u}$:

$$\boldsymbol{u} \coloneqq \frac{\boldsymbol{x}_t - \boldsymbol{x}_0}{\sigma_t} = \boldsymbol{\epsilon} - \boldsymbol{x}_0. \tag{1}$$

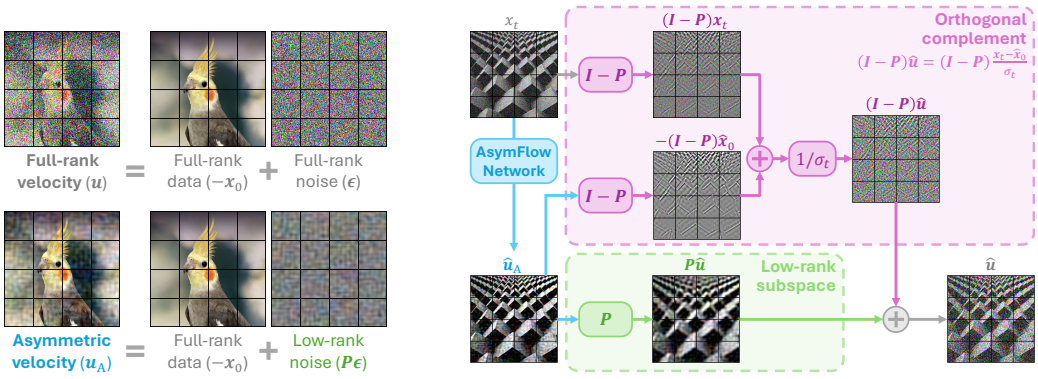
Then, a model $(\boldsymbol{x}_t, t) \mapsto \hat{\boldsymbol{u}}$ is trained to estimate this posterior mean with the flow matching loss:

$$\mathcal{L}_{\mathrm{FM}} = \mathbb{E}_{t, \boldsymbol{x}_0, \boldsymbol{\epsilon}}\left[\left\|\boldsymbol{u} - \hat{\boldsymbol{u}}\right\|^2\right]. \tag{2}$$

$\boldsymbol{u}$-prediction vs. $\boldsymbol{x}_0$-prediction. The mapping $(\boldsymbol{x}_t, t) \mapsto \hat{\boldsymbol{u}}$ is often directly parameterized by a neural network, i.e., $\hat{\boldsymbol{u}} \coloneqq G_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$. This $\boldsymbol{u}$-prediction form is widely used in modern latent flow models [15, 48, 51], where the representation is compressed. When moved to pixels or other high-dimensional representations, however, the target $\boldsymbol{u} = \boldsymbol{\epsilon} - \boldsymbol{x}_0$ requires predicting a high-dimensional noise component in addition to structured data [35, 75]. An alternative is $\boldsymbol{x}_0$-prediction, where the network predicts clean data $\hat{\boldsymbol{x}}_0 = G_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ and recovers velocity as $\hat{\boldsymbol{u}} = (\boldsymbol{x}_t - \hat{\boldsymbol{x}}_0)/\sigma_t$. This avoids directly regressing Gaussian noise [35], but the $1/\sigma_t$ conversion is ill-conditioned at low noise levels [28, 55], limiting final-sample quality. Shin et al. [58] also claim that REPA-style alignment is less effective in $\boldsymbol{x}_0$-prediction pixel models. Thus, $\boldsymbol{u}$- and $\boldsymbol{x}_0$-prediction expose complementary trade-offs where neither is ideal for high-dimensional generation.

4 Asymmetric Flow Modeling

To address the challenges of high-dimensional flow modeling, we introduce AsymFlow, a rank-asymmetric parameterization of the flow target. The key idea is to treat the two terms in the velocity target asymmetrically: the data prediction term remains full-dimensional, while the noise prediction is restricted to a low-rank subspace. This reduces the burden of representing high-dimensional noise in the network's internal states without changing the network architecture. The full-rank velocity is then recovered analytically for training and sampling, leaving the flow matching formulation unchanged.



(a) Full-rank flow vs AsymFlow parameterization **(b) Converting asymmetric velocity to full-rank velocity**

Figure 2: **AsymFlow parameterization and recovery.** (a) AsymFlow changes the standard velocity target by keeping the data term full-dimensional while replacing the noise term with its low-rank projection $\boldsymbol{P\epsilon}$. (b) To recover the full-rank velocity, the low-rank component $\boldsymbol{P}\hat{\boldsymbol{u}}_{\mathrm{A}}$ is used directly, while the orthogonal component is converted using the $\boldsymbol{x}_0$-to-$\boldsymbol{u}$ relation in Eq. (1).

4.1 AsymFlow Parameterization

Let $\boldsymbol{A} \in \mathbb{R}^{D\times r}$ be an orthonormal basis of a rank-r subspace, with $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = \boldsymbol{I}_r$, and let $\boldsymbol{P} \coloneqq \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}$ be the corresponding orthogonal projector. Then $\mathrm{Im}(\boldsymbol{P})$ is the low-rank subspace and $\mathrm{Im}(\boldsymbol{I}-\boldsymbol{P})$ is its orthogonal complement. Given the noise $\boldsymbol{\epsilon} \in \mathbb{R}^D$, we use $\boldsymbol{P\epsilon}$ to denote its subspace component. We refer to $\boldsymbol{P\epsilon}$ as *low-rank noise*, meaning Gaussian noise projected to a low-rank subspace.

AsymFlow changes the target that the network is asked to predict. In standard $\boldsymbol{u}$-prediction (Eq. (1)), the output must reproduce the full noise component $\boldsymbol{\epsilon}$ together with the data term $-\boldsymbol{x}_0$. For high-dimensional data, this forces the model to carry high-dimensional noise through its features, which pollutes its internal states and wastes network capacity. To address this issue, AsymFlow introduces an *asymmetric velocity* $\boldsymbol{u}_{\mathrm{A}}$ where the noise term is low-rank while the data term remains full-rank:

$$\boldsymbol{u}_{\mathrm{A}} \coloneqq \boldsymbol{P\epsilon} - \boldsymbol{x}_0. \tag{3}$$

We then train the network to predict the asymmetric velocity, i.e., $\hat{\boldsymbol{u}}_{\mathrm{A}} = G_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$. This prediction will be converted back to the full-rank velocity $\hat{\boldsymbol{u}}$ for loss calculation and denoising sampling (Sec. 4.2).

Fig. 2 (a) illustrates the visual difference between the full-rank velocity $\boldsymbol{u}$ and the asymmetric velocity $\boldsymbol{u}_{\mathrm{A}}$. Full-rank velocity is perturbed by dense noise, making it highly unpredictable. In contrast, AsymFlow keeps the structured data term full-dimensional but restricts only the stochastic noise term to a low-rank subspace. Since image data itself concentrates near a low-dimensional manifold, this makes the overall asymmetric target more predictable for neural networks.

Patch-wise low-rank projection. Following the patch-token representation of DiTs [48], we apply low-rank projection independently within each image patch. Concretely, for a patch dimension D and rank $r < D$, the matrix $\boldsymbol{A} \in \mathbb{R}^{D\times r}$ defines a low-rank subspace for each patch token, and the same projector $\boldsymbol{P} = \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}$ is shared across all tokens. Thus, AsymFlow reduces the noise prediction dimension within each patch while preserving the full set of image tokens.

Choosing the low-rank subspace. When training AsymFlow from scratch, $\boldsymbol{A}$ can be obtained from a data-dependent patch basis, e.g., by applying PCA to image patches. When adapting a pretrained latent model, $\boldsymbol{A}$ is instead chosen to align the latent space with the pixel patch space, which we compute by a Procrustes alignment between latent variables and their corresponding pixel patches. This latter construction enables a seamless latent-to-pixel initialization, and is discussed in Sec. 5.

4.2 Orthogonal Component View and Full-Rank Velocity Recovery

The asymmetric velocity in Eq. (3) has a simple interpretation after decomposing it into the low-rank subspace $\mathrm{Im}(\boldsymbol{P})$ and its orthogonal complement $\mathrm{Im}(\boldsymbol{I}-\boldsymbol{P})$:

$$\boldsymbol{P}\boldsymbol{u}_{\mathrm{A}} = \boldsymbol{P\epsilon} - \boldsymbol{P}\boldsymbol{x}_0 = \boldsymbol{P}\boldsymbol{u}, \qquad (\boldsymbol{I}-\boldsymbol{P})\boldsymbol{u}_{\mathrm{A}} = -(\boldsymbol{I}-\boldsymbol{P})\boldsymbol{x}_0. \tag{4}$$

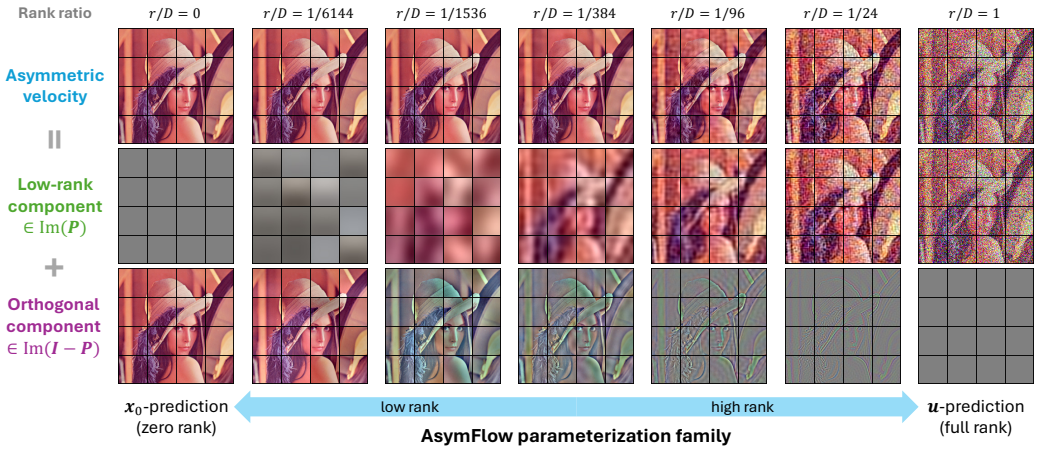


Figure 3: **Orthogonal component view of AsymFlow**. AsymFlow parameterization can be decomposed into a $\boldsymbol{P}\boldsymbol{u}$ component in the low-rank subspace $\mathrm{Im}(\boldsymbol{P})$ and an $(\boldsymbol{I}-\boldsymbol{P})\boldsymbol{x}_0$ component in the orthogonal complement $\mathrm{Im}(\boldsymbol{I}-\boldsymbol{P})$. Varying the rank r yields a parameterization family whose endpoints recover full $\boldsymbol{x}_0$-prediction and full $\boldsymbol{u}$-prediction.

The decomposition reveals that AsymFlow behaves like $\boldsymbol{u}$-prediction in the low-rank subspace and like $\boldsymbol{x}_0$-prediction in the orthogonal complement. Adjusting the rank r creates a family of parameterizations between the two endpoints, as shown in Fig. 3: when $r=0$, the target reduces to full $\boldsymbol{x}_0$-prediction up to sign; when $r=D$, AsymFlow recovers full $\boldsymbol{u}$-prediction. We expect a small but nonzero rank r to be optimal: it retains the benefit of $\boldsymbol{u}$-prediction for controlling the flow on a low-dimensional subspace, while avoiding the burden of predicting full-rank noise.

This component view also provides the conversion back to the full-rank velocity. We keep the low-rank velocity component $\boldsymbol{P}\boldsymbol{u}_{\mathrm{A}}$, and convert the orthogonal $\boldsymbol{x}_0$-style component to velocity using the $\boldsymbol{x}_0$-to-$\boldsymbol{u}$ relation established in Eq. (1):

$$\boldsymbol{u} = \boldsymbol{P}\boldsymbol{u}_{\mathrm{A}} + (\boldsymbol{I}-\boldsymbol{P})\frac{\boldsymbol{x}_t + \boldsymbol{u}_{\mathrm{A}}}{\sigma_t}. \tag{5}$$

In practice, we apply the conversion to the network prediction $\hat{\boldsymbol{u}}_{\mathrm{A}}$ to obtain $\hat{\boldsymbol{u}}$, which is used in the flow matching loss (Eq. (2)) and denoising sampling. Fig. 2 (b) illustrates this conversion visually.

