FASTER DENSITY ESTIMATION AND INFERENCE-TIME STEERING FOR FLOW LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Flow language models generate discrete data by numerical integration in a continuous space using a learned denoiser. In order to estimate log-densities, one typically needs to estimate the divergence of the learned network. This is often achieved using the Hutchinson's estimator, which incurs notable computational overhead. We propose an alternative density estimation mechanism, which we term as Flow-based Analytic TimE-score (FATE), that is consistent, has negligible computational overhead over denoiser evaluations, and unlocks applications of ODE simulations for general inference-time steering problems.

1 INTRODUCTION

Large language models based on autoregressive mechanism (OpenAI, 2023) enjoy widespread applications, showcasing remarkable capabilities in solving complex tasks (Brown et al., 2020; Wei et al., 2022; authors, 2023). However, due to the autoregressive nature, they are fundamentally incapable of generating tokens in parallel. Discrete diffusion models (Austin et al., 2021; Campbell et al., 2022; Lou et al., 2024; Sahoo et al., 2024) provide an alternative that enables parallel generation, but they suffer from factorization errors when discretized during sampling (Lee et al., 2026).

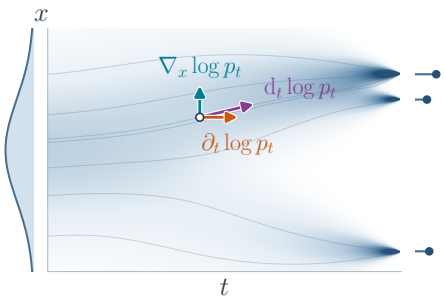


Figure 1: *Illustration of the FATE estimator.* Along the probability path induced by discrete observations, when the model estimates the marginal distributions of the tokens, we derive a closed-form estimator of the the time score $\partial_t \log p_t$. Combined with the analytical Stein score $\nabla_x \log p_t$, we obtain an estimate of the total score $\mathrm{d}_t \log p_t$.

Flow language models (Lee et al., 2026; Roos et al., 2026; Potaptchik et al., 2026; Chen et al., 2026; Batzolis et al., 2026; Deschenaux & Gulcehre, 2026; Yang et al., 2026) have recently achieved success in generating discrete data at scale and provide a promising contestant to autoregressive models and discrete diffusion models. They naturally allow parallel sampling of all tokens, do not suffer from factorization errors, and could further be distilled into flow maps (Boffi et al., 2026; Lee et al., 2026; Potaptchik et al., 2026; Roos et al., 2026), enabling few-step generations. Typically, the neural network model learns to model the marginal distribution of each token, a categorical distribution over the possible values of clean tokens. At inference time, samples are generated by integrating a continuous differential equation, transforming Gaussian noise to the target categorical distribution.

Our attention is on a specific technical question relevant for all flow language models: How to efficiently estimate densities. This is needed, for instance, when the models are steered at inference time to sample from a target distribution that is different from the one learned by the model. SuperDiff (Skreta et al., 2025b) considers the problem of combining multiple diffusion models and sampling from specific distributions induced by the models. Sequential Monte Carlo (SMC) (Wu et al., 2023; Singhal et al., 2025; Skreta et al., 2025a; Chanchu et al., 2026) instead allows sampling from potentially arbitrary distributions induced by the models.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Especially for ODE sampling, these densities can only be estimated using costly numerical procedures, forming a major computational challenge shared by all of the steering methods. The direct approach of numerically computing the divergence of the network is not feasible for large models, and hence previous works (Chen et al., 2026) resort to Hutchinson's trace estimator. It is significantly more practical than estimating the exact divergence, but still incurs additional computational and memory overhead due to differentiating through the large network. Furthermore, the resulting estimates are stochastic in nature. Our key contribution is resolving this challenge by providing a novel *closed-form estimator* for the required densities (Fig. 1), improving computational efficiency of all ODE based inference-time steering methods for flow language models. Overlooked by previous research, the trained models already provide us with such estimator *for free*.

Contributions. In this work, we tackle the problem of fast and accurate density estimation in flow language models. We i) derive a novel closed-form estimator of the total time log-density derivative, ii) show that, compared with the standard Hutchinson-based approach, our estimator introduces negligible overheads over denoiser evaluations, is up to 3.0 times faster, 31.7 times more memory efficient, and has zero variance from stochastic trace estimation; and iii) use these density estimates to enable inference-time steering of ODEs requiring access to intermediate densities, from using SuperDiff (Skreta et al., 2025b) to compose multiple models to using GLASS flows (Holderrieth et al., 2026) as SMC proposal to sample from tempered distribution and composition of models.

2 Background

In this section, we review how flow models are used to model a data distribution, which can be the discrete distribution as in language modeling, and then steered to model a different, target distribution.

Flow models and density estimation. Flow models are increasingly popular models for a data distribution p_1 on $\mathbb{R}^d$ (Liu et al., 2022; Lipman et al., 2023; Peluchetti, 2023; Albergo et al., 2025). They transport samples from a noise distribution p_0, typically Gaussian, to the data distribution p_1 using the velocity as an Ordinary Differential Equation (ODE) $\dot{x}_t = v_t(x_t)$ where $t \in [0, 1]$. The velocity is learned along the interpolation path induced by a prescribed marginal distributions $\text{Law}(\alpha_t x_1 + \beta_t x_0)$, which can be simulated by interpolating the noise and data samples $(x_0, x_1) \sim p_0 \otimes p_1$ with weights (α_t, β_t) where α_t starts at 0 and ends at 1 and β_t starts at 1 and ends at 0. From the perspective of diffusion models (Song et al., 2021b), the flow matching ODE is termed as the Probability Flow ODE (PF ODE), and one can alternatively simulate the stochastic dynamics between noise and data using a pair of forward SDE and reverse SDE.