5 Finetuning Latent Flow into Pixel AsymFlow

A key advantage of AsymFlow is that it provides a direct way to turn pretrained $\boldsymbol{u}$-predicting latent flow models into pixel-space generators. We first lift a pretrained latent model into an equivalent low-rank pixel flow at initialization, with exact input and output conversions between latents and low-rank pixels. Solving this lifted pixel flow ODE preserves the latent trajectory up to an analytically determined orthogonal noise component, so the initialized model generates lifted low-rank pixels whose semantics and structure match the pretrained latent model. Finetuning then focuses on correcting the low-level projection gap between these low-rank pixels and the full-rank pixel targets.

5.1 Latent-to-Pixel Initialization

We consider a latent flow model $\hat{\boldsymbol{u}}_{\boldsymbol{z}} = G_{\boldsymbol{\phi}}(\boldsymbol{z}_t, t)$ pretrained on latent tokens $\boldsymbol{z}_0 \in \mathbb{R}^d$ with velocity $\boldsymbol{u}_{\boldsymbol{z}} \coloneqq \boldsymbol{\epsilon}_{\boldsymbol{z}} - \boldsymbol{z}_0$. To bridge the latent-to-pixel gap, we construct a patch-wise linear lift $\boldsymbol{A} \in \mathbb{R}^{D\times d}$ from latent space to pixel space using Procrustes alignment (details in Appendix A.1), such that the lifted low-rank pixels $\boldsymbol{x}_0^{\mathrm{L}} \coloneqq \boldsymbol{A}\boldsymbol{z}_0$ approximate the full-rank pixels $\boldsymbol{x}_0$. Consider the corresponding pixel-space forward process $\boldsymbol{x}_t^{\mathrm{L}} \coloneqq \alpha_t \boldsymbol{x}_0^{\mathrm{L}} + \sigma_t \boldsymbol{\epsilon}$ and velocity $\boldsymbol{u}^{\mathrm{L}} \coloneqq \boldsymbol{\epsilon} - \boldsymbol{x}_0^{\mathrm{L}}$. Then the latent and pixel quantities are related by exact input and output conversions:

$$\text{input:}\quad \boldsymbol{z}_t = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{x}_t^{\mathrm{L}}, \qquad \text{output:}\quad \boldsymbol{u}^{\mathrm{L}} = \boldsymbol{P}\boldsymbol{A}\boldsymbol{u}_{\boldsymbol{z}} + (\boldsymbol{I}-\boldsymbol{P})\frac{\boldsymbol{x}_t^{\mathrm{L}} + \boldsymbol{A}\boldsymbol{u}_{\boldsymbol{z}}}{\sigma_t}. \tag{6}$$

The input identity shows that noisy low-rank pixels can be projected to noisy latents by $\boldsymbol{A}^{\mathrm{T}}$, while the output identity converts the lifted latent velocity $\boldsymbol{A}\boldsymbol{u}_{\boldsymbol{z}}$ back to the low-rank pixel velocity using the same recovery rule as AsymFlow in Eq. (5). These identities imply trajectory coupling of the lifted pixel and latent ODEs (Theorem 1). Therefore, a d-dimensional latent $\boldsymbol{u}$-prediction model can be reinterpreted as an exact rank-d pixel flow model with the network $\boldsymbol{A}G_{\boldsymbol{\phi}}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{x}_t^{\mathrm{L}}, t)$. In implementation, the projections $\boldsymbol{A}^{\mathrm{T}}$ and $\boldsymbol{A}$ are fused into the learnable input and output linear layers of $G_{\boldsymbol{\phi}}$, yielding the initialized pixel AsymFlow model $\hat{\boldsymbol{u}}_{\mathrm{A}} = G_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ for later finetuning.

Initialization property. The initialized low-rank pixel model predicts a target of the form $\boldsymbol{P}\boldsymbol{\epsilon} - \boldsymbol{x}_0^{\mathrm{L}}$, so its gap to the AsymFlow target $\boldsymbol{u}_{\mathrm{A}}$ (Eq. (3)) is only the approximation gap $\boldsymbol{x}_0 - \boldsymbol{x}_0^{\mathrm{L}}$. Due to the trajectory coupling (Theorem 1), sampling the initialized model generates $\boldsymbol{x}_0^{\mathrm{L}}$-like lifted low-rank pixel samples without accumulating additional trajectory errors. These samples are semantically and structurally aligned with the $\boldsymbol{x}_0$-like decoded latent samples, so the gap $\boldsymbol{x}_0 - \boldsymbol{x}_0^{\mathrm{L}}$ is mainly low-level and easy to correct during finetuning, as shown in Fig. 4.

Figure 4: **Latent-to-pixel initialization.** The lifted low-rank pixel generation are semantically and structurally aligned with the decoded latent generation, leaving only a low-level gap to correct.

Scale calibration. A good initialization requires the scale of the lifted pixels $\boldsymbol{x}_0^{\mathrm{L}}$ to align with the scale of real pixels $\boldsymbol{x}_0$. However, under the orthonormality constraint $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = \boldsymbol{I}$, Procrustes alignment matches directions but not scale. We therefore introduce a scale factor s and use the scale-calibrated lift $\boldsymbol{x}_0^{\mathrm{L}} = s\boldsymbol{A}\boldsymbol{z}_0$. In implementation, this scale correction is folded into the model input, output, and internal timestep calibration, as detailed in Appendix A.2.

5.2 Variance-Reduced Finetuning Loss

The initialization above reduces latent-to-pixel finetuning to correcting the paired low-level gap $\boldsymbol{x}_0 - \boldsymbol{x}_0^{\mathrm{L}}$. While the standard flow matching loss (Eq. (2)) regressing to $\boldsymbol{x}_0$ already provides a valid objective, the paired low-rank target $\boldsymbol{x}_0^{\mathrm{L}}$ offers additional structure that can be used for variance reduction using control variates, thereby improving convergence and sample quality [68].

To achieve this, we inject a term $-\lambda(\boldsymbol{x}_0^{\mathrm{L}} - \mathbb{E}[\boldsymbol{x}_0^{\mathrm{L}}|\boldsymbol{x}_t])$ into Eq. (2). This gives an equivalent flow matching loss whose variance is lower when $\|\boldsymbol{x}_0 - \boldsymbol{x}_0^{\mathrm{L}}\|$ is small. The conditional mean $\mathbb{E}[\boldsymbol{x}_0^{\mathrm{L}}|\boldsymbol{x}_t]$ can then be approximated by the prediction $\hat{\boldsymbol{x}}_0^{\mathrm{L}}$ of a frozen copy of the initialized low-rank model:

$$\mathbb{E}_{t,\boldsymbol{x}_0,\boldsymbol{\epsilon}}\left[\frac{\left\|\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0 - \lambda(\boldsymbol{x}_0^{\mathrm{L}} - \mathbb{E}[\boldsymbol{x}_0^{\mathrm{L}}|\boldsymbol{x}_t])\right\|^2}{\sigma_t^2}\right] \approx \mathbb{E}_{t,\boldsymbol{x}_0,\boldsymbol{\epsilon}}\left[\frac{\left\|\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0 - \lambda(\boldsymbol{x}_0^{\mathrm{L}} - \hat{\boldsymbol{x}}_0^{\mathrm{L}})\right\|^2}{\sigma_t^2}\right] =: \mathcal{L}_{\mathrm{VR}}. \tag{7}$$

Here, $\hat{\boldsymbol{x}}_0$ is predicted by the finetuned AsymFlow model from $\boldsymbol{x}_t$ (converted to the $\boldsymbol{x}_0$ format), and $\hat{\boldsymbol{x}}_0^{\mathrm{L}}$ is predicted by the frozen low-rank model from the paired noisy low-rank sample $\boldsymbol{x}_t^{\mathrm{L}} = \alpha_t\boldsymbol{x}_0^{\mathrm{L}} + \sigma_t\boldsymbol{\epsilon}$, diffused with the same noise as $\boldsymbol{x}_t$. The parameter λ is a patch-wise adaptive weight chosen to minimize the loss gradient norm, thereby reducing the variance of the effective target. In practice, this is implemented via an orthogonal projection and detailed in Appendix A.3. Empirically, the resulting variance-reduced objective $\mathcal{L}_{\mathrm{VR}}$ substantially improves fine-grained details in the generated results.

Perceptual correction. The approximation in Eq. (7) assumes $\mathbb{E}[\boldsymbol{x}_0^{\mathrm{L}}|\boldsymbol{x}_t] \approx \mathbb{E}[\boldsymbol{x}_0^{\mathrm{L}}|\boldsymbol{x}_t^{\mathrm{L}}]$, which is only exact if $\boldsymbol{x}_t - \boldsymbol{x}_t^{\mathrm{L}} \in \mathrm{Im}(\boldsymbol{I} - \boldsymbol{P})$. In practice, this condition is rarely strictly satisfied when $t < 1$, meaning the variance reduction term $\lambda(\boldsymbol{x}_0^{\mathrm{L}} - \hat{\boldsymbol{x}}_0^{\mathrm{L}})$ introduces a bounded approximation error inside the low-rank subspace $\mathrm{Im}(\boldsymbol{P})$. Empirically, this manifests as excessive noise in the generated results. To compensate, we add an LPIPS perceptual loss [46, 73] between $\boldsymbol{x}_0$ and $\hat{\boldsymbol{x}}_0$. This perceptual loss is gated by the same patch-wise weight λ, and we dynamically fade from the variance reduction term to the LPIPS loss across diffusion time. We defer the exact weighting schedule to Appendix A.4.

6 Experiments

We evaluate AsymFlow in two settings: ImageNet pixel models trained from scratch with the JiT-H/16 network, which isolate the parameterization itself, and large text-to-image models finetuned from the FLUX.2 klein latent generator, which test the finetuning approach and scalability of AsymFlow.

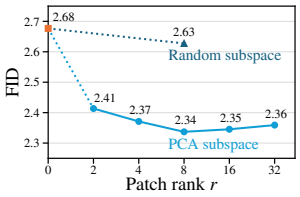


Figure 5: **Patch rank and PCA ablation**. 160 epochs.

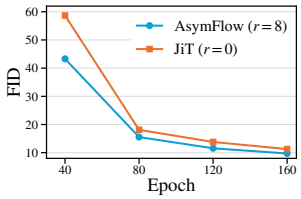


Figure 6: **Convergence speed comparison**. Unguided FIDs.

Table 1: **AsymFlow vs. JiT-H/16** and sensitivity to $\sigma_{\min}$ clamping. 600 epochs (final checkpoint).

Method	$\sigma_{\min}$	FID	IS
AsymFlow ($r = 8$)	0.04	1.76	312.0
	0.00	2.28	306.2
JiT ($r = 0$)	0.04	1.90	300.8
	0.00	3.27	286.7

6.1 Training from Scratch on ImageNet

We train class-conditional ImageNet 256×256 pixel models using the same setup as JiT-H/16 (see Table 9 in [35]), changing only the prediction parameterization. Unless otherwise stated, AsymFlow is trained using the flow matching loss (Eq. (2)) using a $D = 768$ patch-wise PCA subspace of rank r, with $r = 0$ exactly reproducing JiT's $\boldsymbol{x}_0$-prediction. Results use ADM evaluation [14, 19] with grid-searched guidance scales and intervals that optimize FID [20, 33]. We defer the details to Appendix B.

Comparison with JiT baseline. Table 1 compares AsymFlow ($r = 8$) and the official JiT checkpoint using ADM evaluation after 600 epochs. In practical sampling, the $\boldsymbol{x}_0$-to-$\boldsymbol{u}$ conversion in Eq. (1) clamps the denominator by $\sigma_{\min}$ to avoid numerical instability [35]. Since AsymFlow applies this conversion only in the orthogonal complement, it should be less sensitive to this clamp. The results confirm this: with the optimal $\sigma_{\min} = 0.04$ for both methods, AsymFlow improves over JiT in both FID and IS by a clear margin; disabling clamping degrades JiT by 1.37 FID, but AsymFlow by only 0.52. This shows that the asymmetric parameterization improves both overall quality and low-noise numerical stability.

Table 2: **ImageNet 256×256 pixel diffusion comparison**. FLOP estimation follows the convention in [71]. * denotes JiT evaluation protocol, which may have up to 0.08 better FID than ADM according to our tests.

Method	Pred (±)	Params	GFLOPs	FID↓
Hierarchical CNNs (skip connections / U-Net-like)				
ADM-G [14]	$\boldsymbol{\epsilon}$	554M	2240	4.59
Hierarchical transformers (skip connections / U-ViT-like)				
RIN [26]	$\boldsymbol{\epsilon}$	320M	668	3.42
SiD, UViT/2 [22]	$\boldsymbol{\epsilon}$	2B	1110	2.44
VDM++, UViT/2 [31]	$\boldsymbol{\epsilon}$	2B	1110	2.12
SiD2, UViT/2 [23]	$\boldsymbol{\epsilon}$	-	274	1.73
EPG-G/16 [34]	$\boldsymbol{x}_0$	1.4B	642	<u>1.58</u>
SiD2, UViT/1 [23]	$\boldsymbol{\epsilon}$	-	1306	**1.38**
Hierarchical transformers (decoder head / DDT-like)				
PixNerd-XL/16 [64]	$\boldsymbol{\epsilon} - \boldsymbol{x}_0$	700M	268	2.15
DiP-XL/16 [10]	$\boldsymbol{\epsilon} - \boldsymbol{x}_0$	631M	-	1.79
DeCo-XL/16 [45]	$\boldsymbol{\epsilon} - \boldsymbol{x}_0$	682M	245	<u>1.62</u>
PixelDiT-XL/16 [71]	$\boldsymbol{\epsilon} - \boldsymbol{x}_0$	797M	311	**1.61**
Plain transformers (DiT-like)				
PixelFlow-XL/4 [9]	$\boldsymbol{\epsilon} - \boldsymbol{x}_0$	677M	5818	1.98
JiT-H/16 [35]	$\boldsymbol{x}_0$	953M	363	1.86*
PixelGen-XL/16 [46]	$\boldsymbol{x}_0$	676M	260	1.83
JiT-G/16 [35]	$\boldsymbol{x}_0$	2B	766	1.82*
PixelREPA-H/16 [58]	$\boldsymbol{x}_0$	953M	363	<u>1.81</u>*
AsymFlow-H/16	$\boldsymbol{P\epsilon} - \boldsymbol{x}_0$	953M	363	**1.57**

Patch rank. Figure 5 studies the effect of the patch rank. Moving from JiT ($r = 0$) to AsymFlow sharply improves guided FID, with the best result at $r = 8$; increasing the rank further gives mild degradation. This matches the intended trade-off: AsymFlow keeps velocity prediction in a useful low-rank subspace while avoiding the burden of predicting high-dimensional noise.

PCA subspace. Figure 5 also compares PCA and random subspaces at $r = 8$. The random subspace performs close to the JiT baseline and far worse than PCA, showing that the gain comes from using a meaningful low-rank subspace, not merely reducing rank.

Convergence speed. Figure 6 compares FID during training. With the same architecture and recipe, AsymFlow ($r = 8$) consistently improves over JiT and reaches comparable FID roughly 40% faster. Thus, the rank-asymmetric target improves not only final quality but also optimization efficiency.

Comparison with prior pixel diffusion models. Table 2 compares AsymFlow ($r = 8$ plus a standard REPA loss [70]) with prior ImageNet 256×256 pixel diffusion models. With REPA, AsymFlow reaches 1.57 FID, establishing the state of the art among practical pixel diffusion models (excluding the much more expensive SiD2 UViT/1). In particular, AsymFlow outperforms previous plain-transformer models by a large margin (FID 1.57 vs. 1.81*). This result also shows that AsymFlow is strongly compatible with REPA: PixelREPA [58] reports that plain REPA is ineffective for larger JiT models, and its additional designs improve JiT-H/16 only from 1.86* to 1.81* FID; in contrast, adding plain REPA to AsymFlow improves FID from 1.76 to 1.57, suggesting that the AsymFlow parameterization is much more robust to auxiliary losses and can better leverage their benefits.

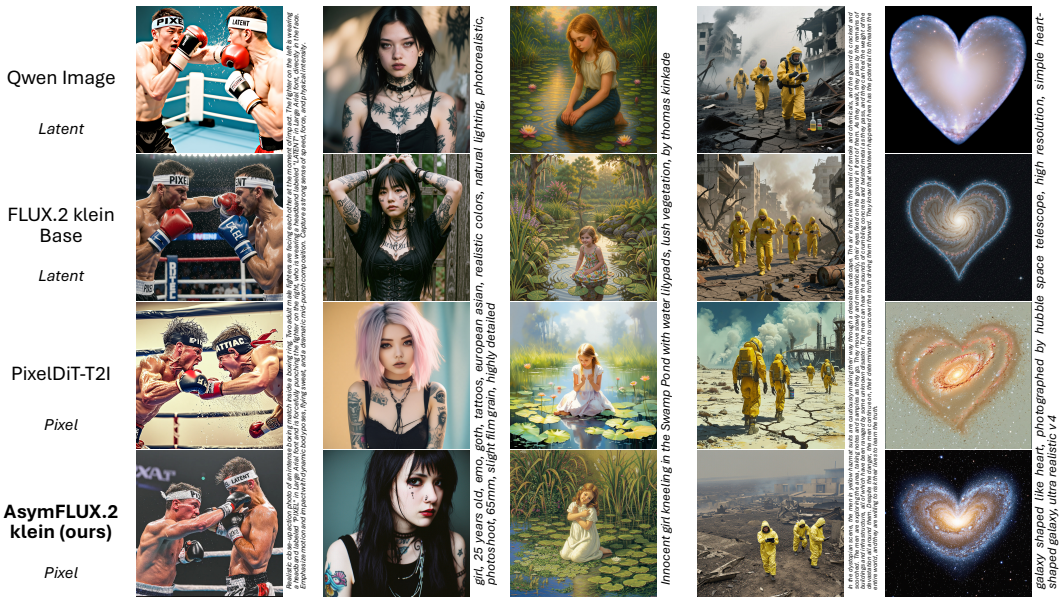


Figure 7: **Qualitative comparison of T2I diffusion models.** AsymFLUX.2 klein produces more realistic images with richer visual styles than prior models. More results are shown in Fig. 9 and 10.

Table 3: **Comparison with baselines and ablation studies**. All models are finetuned on the LAION-Aesthetics dataset [56] for 10K iterations, and evaluated on the COCO-10K dataset [38].

Method	HPSv3↑	HPSv2.1↑	VQA↑	CLIP↑	FID↓	pFID↓
FLUX.2 klein Base + latent finetune	10.70	0.290	**0.936**	0.276	**15.0**	**18.8**
FLUX.2 klein Base + DDT finetune	10.33	0.291	0.922	0.273	20.4	26.0
AsymFLUX.2 klein (standard FM)	12.03	0.293	0.922	0.277	20.2	25.4
AsymFLUX.2 klein (variance reduction)	12.99	0.296	0.925	**0.280**	18.5	27.8
+ perceptual correction	**13.06**	**0.297**	0.925	0.278	19.1	22.5

6.2 Finetuning Large Text-to-Image Models

For text-to-image generation, we finetune the pretrained FLUX.2 klein Base 9B latent flow model [6] (patch dimension $d = 128$) into a pixel-space AsymFlow model. We call the resulting model AsymFLUX.2 klein. The model is finetuned on 3M LAION-Aesthetics images [56], resized to one-megapixel resolution and captioned with Qwen2.5-VL [3]. To reduce overfitting, we freeze the base model and finetune only the input/output projection layers together with rank-256 LoRA adapters [24]. Sampling uses UniPC [74] with APG orthogonal-projection guidance [53]. We defer additional details to Appendix B.

Table 4: **System-level comparison of text-to-image (1024×1024) diffusion models**.

Method	HPSv3↑	DPG↑	GenEval↑
Latent diffusion models			
SDXL [49]	8.20	74.7	0.55
PixArt-Σ [8]	9.37	80.5	0.54
Hunyuan-DiT [36]	8.19	78.9	0.63
FLUX.1 dev [5]	**10.43**	84.0	0.67
Qwen-Image [66]	9.52	**87.8**	**0.86**
FLUX.2 klein Base [6]	9.50	85.2	0.80
Pixel diffusion models			
PixelDiT-T2I [71]	8.95	83.5	0.74
AsymFLUX.2 klein	**10.66**	**86.8**	**0.82**

Evaluation protocol. All text-to-image evaluations generate 1024×1024 images. For system-level comparison, we use three benchmarks: HPSv3 [44] measures human preference, which combines realism, style, and overall prompt following, while DPG-Bench [25] and GenEval [16] focus more on fine-grained entities, attributes, relations, counting, and composition. For controlled ablations, we generate images using 10K captions from the COCO 2014 validation set [37, 38] and report preference metrics HPSv3 [44] and HPSv2.1 [67], prompt-alignment metrics VQAScore [39] and CLIP score [50], and distribution metrics FID [19] and patch FID (pFID) [37].

System-level comparison. Table 4 compares AsymFLUX.2 klein (with variance reduction and perceptual correction) with prior latent and pixel text-to-image diffusion models. AsymFLUX.2 klein improves over its FLUX.2 klein latent base on all three benchmarks, with the largest gain

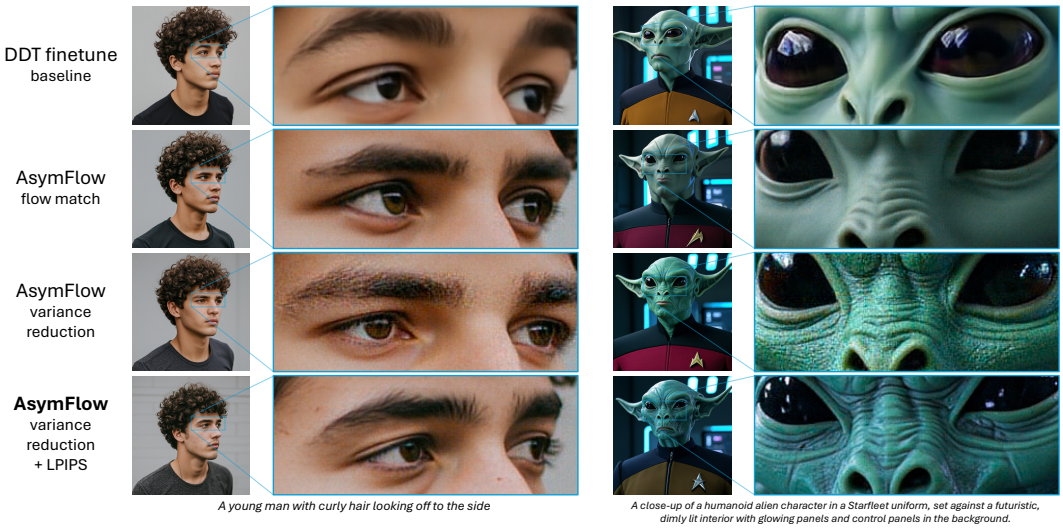


Figure 8: **Ablation of AsymFLUX.2 klein finetuning**. AsymFlow produces finer details than the DDT baseline. Variance reduction further improves details and texture but introduces excessive noise. The LPIPS perceptual correction suppresses this artifact while preserving the sharp appearance.

on HPSv3, indicating a substantial improvement in human-aligned visual quality. Consequently, it outperforms the prior pixel model PixelDiT-T2I [71] by a large margin across all metrics, establishing a new state of the art for pixel-space text-to-image generation. Figure 7 shows the same trend qualitatively: AsymFLUX.2 klein produces realistic and diverse visual styles with stronger texture, while popular latent models such as Qwen Image [3] and FLUX.2 klein Base [6] still have a more artificial appearance; compared to PixelDiT-T2I, AsymFLUX.2 klein recovers much sharper details in addition to other qualitative improvements, marking a significant step forward for pixel-space text-to-image generation.

Controlled baselines. To separate dataset effects from latent-to-pixel conversion, we include a latent-finetuned FLUX.2 klein baseline trained on the same data. We also include a $\boldsymbol{u}$-prediction pixel finetuning baseline with a DDT decoder head [65, 75], similar in spirit to PixelDiT [71]. The results are presented in Table 3: compared to the latent baseline, finetuned AsymFLUX.2 klein models yield clear improvements in HPSv3 and HPSv2.1, indicating that the improved overall quality comes from AsymFlow pixel-space conversion instead of dataset bias. In contrast, the DDT baseline falls behind in all metrics, despite having more parameters and capacity. This is also reflected in the qualitative comparison in Figure 8, where the DDT baseline produces blurry images and exhibits minor patch seams, while AsymFLUX.2 klein recovers sharper details and more realistic texture.

Loss ablations. The results in Table 3 also validate the effectiveness of variance reduction and perceptual correction losses: variance reduction boosts all metrics except pFID, due to its low-noise approximation error that introduces excessive noise (Figure 8). This is directly addressed by the LPIPS perceptual correction loss, which significantly improves pFID and HPS scores, resulting in the most natural and realistic texture in Figure 8.

7 Conclusion

We introduced AsymFlow, a rank-asymmetric flow velocity parameterization that enables high-dimensional pixel-space generation with plain diffusion transformers. When trained from scratch, this single parameterization yields a leading 1.57 FID among ImageNet pixel diffusion models. It also provides the first path for finetuning pretrained large latent flow models into pixel generators with improved visual fidelity, demonstrating AsymFlow's scalability and practical impact. This opens promising directions for high-fidelity image and video generation with finer low-level control, as well as other high-dimensional data modalities previously out of reach for flow-based modeling.

Limitations. Latent-to-pixel finetuning assumes a good patch-level linear lift. It may not work well when the pretrained latent space does not preserve pixel structure, such as in RAE models [75].

References

[1] Michael Samuel Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. In *ICLR*, 2023.

[2] Alan Baade, Eric Ryan Chan, Kyle Sargent, Changan Chen, Justin Johnson, Ehsan Adeli, and Li Fei-Fei. Latent forcing: Reordering the diffusion trajectory for pixel-space image generation. *arXiv preprint arXiv:2602.11401*, 2026.

[3] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025. URL `https://arxiv.org/abs/2502.13923`.

[4] Fan Bao, Shen Nie, Kaiwen Xue, Yue Cao, Chongxuan Li, Hang Su, and Jun Zhu. All are worth words: A ViT backbone for diffusion models. In *CVPR*, 2023.

[5] Black Forest Labs. Flux. `https://github.com/black-forest-labs/flux`, 2024.

[6] Black Forest Labs. Flux.2: Frontier visual intelligence. `https://bfl.ai/blog/flux-2`, 2025.

[7] Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, Clarence Ng, Ricky Wang, and Aditya Ramesh. Video generation models as world simulators. https://openai.com/research/video-generation-models-as-world-simulators, 2024.

[8] Junsong Chen, Chongjian Ge, Enze Xie, Yue Wu, Lewei Yao, Xiaozhe Ren, Zhongdao Wang, Ping Luo, Huchuan Lu, and Zhenguo Li. PixArt-Σ: Weak-to-strong training of diffusion transformer for 4k text-to-image generation. In *ECCV*, page 74–91, Berlin, Heidelberg, 2024. Springer-Verlag. ISBN 978-3-031-73410-6. doi: 10.1007/978-3-031-73411-3_5. URL `https://doi.org/10.1007/978-3-031-73411-3_5`.

[9] Shoufa Chen, Chongjian Ge, Shilong Zhang, Peize Sun, and Ping Luo. Pixelflow: Pixel-space generative models with flow. *arXiv preprint arXiv:2504.07963*, 2025.

[10] Zhennan Chen, Junwei Zhu, Xu Chen, Jiangning Zhang, Xiaobin Hu, Hanzhen Zhao, Chengjie Wang, Jian Yang, and Ying Tai. Dip: Taming diffusion models in pixel space. In *CVPR*, 2026.

[11] Katherine Crowson, Stefan Andreas Baumann, Alex Birch, Tanishq Mathew Abraham, Daniel Z Kaplan, and Enrico Shippole. Scalable high-resolution pixel-space image synthesis with hourglass diffusion transformers. In *ICML*, 2024.

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[13] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. In *ICLR*, 2022.

[14] Prafulla Dhariwal and Alexander Quinn Nichol. Diffusion models beat GANs on image synthesis. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *NeurIPS*, 2021. URL `https://openreview.net/forum?id=AAWuCvzaVt`.

[15] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis. In *ICML*, 2024.

[16] Dhruba Ghosh, Hannaneh Hajishirzi, and Ludwig Schmidt. Geneval: an object-focused framework for evaluating text-to-image alignment. In *NeurIPS*, Red Hook, NY, USA, 2023. Curran Associates Inc.

[17] Jiatao Gu, Shuangfei Zhai, Yizhe Zhang, Joshua M Susskind, and Navdeep Jaitly. Matryoshka diffusion models. In *ICLR*, 2023.

[18] Yoav HaCohen, Nisan Chiprut, Benny Brazowski, Daniel Shalem, Dudu Moshe, Eitan Richardson, Eran Levin, Guy Shiran, Nir Zabari, Ori Gordon, Poriya Panet, Sapir Weissbuch, Victor Kulikov, Yaki Bitterman, Zeev Melumian, and Ofir Bibi. Ltx-video: Realtime video latent diffusion. *arXiv preprint arXiv:2501.00103*, 2024. URL https://arxiv.org/abs/2501.00103.

[19] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, 2017.

[20] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS Workshop*, 2021.

[21] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.

[22] Emiel Hoogeboom, Jonathan Heek, and Tim Salimans. Simple diffusion: End-to-end diffusion for high resolution images. In *ICML*, pages 13213–13232, 2023.

[23] Emiel Hoogeboom, Thomas Mensink, Jonathan Heek, Kay Lamerigts, Ruiqi Gao, and Tim Salimans. Simpler diffusion (sid2): 1.5 fid on imagenet512 with pixel-space diffusion. In *CVPR*, 2025.

[24] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *ICLR*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

[25] Xiwei Hu, Rui Wang, Yixiao Fang, Bin Fu, Pei Cheng, and Gang Yu. Ella: Equip diffusion models with llm for enhanced semantic alignment. *arXiv preprint arXiv:2403.05135*, 2024. URL https://arxiv.org/abs/2403.05135.

[26] Allan Jabri, David Fleet, and Ting Chen. Scalable adaptive computation for iterative generation. In *ICML*, 2023.

[27] Qing Jin and Chaoyang Wang. Revisiting diffusion model predictions through dimensionality. *arXiv preprint arXiv:2601.21419*, 2026.

[28] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In *NeurIPS*, 2022.

[29] Tero Karras, Miika Aittala, Jaakko Lehtinen, Janne Hellsten, Timo Aila, and Samuli Laine. Analyzing and improving the training dynamics of diffusion models. In *CVPR*, 2024.

[30] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2014.

[31] Diederik P Kingma and Ruiqi Gao. Understanding diffusion objectives as the ELBO with simple data augmentation. In *NeurIPS*, 2023. URL https://openreview.net/forum?id=NnMEadcdyD.

[32] Weijie Kong, Qi Tian, Zijian Zhang, Rox Min, Zuozhuo Dai, Jin Zhou, Jiangfeng Xiong, Xin Li, Bo Wu, Jianwei Zhang, Kathrina Wu, Qin Lin, Junkun Yuan, Yanxin Long, Aladdin Wang, Andong Wang, Changlin Li, Duojun Huang, Fang Yang, Hao Tan, Hongmei Wang, Jacob Song, Jiawang Bai, Jianbing Wu, Jinbao Xue, Joey Wang, Kai Wang, Mengyang Liu, Pengyu Li, Shuai Li, Weiyan Wang, Wenqing Yu, Xinchi Deng, Yang Li, Yi Chen, Yutao Cui, Yuanbo Peng, Zhentao Yu, Zhiyu He, Zhiyong Xu, Zixiang Zhou, Zunnan Xu, Yangyu Tao, Qinglin Lu, Songtao Liu, Daquan Zhou, Hongfa Wang, Yong Yang, Di Wang, Yuhong Liu, Jie Jiang, and Caesar Zhong. Hunyuanvideo: A systematic framework for large video generative models. *arXiv preprint arXiv:2412.03603*, 2025. URL https://arxiv.org/abs/2412.03603.

[33] Tuomas Kynkäänniemi, Miika Aittala, Tero Karras, Samuli Laine, Timo Aila, and Jaakko Lehtinen. Applying guidance in a limited interval improves sample and distribution quality in diffusion models. In *NeurIPS*, 2024.

[34] Jiachen Lei, Keli Liu, Julius Berner, Y HoiM, Hongkai Zheng, Jiahong Wu, and Xiangxiang Chu. There is no VAE: End-to-end pixel-space generative modeling via self-supervised pre-training. In *ICLR*, 2026. URL https://openreview.net/forum?id=HbUoKPIZmp.

[35] Tianhong Li and Kaiming He. Back to basics: Let denoising generative models denoise. In *CVPR*, 2026.

[36] Zhimin Li, Jianwei Zhang, Qin Lin, Jiangfeng Xiong, Yanxin Long, Xinchi Deng, Yingfang Zhang, Xingchao Liu, Minbin Huang, Zedong Xiao, Dayou Chen, Jiajun He, Jiahao Li, Wenyue Li, Chen Zhang, Rongwei Quan, Jianxiang Lu, Jiabin Huang, Xiaoyan Yuan, Xiaoxiao Zheng, Yixuan Li, Jihong Zhang, Chao Zhang, Meng Chen, Jie Liu, Zheng Fang, Weiyan Wang, Jinbao Xue, Yangyu Tao, Jianchen Zhu, Kai Liu, Sihuan Lin, Yifu Sun, Yun Li, Dongdong Wang, Mingtao Chen, Zhichao Hu, Xiao Xiao, Yan Chen, Yuhong Liu, Wei Liu, Di Wang, Yong Yang, Jie Jiang, and Qinglin Lu. Hunyuan-dit: A powerful multi-resolution diffusion transformer with fine-grained chinese understanding. *arXiv preprint arXiv:2405.08748*, 2024. URL https://arxiv.org/abs/2405.08748.

[37] Shanchuan Lin, Anran Wang, and Xiao Yang. Sdxl-lightning: Progressive adversarial diffusion distillation. *arXiv preprint arXiv:2402.13929*, 2024. URL https://arxiv.org/abs/2402.13929.

[38] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *ECCV*, pages 740–755, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10602-1.

[39] Zhiqiu Lin, Deepak Pathak, Baiqi Li, Jiayao Li, Xide Xia, Graham Neubig, Pengchuan Zhang, and Deva Ramanan. Evaluating text-to-visual generation with image-to-text generation. In *ECCV*, 2024.

[40] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *ICLR*, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t.

[41] Qiang Liu. Rectified flow: A marginal preserving approach to optimal transport. *arXiv preprint arXiv:2209.14577*, 2022.

[42] Xingchao Liu, Chengyue Gong, and qiang liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *ICLR*, 2023. URL https://openreview.net/forum?id=XVjTT1nw5z.

[43] Nanye Ma, Mark Goldstein, Michael S. Albergo, Nicholas M. Boffi, Eric Vanden-Eijnden, and Saining Xie. Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers. In *ECCV*, 2024.

[44] Yuhang Ma, Xiaoshi Wu, Keqiang Sun, and Hongsheng Li. Hpsv3: Towards wide-spectrum human preference score. In *ICCV*, 2025.

[45] Zehong Ma, Longhui Wei, Shuai Wang, Shiliang Zhang, and Qi Tian. Deco: Frequency-decoupled pixel diffusion for end-to-end image generation. In *CVPR*, 2026.

[46] Zehong Ma, Ruihan Xu, and Shiliang Zhang. Pixelgen: Pixel diffusion beats latent diffusion with perceptual loss. *arXiv preprint arXiv:2602.02493*, 2026.

[47] Björn Ottosson. A perceptual color space for image processing, 2020. URL https://bottosson.github.io/posts/oklab/.

[48] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*, 2023.

[49] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. SDXL: Improving latent diffusion models for high-resolution image synthesis. In *ICLR*, 2024. URL https://openreview.net/forum?id=di52zR8xgf.

[50] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, pages 8748–8763, 2021.

[51] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Bjorn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

[52] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 234–241, 2015.

[53] Seyedmorteza Sadat, Otmar Hilliges, and Romann M. Weber. Eliminating oversaturation and artifacts of high guidance scales in diffusion models. In *ICLR*, 2025.

[54] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. In *NeurIPS*, 2022.

[55] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *ICLR*, 2022.

[56] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade W Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa R Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. LAION-5b: An open large-scale dataset for training next generation image-text models. In *NeurIPS Datasets and Benchmarks*, 2022. URL `https://openreview.net/forum?id=M3Y74vmsMcY`.

[57] Peter H. Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966. doi: 10.1007/BF02289451.

[58] Jaeyo Shin, Jiwook Kim, and Hyunjung Shim. Representation alignment for just image transformers is not easier than you think. *arXiv preprint arXiv:2603.14366*, 2026.

[59] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, pages 2256–2265, 2015.

[60] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *NeurIPS*, 2019.

[61] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *ICLR*, 2021.

[62] Shengbang Tong, Boyang Zheng, Ziteng Wang, Bingda Tang, Nanye Ma, Ellis Brown, Jihan Yang, Rob Fergus, Yann LeCun, and Saining Xie. Scaling text-to-image diffusion transformers with representation autoencoders. *arXiv preprint arXiv:2601.16208*, 2026.

[63] Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, Jianyuan Zeng, Jiayu Wang, Jingfeng Zhang, Jingren Zhou, Jinkai Wang, Jixuan Chen, Kai Zhu, Kang Zhao, Keyu Yan, Lianghua Huang, Mengyang Feng, Ningyi Zhang, Pandeng Li, Pingyu Wu, Ruihang Chu, Ruili Feng, Shiwei Zhang, Siyang Sun, Tao Fang, Tianxing Wang, Tianyi Gui, Tingyu Weng, Tong Shen, Wei Lin, Wei Wang, Wei Wang, Wenmeng Zhou, Wente Wang, Wenting Shen, Wenyuan Yu, Xianzhong Shi, Xiaoming Huang, Xin Xu, Yan Kou, Yangyu Lv, Yifei Li, Yijing Liu, Yiming Wang, Yingya Zhang, Yitong Huang, Yong Li, You Wu, Yu Liu, Yulin Pan, Yun Zheng, Yuntao Hong, Yupeng Shi, Yutong Feng, Zeyinzi Jiang, Zhen Han, Zhi-Fan Wu, and Ziyu Liu. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025. URL `https://arxiv.org/abs/2503.20314`.

[64] Shuai Wang, Ziteng Gao, Chenhui Zhu, Weilin Huang, and Limin Wang. Pixnerd: Pixel neural field diffusion. In *ICLR*, 2026. URL `https://openreview.net/forum?id=BDnOrExHmt`.

[65] Shuai Wang, Zhi Tian, Weilin Huang, and Limin Wang. Ddt: Decoupled diffusion transformer. In *CVPR*, 2026.

[66] Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, Yuxiang Chen, Zecheng Tang, Zekai Zhang, Zhengyi Wang, An Yang, Bowen Yu, Chen Cheng, Dayiheng Liu, Deqing Li, Hang Zhang, Hao Meng, Hu Wei, Jingyuan Ni, Kai Chen, Kuan Cao, Liang Peng, Lin Qu, Minggang Wu, Peng Wang, Shuting Yu, Tingkun Wen, Wensen Feng, Xiaoxiao Xu, Yi Wang, Yichang Zhang, Yongqiang Zhu, Yujia Wu, Yuxuan Cai, and Zenan Liu. Qwen-image technical report. *arXiv preprint arXiv:2508.02324*, 2025. URL https://arxiv.org/abs/2508.02324.

[67] Xiaoshi Wu, Yiming Hao, Keqiang Sun, Yixiong Chen, Feng Zhu, Rui Zhao, and Hongsheng Li. Human preference score v2: A solid benchmark for evaluating human preferences of text-to-image synthesis. *arXiv preprint arXiv:2306.09341*, 2023. URL https://arxiv.org/abs/2306.09341.

[68] Yilun Xu, Shangyuan Tong, and Tommi S. Jaakkola. Stable target field for reduced variance score estimation in diffusion models. In *ICLR*, 2023. URL https://openreview.net/forum?id=WmIwYTd0YTF.

[69] Jingfeng Yao, Bin Yang, and Xinggang Wang. Reconstruction vs. generation: Taming optimization dilemma in latent diffusion models. In *CVPR*, 2025.

[70] Sihyun Yu, Sangkyung Kwak, Huiwon Jang, Jongheon Jeong, Jonathan Huang, Jinwoo Shin, and Saining Xie. Representation alignment for generation: Training diffusion transformers is easier than you think. In *ICLR*, 2025.

[71] Yongsheng Yu, Wei Xiong, Weili Nie, Yichen Sheng, Shiqiu Liu, and Jiebo Luo. Pixeldit: Pixel diffusion transformers for image generation. In *CVPR*, 2026.

[72] Z-Image Team, Huanqia Cai, Sihan Cao, Ruoyi Du, Peng Gao, Steven Hoi, Zhaohui Hou, Shijie Huang, Dengyang Jiang, Xin Jin, Liangchen Li, Zhen Li, Zhong-Yu Li, David Liu, Dongyang Liu, Junhan Shi, Qilong Wu, Feng Yu, Chi Zhang, Shifeng Zhang, and Shilin Zhou. Z-image: An efficient image generation foundation model with single-stream diffusion transformer. *arXiv preprint arXiv:2511.22699*, 2025. URL https://arxiv.org/abs/2511.22699.

[73] Richard Zhang, Phillip Isola, Alexei Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.

[74] Wenliang Zhao, Lujia Bai, Yongming Rao, Jie Zhou, and Jiwen Lu. Unipc: A unified predictor-corrector framework for fast sampling of diffusion models. In *NeurIPS*, 2023.

[75] Boyang Zheng, Nanye Ma, Shengbang Tong, and Saining Xie. Diffusion transformers with representation autoencoders. In *ICLR*, 2026. URL https://openreview.net/forum?id=0u1LigJaab.

A Method Details

A.1 Low-Rank Subspace Construction

For transformer-based pixel generation, AsymFlow requires a patch-wise low-rank subspace. We use two constructions, depending on whether the model is trained from scratch or initialized from a latent model.

Orthonormality requirement. In both cases we require the columns of $\boldsymbol{A}$ to be orthonormal. This ensures that projecting standard pixel-space Gaussian noise preserves its Gaussian form inside the low-rank coordinates: if $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_D)$ and $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = \boldsymbol{I}_r$, then $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_r)$.

PCA basis for from-scratch training. Ideally, the low-rank directions would preserve the most perceptually important information in each image patch. When training from scratch, PCA gives a practical proxy by retaining the dominant patch variations without introducing an additional learned representation. Let $\boldsymbol{X} \in \mathbb{R}^{D\times N}$ collect N image patches with normalized pixel values. Taking the top left singular vectors of $\boldsymbol{X}$ gives the PCA subspace:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}, \qquad \boldsymbol{A} = \boldsymbol{U}_r, \qquad \boldsymbol{P} = \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}. \tag{8}$$

Here $\boldsymbol{U}_r$ denotes the top r columns of $\boldsymbol{U}$. Thus $\boldsymbol{P}$ keeps the data-adaptive PCA directions and removes the remaining patch-space directions from the noise prediction.

Procrustes basis for latent-to-pixel finetuning. For latent-to-pixel finetuning, the subspace should be aligned with the pretrained latent representation to minimize the paired gap $\|\boldsymbol{x}_0 - \boldsymbol{x}_0^{\mathrm{L}}\|$. Let $\boldsymbol{X} \in \mathbb{R}^{D\times N}$ collect image patches with normalized pixel values and $\boldsymbol{Z} \in \mathbb{R}^{d\times N}$ collect the corresponding latent tokens. We solve the orthogonal Procrustes problem [57]

$$\boldsymbol{A}^{\star} = \underset{\boldsymbol{A}\in\mathbb{R}^{D\times d},\ \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}=\boldsymbol{I}_d}{\arg\min} \|\boldsymbol{X} - \boldsymbol{A}\boldsymbol{Z}\|_{\mathrm{F}}^2. \tag{9}$$

This objective finds an orthonormal lift from latent tokens to pixel patches. Equivalently, it maximizes the inner-product alignment between $\boldsymbol{A}\boldsymbol{Z}$ and $\boldsymbol{X}$, so $\boldsymbol{A}^{\star} = \arg\max_{\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}=\boldsymbol{I}_d} \mathrm{Tr}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{Z}^{\mathrm{T}})$. If $\boldsymbol{X}\boldsymbol{Z}^{\mathrm{T}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$ is the compact SVD, the solution is

$$\boldsymbol{X}\boldsymbol{Z}^{\mathrm{T}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}, \qquad \boldsymbol{A}^{\star} = \boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}, \qquad \boldsymbol{P} = \boldsymbol{A}^{\star}(\boldsymbol{A}^{\star})^{\mathrm{T}}. \tag{10}$$

Procrustes aligns directions under the orthonormality constraint. It does not determine the correct pixel scale, so we apply the scalar calibration below.

A.2 Scale and Timestep Calibration

The Procrustes lift gives a directionally aligned low-rank pixel reconstruction, but its magnitude may not match the pixel scale within the Procrustes subspace. We therefore introduce a scalar s and use the calibrated lift

$$\boldsymbol{x}_0^{\mathrm{L}} = s\boldsymbol{A}\boldsymbol{z}_0, \qquad \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = \boldsymbol{I}_d, \qquad \boldsymbol{P} = \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}. \tag{11}$$

The scalar s is estimated from the same paired latent-token and pixel-patch statistics used above, by matching the Frobenius norm of the latents $\boldsymbol{Z}$ and the rescaled projected pixels $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{X}/s$:

$$s = \frac{\|\boldsymbol{A}^{\mathrm{T}}\boldsymbol{X}\|_{\mathrm{F}}}{\|\boldsymbol{Z}\|_{\mathrm{F}}}. \tag{12}$$

Equivalently, the calibrated lift $s\boldsymbol{A}\boldsymbol{Z}$ and the low-rank pixels $\boldsymbol{P}\boldsymbol{X}$ have the same Frobenius norm.

Scale calibration must also be reflected in noisy inputs, not only in the clean lift. Projecting a noisy pixel state gives signal coefficient $s\alpha_t$ and noise coefficient σ_t, so the latent-space signal-to-noise ratio (SNR) is $s\alpha_t/\sigma_t$. The SNR constraint first determines the latent time τ at which the pretrained model should be evaluated. Under the linear flow schedule, this gives

$$\frac{1-\tau}{\tau} = \frac{s(1-t)}{t} \quad \Longrightarrow \quad \tau = \frac{t}{s(1-t)+t}. \tag{13}$$

After fixing τ, the projected input must also have the correct noise magnitude $\sigma_\tau = \tau$. This determines the input rescaling

$$k = \frac{\tau}{t} = \frac{1}{s(1-t)+t}, \tag{14}$$

which places the projected state on the latent trajectory expected by the pretrained model, up to a low-rank approximation error:

$$\boldsymbol{A}^{\mathrm{T}}(k\boldsymbol{x}_t) \approx \boldsymbol{A}^{\mathrm{T}}(k\boldsymbol{x}_t^{\mathrm{L}}) = \alpha_\tau \boldsymbol{z}_0 + \sigma_\tau \boldsymbol{\epsilon}_{\boldsymbol{z}} = \boldsymbol{z}_\tau. \tag{15}$$

The output conversion must use the same calibration. The network is finetuned to predict the calibrated AsymFlow target

$$\boldsymbol{u}_{\mathrm{A}}^{\mathrm{cal}} \coloneqq \boldsymbol{P}\boldsymbol{\epsilon} - \frac{\boldsymbol{x}_0}{s}, \tag{16}$$

which is defined in the coordinate system of the rescaled input $k\boldsymbol{x}_t$. Recovering the original pixel-space full-rank velocity $\boldsymbol{u} = \boldsymbol{\epsilon} - \boldsymbol{x}_0$ gives

$$\boldsymbol{u} = \underbrace{\boldsymbol{P}\left(sk\, \boldsymbol{u}_{\mathrm{A}}^{\mathrm{cal}} + (1 - sk)\frac{\boldsymbol{x}_t}{\sigma_t} \right)}_{\text{low-rank subspace}} + \underbrace{(\boldsymbol{I} - \boldsymbol{P})\left(\frac{\boldsymbol{x}_t + s\boldsymbol{u}_{\mathrm{A}}^{\mathrm{cal}}}{\sigma_t} \right)}_{\text{orthogonal complement}}. \tag{17}$$

Eq. (17) is a generalized form of the uncalibrated conversion formula in Eq. (5). When $s = 1$ and $k = 1$, it reduces to the uncalibrated formula.

In practice, we apply this generalized conversion to the calibrated network prediction $\hat{\boldsymbol{u}}_{\mathrm{A}}^{\mathrm{cal}} = G_{\boldsymbol{\theta}}(k\boldsymbol{x}_t, kt)$ to obtain $\hat{\boldsymbol{u}}$, which is used in the flow matching loss (Eq. (2)) and denoising sampling.

A.3 Adaptive Weighting for Variance Reduction

The variance-reduced loss in Eq. (7) uses a patch-wise coefficient λ. For a given patch prediction, λ is determined by directly minimizing the loss residual along the one-dimensional control-variate direction (see Appendix C.3 for mathematical justification). Since the gradient of the squared loss is proportional to the corrected residual, this also minimizes the corresponding gradient norm, effectively selecting the lowest-variance target available along that direction.

The one-dimensional minimization has a closed-form solution given by an orthogonal projection. For each patch, define the low-rank prediction deviation of the frozen low-rank model as $\boldsymbol{d}^{\mathrm{L}} \coloneqq \boldsymbol{x}_0^{\mathrm{L}} - \hat{\boldsymbol{x}}_0^{\mathrm{L}}$ and the full-rank prediction deviation of the finetuned model as $\boldsymbol{d} \coloneqq \boldsymbol{x}_0 - \mathrm{stopgrad}(\hat{\boldsymbol{x}}_0)$. The variance-reduced loss residual is then $\boldsymbol{d} - \lambda \boldsymbol{d}^{\mathrm{L}}$. Minimizing the patch loss over λ gives the one-dimensional least-squares solution:

$$\lambda^\star = \arg\min_{\lambda} \|\boldsymbol{d} - \lambda \boldsymbol{d}^{\mathrm{L}}\|^2 = \frac{\langle \boldsymbol{d}, \boldsymbol{d}^{\mathrm{L}} \rangle}{\|\boldsymbol{d}^{\mathrm{L}}\|^2}. \tag{18}$$

Geometrically, this subtracts the component of the full-pixel prediction deviation that lies along the low-rank prediction deviation, leaving the smallest possible loss residual within this one-dimensional family. In practice, we use the clamped coefficient $\lambda = \min(\max(\lambda^\star, 0), 1)$.

A.4 Perceptual Correction

The variance-reduced loss in Eq. (7) uses the approximation $\mathbb{E}[\boldsymbol{x}_0^{\mathrm{L}} \mid \boldsymbol{x}_t] \approx \mathbb{E}[\boldsymbol{x}_0^{\mathrm{L}} \mid \boldsymbol{x}_t^{\mathrm{L}}]$, as analyzed in Appendix C.3. This approximation is valid when $\boldsymbol{x}_t - \boldsymbol{x}_t^{\mathrm{L}} \in \mathrm{Im}(\boldsymbol{I} - \boldsymbol{P})$, which is guaranteed at $t = 1$ because both inputs are pure noise. For $t < 1$, this condition requires $\boldsymbol{x}_0 - \boldsymbol{x}_0^{\mathrm{L}} \in \mathrm{Im}(\boldsymbol{I} - \boldsymbol{P})$, which generally does not hold, so the variance-reduction term $\lambda(\boldsymbol{x}_0^{\mathrm{L}} - \hat{\boldsymbol{x}}_0^{\mathrm{L}})$ can introduce approximation error in the low-rank subspace $\mathrm{Im}(\boldsymbol{P})$. Therefore, we need to reduce reliance on this term near the low-noise end of the trajectory.

Simply downweighting the variance-reduction term near low noise is not ideal, because the variance-reduced target is important for learning fine details. To compensate, we introduce a fading schedule $\omega_t \in [0, 1]$ that interpolates from the variance-reduction term to an LPIPS [73] perceptual loss between $\hat{\boldsymbol{x}}_0$ and $\boldsymbol{x}_0$. The variance-reduction term in Eq. (7) is multiplied by $1 - \omega_t$:

$$\mathcal{L}_{\mathrm{VR}} = \mathbb{E}_{t, \boldsymbol{x}_0, \boldsymbol{\epsilon}} \left[\frac{\left\| \boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0 - (1 - \omega_t)\lambda(\boldsymbol{x}_0^{\mathrm{L}} - \hat{\boldsymbol{x}}_0^{\mathrm{L}}) \right\|^2}{\sigma_t^2} \right], \tag{19}$$

while the complementary perceptual term is multiplied by ω_t:

$$\mathcal{L}_{\mathrm{P}} = \mathbb{E}_{t, \boldsymbol{x}_0, \boldsymbol{\epsilon}} \left[\frac{\omega_t \lambda}{\sigma_t^2} \mathrm{LPIPS}(\hat{\boldsymbol{x}}_0, \boldsymbol{x}_0) \right]. \tag{20}$$

Here λ is reused only as the patch-wise adaptive gate for the perceptual correction, and $1/\sigma_t^2$ recovers velocity-space weighting.

In our implementation, we define ω_t as a shifted signal-ratio schedule:

$$\omega_t = \frac{\alpha_t^2}{\alpha_t^2 + (\kappa\sigma_t)^2}, \tag{21}$$

where κ is a shift hyperparameter [15] that controls the transition. The final finetuning loss is

$$\mathcal{L} = \mathcal{L}_{\mathrm{VR}} + \omega_{\mathrm{P}}\mathcal{L}_{\mathrm{P}}, \tag{22}$$

where ω_{P} is a hyperparameter that controls the overall weight of the perceptual correction. In our experiments, we use $\kappa = 0.3$ and $\omega_{\mathrm{P}} = 0.2$. We did not perform a systematic hyperparameter sweep due to computational constraints, so there may be room for further improvement.

B Experiment Details

B.1 ImageNet Experiments

For ImageNet 256×256 experiments, we use the same architecture, optimizer, and other training hyperparameters as JiT-H/16 (see Table 9 of JiT [35]). Training for 600 epochs costs approximately 1750 NVIDIA H100 GPU hours. The REPA-enhanced variant follows the standard REPA setting [70]: we apply the REPA loss to the features after the 8th transformer block with loss weight 0.5.

At inference time, we set the velocity-recovery clamp to $\sigma_{\min} = 0.04$, which performs better than the JiT default $\sigma_{\min} = 0.05$ for both the JiT baseline and AsymFlow. Unless otherwise stated, all other inference settings follow JiT exactly, including the 50-step Heun ODE solver, class-balanced sampling, BF16 inference, and attention upcasting.

For each classifier-free guidance (CFG) [20] result, we grid-search the CFG scale with step size 0.1 and the guidance interval with step size 0.02 [33]. Table 5 lists the selected settings for Fig. 5. The final AsymFlow result in Table 1 uses CFG scale 2.3 and interval $[0, 0.88]$, while the REPA-enhanced result in Table 2 uses CFG scale 2.2 and interval $[0, 0.88]$.

Table 5: **Guidance settings for the ImageNet patch-rank sweep.** These settings are selected by grid-searching guided FID for each rank.

Patch rank r	**CFG scale**	**Guidance interval**
0	2.7	$[0, 0.82]$
2	2.6	$[0, 0.82]$
4	2.6	$[0, 0.82]$
8	2.5	$[0, 0.82]$
16	2.7	$[0, 0.82]$
32	2.7	$[0, 0.82]$
8 (random subspace)	2.8	$[0, 0.82]$

B.2 Text-to-Image Experiments

For text-to-image experiments, we represent pixels in Oklab color space [47] because of its perceptual uniformity, then normalize the values to mean 0 and standard deviation 1 before Procrustes alignment and scale calibration. The patch size is 16, matching the ImageNet model. Thus the pixel patch dimension is $D = 16 \times 16 \times 3 = 768$, while the AsymFlow rank follows the original FLUX.2 latent dimension, $r = d = 128$.

We finetune on a 3M subset of LAION-Aesthetics images [56], curated with safety and aesthetics filters. The images are resized to one-megapixel resolution and captioned with Qwen2.5-VL [3]. To reduce overfitting and preserve the pretrained model, we freeze the base weights and update only the input/output projection layers together with rank-256 LoRA adapters [24]. The trained modules are:

- `x_embedder`, `proj_out`, and `norm_out`;

- rank-256 LoRA adapters with dropout 0.05 on `*.ff.linear_in`, `*.ff.linear_out`, `*.ff_context.linear_in`, `*.ff_context.linear_out`, `timestep_embedder.linear_1`, `timestep_embedder.linear_2`, and `single_transformer_blocks.*.attn.to_out`.

Optimization uses 8-bit Adam [13, 30] with batch size 256, betas $(0.9, 0.95)$, learning rate 10^{-4} for all trainable parameters (except that `proj_out` uses 10^{-3}). The final model used in the system comparison is trained for 15K iterations, costing approximately 1100 NVIDIA H100 GPU hours. For evaluation, we use the exponential moving average (EMA) of the finetuned weights with the dynamic EMA schedule of Karras et al. [29] (using the hyperparameter $\gamma = 7.0$). Sampling uses UniPC [74] with APG orthogonal-projection guidance [53]. At each sampling step, we convert the denoised pixels to RGB color space and clamp the values to the valid range before converting them back to Oklab velocity. Table 6 summarizes the main text-to-image settings.

Table 6: **Text-to-image finetuning and evaluation settings.**

Setting	Value
Pixel color space	Normalized Oklab [47]
Patch size	16
Patch dimension D	768
Patch rank r	128
Subspace construction	Orthogonal Procrustes lift with scale calibration
LoRA rank / dropout	256 / 0.05
Flow shift [15]	17.0
Training resolution	1MP with mixed aspect ratios
Pre-shift time sampling	$\mathrm{LogitNormal}(0, 1)$
Optimizer	8-bit Adam [13, 30]
Learning rate	10^{-4} (10^{-3} for `proj_out`)
Adam betas	(0.9, 0.95)
Weight decay	0.0
Batch size	256
Training iterations	15K iterations
EMA	Dynamic EMA, $\gamma = 7.0$ [29]
Sampler	UniPC [74]
Guidance scale	4.0 with APG orthogonal projection [53]
Sampling steps	32

Latent baseline. For the latent finetuning baseline, we use its native flow shift of 7.0. Other settings are the same as AsymFlow for strict comparability.

DDT baseline. For the DDT pixel finetuning baseline, the DDT head uses two transformer blocks with a wider dimension of 32 attention heads ×192 features per head, similar to the RAE design [75]. We use the same $\boldsymbol{A}$ matrix as AsymFlow to initialize the input projection layer of the backbone, which closes the input gap and significantly improves the DDT baseline over a random initialization. The DDT head, input/output layers, and LoRA adapters are trained using a common learning rate of 10^{-4}. Other settings are the same as AsymFlow for strict comparability.

Inference time. AsymFLUX.2 klein uses the same number of tokens as the original FLUX.2 klein, so the per-step running time stays exactly the same as the original latent model. Since VAE is not used, the overall generation speed is marginally faster than the latent model.

C Mathematical Derivations

C.1 AsymFlow Decomposition and Recovery

We first make explicit the rank-r projector properties used throughout the paper. The columns of $\boldsymbol{A} \in \mathbb{R}^{D\times r}$ form an orthonormal basis for the chosen low-rank subspace, so $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = \boldsymbol{I}_r$. This orthonormality makes $\boldsymbol{P} = \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}$ the orthogonal projector onto that subspace. Applying $\boldsymbol{P}$ twice is the same as applying it once, so $\boldsymbol{P}^2 = \boldsymbol{P}$. The complementary projector $\boldsymbol{I} - \boldsymbol{P}$ removes everything in the low-rank subspace, which gives $(\boldsymbol{I} - \boldsymbol{P})\boldsymbol{P} = \boldsymbol{0}$. Together, these properties mean that any vector can be cleanly separated into a low-rank component and an orthogonal component. The notation is summarized as:

$$\boldsymbol{A} \in \mathbb{R}^{D\times r}, \qquad \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = \boldsymbol{I}_r, \qquad \boldsymbol{P} = \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}, \qquad \boldsymbol{P}^2 = \boldsymbol{P}, \qquad (\boldsymbol{I} - \boldsymbol{P})\boldsymbol{P} = \boldsymbol{0}. \tag{23}$$

We now restate the two targets in this notation. The standard velocity target combines full Gaussian noise with the data term. AsymFlow keeps the same full data term, but applies the projector only to the noise term:

$$\boldsymbol{u} \coloneqq \boldsymbol{\epsilon} - \boldsymbol{x}_0, \qquad \boldsymbol{u}_{\mathrm{A}} \coloneqq \boldsymbol{P}\boldsymbol{\epsilon} - \boldsymbol{x}_0. \tag{24}$$

Component decomposition. Projecting $\boldsymbol{u}_{\mathrm{A}}$ onto the low-rank subspace gives the true low-rank velocity. This branch of AsymFlow is still a velocity target. It contains low-rank noise minus low-rank data:

$$\boldsymbol{P}\boldsymbol{u}_{\mathrm{A}} = \boldsymbol{P}(\boldsymbol{P}\boldsymbol{\epsilon} - \boldsymbol{x}_0) = \boldsymbol{P}\boldsymbol{\epsilon} - \boldsymbol{P}\boldsymbol{x}_0 = \boldsymbol{P}(\boldsymbol{\epsilon} - \boldsymbol{x}_0) = \boldsymbol{P}\boldsymbol{u}. \tag{25}$$

Projecting $\boldsymbol{u}_{\mathrm{A}}$ onto the orthogonal complement removes the noise term entirely. This branch is no longer a velocity target. It is the orthogonal clean-data component up to a minus sign:

$$(\boldsymbol{I} - \boldsymbol{P})\boldsymbol{u}_{\mathrm{A}} = (\boldsymbol{I} - \boldsymbol{P})(\boldsymbol{P}\boldsymbol{\epsilon} - \boldsymbol{x}_0) = -(\boldsymbol{I} - \boldsymbol{P})\boldsymbol{x}_0. \tag{26}$$

Together, Eqs. (25) and (26) show that AsymFlow is velocity-like in $\mathrm{Im}(\boldsymbol{P})$ and $\boldsymbol{x}_0$-like in $\mathrm{Im}(\boldsymbol{I}-\boldsymbol{P})$.

Recovery rule. The same decomposition gives an exact route from the asymmetric target back to the standard velocity target. The low-rank branch is already in velocity form, so this component is kept directly:

$$\boldsymbol{P}\boldsymbol{u} = \boldsymbol{P}\boldsymbol{u}_{\mathrm{A}}. \tag{27}$$

The orthogonal branch is different. Since Eq. (26) says that $(\boldsymbol{I} - \boldsymbol{P})\boldsymbol{u}_{\mathrm{A}}$ equals the negative clean-data component, the orthogonal clean data is obtained by changing the sign:

$$(\boldsymbol{I} - \boldsymbol{P})\boldsymbol{x}_0 = -(\boldsymbol{I} - \boldsymbol{P})\boldsymbol{u}_{\mathrm{A}}. \tag{28}$$

This clean-data component is then converted to velocity using the usual $\boldsymbol{x}_0$-to-$\boldsymbol{u}$ relation. The orthogonal velocity is obtained by subtracting clean data from the noisy input and dividing by the noise level:

$$(\boldsymbol{I} - \boldsymbol{P})\boldsymbol{u} = (\boldsymbol{I} - \boldsymbol{P})\frac{\boldsymbol{x}_t - \boldsymbol{x}_0}{\sigma_t} = (\boldsymbol{I} - \boldsymbol{P})\frac{\boldsymbol{x}_t + \boldsymbol{u}_{\mathrm{A}}}{\sigma_t}. \tag{29}$$

Combining the direct low-rank velocity branch with the converted orthogonal branch gives the full-rank velocity target:

$$\boldsymbol{u} = \boldsymbol{P}\boldsymbol{u}_{\mathrm{A}} + (\boldsymbol{I} - \boldsymbol{P})\frac{\boldsymbol{x}_t + \boldsymbol{u}_{\mathrm{A}}}{\sigma_t}. \tag{30}$$

Thus, the asymmetric target itself contains enough information to reconstruct the standard full-rank velocity target exactly.

Endpoint cases. The rank controls how much of the target is velocity-like. At rank zero, the projector is zero, so AsymFlow becomes full $\boldsymbol{x}_0$-prediction up to sign. At full rank, the projector is the identity, so AsymFlow becomes standard velocity prediction:

$$r = 0 \implies \boldsymbol{P} = \boldsymbol{O},\ \boldsymbol{u}_{\mathrm{A}} = -\boldsymbol{x}_0, \qquad r = D \implies \boldsymbol{P} = \boldsymbol{I},\ \boldsymbol{u}_{\mathrm{A}} = \boldsymbol{\epsilon} - \boldsymbol{x}_0 = \boldsymbol{u}. \tag{31}$$

C.2 Latent–Pixel Flow Coupling at Initialization

We next show the trajectory coupling relationship that makes latent-to-pixel initialization exact: when the latent and lifted pixel ODEs start from paired noise, the entire low-rank pixel trajectory can be lifted from the latent trajectory plus the analytically determined orthogonal noise component. This trajectory coupling holds for both scale-calibrated (Appendix A.2) and uncalibrated AsymFlows. Below we analyze the uncalibrated version for simplicity.

Let $\boldsymbol{z}_0 \in \mathbb{R}^d$ denote a latent token, where d is the latent dimension. In this construction we choose the pixel low-rank subspace to have the same rank $r = d$, and use a linear lift $\boldsymbol{A} \in \mathbb{R}^{D\times d}$ from latent tokens to pixel patches. As before, the columns of $\boldsymbol{A}$ are orthonormal, so $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = \boldsymbol{I}_d$ and $\boldsymbol{P} = \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}$ projects onto the latent-induced pixel subspace. The lifted low-rank pixel target is $\boldsymbol{x}_0^{\mathrm{L}} \coloneqq \boldsymbol{A}\boldsymbol{z}_0$, and projecting pixel noise back through $\boldsymbol{A}^{\mathrm{T}}$ gives the latent noise $\boldsymbol{\epsilon}_{\boldsymbol{z}} \coloneqq \boldsymbol{A}^{\mathrm{T}}\boldsymbol{\epsilon}$. The notation is summarized as:

$$\boldsymbol{A} \in \mathbb{R}^{D\times d}, \qquad \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = \boldsymbol{I}_d, \qquad \boldsymbol{P} = \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}, \qquad \boldsymbol{x}_0^{\mathrm{L}} \coloneqq \boldsymbol{A}\boldsymbol{z}_0, \qquad \boldsymbol{\epsilon}_{\boldsymbol{z}} \coloneqq \boldsymbol{A}^{\mathrm{T}}\boldsymbol{\epsilon}. \tag{32}$$

With these definitions, projecting the lifted low-rank pixel process recovers the pretrained latent process.

Input identity. The pixel forward process diffuses the lifted low-rank pixels with full-rank pixel-space noise:

$$\boldsymbol{x}_t^{\mathrm{L}} \coloneqq \alpha_t \boldsymbol{x}_0^{\mathrm{L}} + \sigma_t \boldsymbol{\epsilon} = \alpha_t \boldsymbol{A}\boldsymbol{z}_0 + \sigma_t \boldsymbol{\epsilon}. \tag{33}$$

Projecting this noisy pixel sample by $\boldsymbol{A}^{\mathrm{T}}$ gives exactly the corresponding noisy latent sample:

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{x}_t^{\mathrm{L}} = \alpha_t \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{z}_0 + \sigma_t \boldsymbol{A}^{\mathrm{T}}\boldsymbol{\epsilon} = \alpha_t \boldsymbol{z}_0 + \sigma_t \boldsymbol{\epsilon}_{\boldsymbol{z}} = \boldsymbol{z}_t. \tag{34}$$

Thus, the lifted pixel model evaluates the pretrained latent network at the paired noisy latent state.

Output identity. The latent model predicts latent velocity $\boldsymbol{u}_{\boldsymbol{z}} \coloneqq \boldsymbol{\epsilon}_{\boldsymbol{z}} - \boldsymbol{z}_0$. Lifting this prediction to pixel space gives an AsymFlow-like target for the low-rank pixels $\boldsymbol{x}_0^{\mathrm{L}}$:

$$\boldsymbol{A}\boldsymbol{u}_{\boldsymbol{z}} = \boldsymbol{A}(\boldsymbol{\epsilon}_{\boldsymbol{z}} - \boldsymbol{z}_0) = \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\epsilon} - \boldsymbol{A}\boldsymbol{z}_0 = \boldsymbol{P}\boldsymbol{\epsilon} - \boldsymbol{x}_0^{\mathrm{L}}. \tag{35}$$

Therefore the low-rank pixel velocity $\boldsymbol{u}^{\mathrm{L}} \coloneqq \boldsymbol{\epsilon} - \boldsymbol{x}_0^{\mathrm{L}}$ is obtained by applying the same recovery rule from Sec. C.1 with $\boldsymbol{u}_{\mathrm{A}} = \boldsymbol{A}\boldsymbol{u}_{\boldsymbol{z}}$ and $\boldsymbol{x}_t = \boldsymbol{x}_t^{\mathrm{L}}$:

$$\boldsymbol{u}^{\mathrm{L}} = \boldsymbol{P}\boldsymbol{A}\boldsymbol{u}_{\boldsymbol{z}} + (\boldsymbol{I} - \boldsymbol{P})\frac{\boldsymbol{x}_t^{\mathrm{L}} + \boldsymbol{A}\boldsymbol{u}_{\boldsymbol{z}}}{\sigma_t}. \tag{36}$$

For analyzing the lifted latent initialization, this expression can be simplified because the lifted latent prediction already lies in the low-rank subspace, so we have $(\boldsymbol{I} - \boldsymbol{P})\boldsymbol{A}\boldsymbol{u}_{\boldsymbol{z}} = \boldsymbol{0}$. This gives

$$\boldsymbol{u}^{\mathrm{L}} = \boldsymbol{A}\boldsymbol{u}_{\boldsymbol{z}} + \frac{(\boldsymbol{I} - \boldsymbol{P})\boldsymbol{x}_t^{\mathrm{L}}}{\sigma_t}. \tag{37}$$

Thus, at initialization, the low-rank branch is exactly the lifted latent velocity, while the orthogonal branch is recovered directly from the current noisy pixel state. Note that this simplification does not apply to the finetuned AsymFlow model and should not be used in the implementation.

Trajectory coupling. The identities above are pointwise statements about the noisy input and the recovered velocity. What we need for initialization is slightly stronger: if the latent model and the lifted pixel model are solved in parallel from paired noise, then their whole trajectories remain paired, and their final samples still satisfy the same lifting relation.

Theorem 1. Let $\boldsymbol{\epsilon} \in \mathbb{R}^D$ be a pixel-space noise sample and let $\boldsymbol{\epsilon}_{\boldsymbol{z}} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{\epsilon}$ be its low-rank projection. Let $G_{\boldsymbol{\phi}}$ denote the pretrained latent flow velocity network. Consider the latent flow ODE on $(0, 1]$:

$$\frac{\mathrm{d}\boldsymbol{z}_t}{\mathrm{d}t} = G_{\boldsymbol{\phi}}(\boldsymbol{z}_t, t), \qquad \boldsymbol{z}_1 = \boldsymbol{\epsilon}_{\boldsymbol{z}}, \tag{38}$$

and the lifted pixel flow ODE obtained by applying the simplified form in Eq. (37) to the latent network output:

$$\frac{\mathrm{d}\boldsymbol{x}_t^{\mathrm{L}}}{\mathrm{d}t} = \boldsymbol{A}G_{\boldsymbol{\phi}}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{x}_t^{\mathrm{L}}, t) + \frac{(\boldsymbol{I} - \boldsymbol{P})\boldsymbol{x}_t^{\mathrm{L}}}{\sigma_t}, \qquad \boldsymbol{x}_1^{\mathrm{L}} = \boldsymbol{\epsilon}. \tag{39}$$

Then the two trajectories satisfy

$$\boldsymbol{x}_t^{\mathrm{L}} = \boldsymbol{A}\boldsymbol{z}_t + \sigma_t(\boldsymbol{I} - \boldsymbol{P})\boldsymbol{\epsilon} \quad \text{for all } t \in (0, 1]. \tag{40}$$

In particular, taking $t \to 0$ gives the final sample identity $\boldsymbol{x}_0^{\mathrm{L}} = \boldsymbol{A}\boldsymbol{z}_0$.

Proof. For brevity, write the orthogonal noise component as $\boldsymbol{\epsilon}^{\perp} \coloneqq (\boldsymbol{I} - \boldsymbol{P})\boldsymbol{\epsilon}$. Then the pixel noise decomposes into the lifted latent noise plus the orthogonal residual:

$$\boldsymbol{\epsilon} = \boldsymbol{P}\boldsymbol{\epsilon} + (\boldsymbol{I} - \boldsymbol{P})\boldsymbol{\epsilon} = \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\epsilon} + \boldsymbol{\epsilon}^{\perp} = \boldsymbol{A}\boldsymbol{\epsilon}_{\boldsymbol{z}} + \boldsymbol{\epsilon}^{\perp}. \tag{41}$$

At $t = 1$, this decomposition matches the two ODE initial conditions:

$$\boldsymbol{x}_1^{\mathrm{L}} = \boldsymbol{A}\boldsymbol{z}_1 + \sigma_1\boldsymbol{\epsilon}^{\perp}. \tag{42}$$

Now define a candidate lifted pixel trajectory from the latent trajectory:

$$\tilde{\boldsymbol{x}}_t^{\mathrm{L}} \coloneqq \boldsymbol{A}\boldsymbol{z}_t + \sigma_t\boldsymbol{\epsilon}^{\perp}. \tag{43}$$

We will show that this candidate trajectory satisfies the lifted pixel ODE in Eq. (39) with the same initial condition, so by uniqueness of ODE solutions, it must be identical to $\boldsymbol{x}_t^{\mathrm{L}}$ for all t. The candidate trajectory has exactly the input identity required by the latent network:

$$\boldsymbol{A}^{\mathrm{T}}\tilde{\boldsymbol{x}}_t^{\mathrm{L}} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{z}_t + \sigma_t\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\epsilon}^{\perp} = \boldsymbol{z}_t. \tag{44}$$

It also has an orthogonal component determined only by the fixed orthogonal noise:

$$(\boldsymbol{I} - \boldsymbol{P})\tilde{\boldsymbol{x}}_t^{\mathrm{L}} = \sigma_t \boldsymbol{\epsilon}^{\perp}. \tag{45}$$

Substituting these two identities into the lifted pixel vector field gives the lifted latent velocity plus the orthogonal noise velocity:

$$\boldsymbol{A}G_{\boldsymbol{\phi}}(\boldsymbol{A}^{\mathrm{T}}\tilde{\boldsymbol{x}}_t^{\mathrm{L}}, t) + \frac{(\boldsymbol{I} - \boldsymbol{P})\tilde{\boldsymbol{x}}_t^{\mathrm{L}}}{\sigma_t} = \boldsymbol{A}G_{\boldsymbol{\phi}}(\boldsymbol{z}_t, t) + \boldsymbol{\epsilon}^{\perp}. \tag{46}$$

The derivative of the candidate trajectory gives the same expression:

$$\frac{\mathrm{d}\tilde{\boldsymbol{x}}_t^{\mathrm{L}}}{\mathrm{d}t} = \boldsymbol{A}\frac{\mathrm{d}\boldsymbol{z}_t}{\mathrm{d}t} + \frac{\mathrm{d}\sigma_t}{\mathrm{d}t}\boldsymbol{\epsilon}^{\perp} = \boldsymbol{A}G_{\boldsymbol{\phi}}(\boldsymbol{z}_t, t) + \boldsymbol{\epsilon}^{\perp}, \tag{47}$$

where we used Eq. (38) and $\sigma_t = t$. Thus $\tilde{\boldsymbol{x}}_t^{\mathrm{L}}$ satisfies the lifted pixel ODE in Eq. (39). Since it also has the same value as $\boldsymbol{x}_t^{\mathrm{L}}$ at $t = 1$, uniqueness of the ODE solution gives

$$\boldsymbol{x}_t^{\mathrm{L}} = \tilde{\boldsymbol{x}}_t^{\mathrm{L}} = \boldsymbol{A}\boldsymbol{z}_t + \sigma_t(\boldsymbol{I} - \boldsymbol{P})\boldsymbol{\epsilon} \quad \text{for all } t \in (0, 1]. \tag{48}$$

Finally, taking $t \to 0$ gives $\boldsymbol{x}_0^{\mathrm{L}} = \boldsymbol{A}\boldsymbol{z}_0$. □

The same argument applies to Euler discretization with a shared time grid: if the relation holds before a step, the latent update changes the low-rank component by $\Delta t\, \boldsymbol{A}G_{\boldsymbol{\phi}}(\boldsymbol{z}_t, t)$, while the lifted pixel update additionally changes the orthogonal component by $\Delta t\, \boldsymbol{\epsilon}^{\perp}$, preserving the same paired form after the step; by induction, the relation holds at all steps. Thus, at network initialization, the lifted latent model is an exact low-rank pixel flow model. Note that this initialization is not yet a full AsymFlow model on real pixels, as finetuning replaces the lifted low-rank data target $\boldsymbol{x}_0^{\mathrm{L}}$ with the full-rank pixel target $\boldsymbol{x}_0$.

C.3 Details on Variance-Reduced Loss

The variance-reduced loss in Sec. 5.2 can be viewed as a control variate. The paired low-rank target $\boldsymbol{x}_0^{\mathrm{L}}$ is correlated with the full pixel target $\boldsymbol{x}_0$, and a frozen initialized low-rank model gives a good estimate of it. We use this paired target to reduce the variance of the pixel residual without changing the conditional mean target.

The exact control-variate identity is

$$\mathbb{E}\big[\boldsymbol{x}_0^{\mathrm{L}} - \mathbb{E}[\boldsymbol{x}_0^{\mathrm{L}}|\boldsymbol{x}_t] \,\big|\, \boldsymbol{x}_t\big] = \boldsymbol{0}. \tag{49}$$

Therefore adding any coefficient times this zero-mean residual does not change the conditional target. The posterior mean remains unchanged, while the sampled target can have lower variance:

$$\mathbb{E}\big[\boldsymbol{x}_0 - \lambda\big(\boldsymbol{x}_0^{\mathrm{L}} - \mathbb{E}[\boldsymbol{x}_0^{\mathrm{L}}|\boldsymbol{x}_t]\big) \,\big|\, \boldsymbol{x}_t\big] = \mathbb{E}[\boldsymbol{x}_0|\boldsymbol{x}_t]. \tag{50}$$

Before approximation, the objective is therefore equivalent to the standard flow matching loss in $\boldsymbol{x}_0$ format (Eq. (2)). The only role of the additional term is to reduce sampling variance when the low-rank residual explains part of the full pixel residual.

In practice, the conditional mean $\mathbb{E}[\boldsymbol{x}_0^{\mathrm{L}}|\boldsymbol{x}_t]$ is unavailable. We approximate it using the frozen low-rank model prediction $\hat{\boldsymbol{x}}_0^{\mathrm{L}}$ from the paired noisy low-rank sample:

$$\boldsymbol{x}_t^{\mathrm{L}} = \alpha_t \boldsymbol{x}_0^{\mathrm{L}} + \sigma_t \boldsymbol{\epsilon}, \qquad \mathbb{E}[\boldsymbol{x}_0^{\mathrm{L}}|\boldsymbol{x}_t] \approx \mathbb{E}[\boldsymbol{x}_0^{\mathrm{L}}|\boldsymbol{x}_t^{\mathrm{L}}] \approx \hat{\boldsymbol{x}}_0^{\mathrm{L}} = \boldsymbol{P}\boldsymbol{x}_t^{\mathrm{L}} - \sigma_t \boldsymbol{A}G_{\boldsymbol{\phi}}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{x}_t^{\mathrm{L}}, t). \tag{51}$$

Substituting this approximation gives the practical variance-reduced loss in Eq. (7).

The approximation $\mathbb{E}[\boldsymbol{x}_0^{\mathrm{L}}|\boldsymbol{x}_t] \approx \mathbb{E}[\boldsymbol{x}_0^{\mathrm{L}}|\boldsymbol{x}_t^{\mathrm{L}}]$ is exact under the sufficient condition that the full noisy input and the paired low-rank noisy input differ only in the orthogonal complement. In that case, their low-rank components match, so the frozen low-rank model receives the same low-rank information:

$$\boldsymbol{x}_t - \boldsymbol{x}_t^{\mathrm{L}} \in \mathrm{Im}(\boldsymbol{I} - \boldsymbol{P}) \implies \boldsymbol{A}^{\mathrm{T}}\boldsymbol{x}_t = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{x}_t^{\mathrm{L}}. \tag{52}$$

This requires either $t = 1$ or $\boldsymbol{x}_0 - \boldsymbol{x}_0^{\mathrm{L}} \in \mathrm{Im}(\boldsymbol{I} - \boldsymbol{P})$, which is generally not satisfied due to the non-linearity of the VAE encoder [51]. When this condition is not satisfied, the approximation error appears inside the low-rank subspace $\mathrm{Im}(\boldsymbol{P})$. To compensate for this, the perceptual correction is introduced in the low-noise regime in place of the variance reduction, as detailed in Sec. A.4.

D Additional Qualitative Results

Figure 9: Additional qualitative text-to-image comparisons (part A).

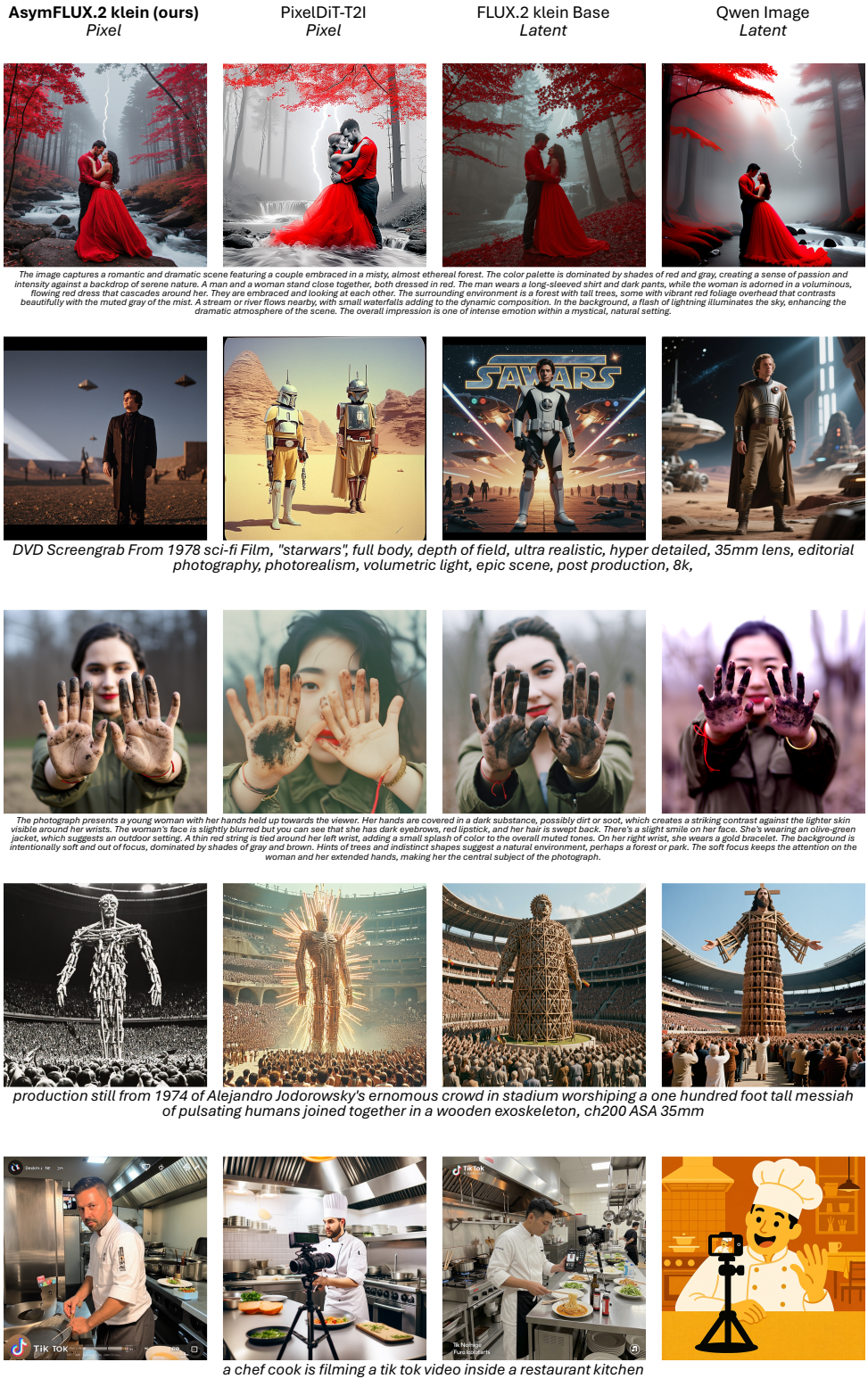


Figure 10: Additional qualitative text-to-image comparisons (part B).

E Impact Statement

Our method enhances the photorealism of diffusion models, which significantly benefits creative industries by enabling high-fidelity prototyping and asset creation. This advancement, however, presents a dual-use challenge: more realistic imagery facilitates the creation of convincing disinformation or non-consensual media, increasing the potential for societal harm. Higher visual quality also requires renewed scrutiny of dataset biases, as those biases will be rendered more persuasively. We open-source our model to encourage scientific replication, but emphasize that responsible deployment requires the use of standard safety filters and content provenance tools (like watermarking) to manage these risks.