Such models provide an estimate of the intermediate densities using $\log p_t(x_t) = \int_0^t \mathrm{d}_s \log p_s(x_s) ds + \log p_0(x_0)$. The instantaneous change of variable theorem (Chen et al., 2018) yields the identity $\mathrm{d}_s \log p_s(x_s) = -\nabla \cdot v_s(x_s) = -\text{tr}(\nabla v_s(x_s))$. However, directly computing the divergence is computationally prohibitive. For larger scale models, the popular solution is to use Hutchinson's trace estimator (Hutchinson, 1990; Grathwohl et al., 2019; Song et al., 2021b; Ai et al., 2026; Chen et al., 2026) $\mathrm{d}_s \log p_s(x_s) = -\mathbb{E}_\epsilon[\epsilon^\top \nabla v_s(x_s)\epsilon] \approx -\epsilon^\top \nabla v_s(x_s)\epsilon$, where ϵ is a random unit vector, a *probe*, drawn from a distribution with zero mean and identity covariance. One can in principle use multiple probes at the cost of further computational overheads. This is typically implemented as an extra reverse mode automatic differentiation, which is roughly as costly as a forward pass (Baydin et al., 2018), effectively doubling the number of network evaluations per step. Indeed, Pang et al. (2020, Table 3) noted that Hutchinson estimator (SSM) has around 2 times overhead both in terms of time and memory over a single network evaluation (DSM), though the exact amounts vary across applications. In general, Hutchinson's trace estimator is deemed inefficient for inference-time steering applications (Skreta et al., 2025b).

Applications to language modeling. In practice, the velocity in the previous paragraph is given by

$$v_t(x_t) = \frac{\dot{\beta}_t}{\beta_t} x_t + \left(\dot{\alpha}_t - \frac{\alpha_t \dot{\beta}_t}{\beta_t} \right) \mathbb{E}\left[x_1 | x_t\right], \tag{1}$$

where the denoiser $\mathbb{E}\left[x_1|x_t\right]$ is approximated by a neural network (Lee et al., 2026; Potaptchik et al., 2026; Chen et al., 2026; Batzolis et al., 2026). Specifically, a sentence is a collection of L tokens, where each token is drawn from a vocabulary V. By embedding each token in $\mathbb{R}^D$, the sentence becomes a vector in $\mathbb{R}^{LD}$. The embedded sequence is denoted by x_1 and the i-th token as x_1^i. The

neural network parameterizes for all i the categorical distribution $p_t(x_1^i|x_t)$ with V values, and can be trained using standard loss functions like the cross-entropy loss. Upon optimality, one forward pass of the neural network yields all $p(x_1^i|x_t)$, which can then be summed to compute $\mathbb{E}[x_1 \mid x_t]$ utilizing the known embeddings of the tokens.

Steering a flow model at inference time. Given a trained diffusion or flow, we may want to adjust its behavior at inference time, so as to sample from a different yet related distribution, e.g. the distribution tilted by a reward in the case of reward alignment (Uehara et al., 2025) $p_{\text{target}}(x) \propto p_1(x) \times \exp(\text{reward}(x))$. One approach to achieve this is inference-time steering (Wu et al., 2023; Singhal et al., 2025; Skreta et al., 2025a; He et al., 2026; Holderrieth et al., 2026), where the Sequential Monte Carlo (SMC) algorithm (Dai et al., 2022; Naesseth et al., 2019) is used to progressively sample from the sequence of target distributions as follows (see Algorithm 2 in Appendix): First, the user chooses a sequence of distributions described by their unnormalized densities $(\tilde{\gamma}_\tau)_{\tau\in[1,T]}$. That sequence of distributions typically starts with an "easy" distribution such as a Gaussian, and ends with the desired target distribution p_{target}. The SMC algorithm draws samples from that sequence. Each update of the SMC algorithm is performed using two kernels chosen by the user: a forward kernel $q_\tau(x_\tau|x_{\tau-1})$ and a backward kernel $s_{\tau-1}(x_{\tau-1}|x_\tau)$. While many choices of kernels are possible, they will strongly affect the performance of the algorithm. It is popular to choose the kernels as stochastic transitions $p_{t'|t}(x_{t'}|x_t)$ that are consistent with the marginals of the flow model described above. The forward kernel can be implemented as a time-reversed Stochastic Differential Equation (Reverse SDE) (Song et al., 2021b; Skreta et al., 2025a; Singhal et al., 2025; He et al., 2026) or as an ODE (Holderrieth et al., 2026). Specifically, the latter approach of Holderrieth et al. (2026) consists in defining an ODE called "inner" GLASS flow with time $s \in [0, 1]$, that transports x_t to $x_{t'}$.

3 Efficient density estimates

In this section, we introduce new efficient density tracking estimators along flows.

3.1 FATE: Density tracking for a single flow

We describe in Algorithm 1 a general recipe for Flow-based Analytic TimE-score (FATE), an efficient density estimation algorithm that is applicable when the flow model provides estimates of conditional moments (Chen et al., 2026; Eijkelboom et al., 2024; 2025; Batzolis et al., 2026; Deschenaux & Gulcehre, 2026; Yang et al., 2026). Our contribution is a novel, closed-form formula in step 2. Using our formula, we can track the marginal densities *for free* relying on the denoiser. Our experiments in later sections show that this dramatically increases the efficiency of density-tracking along ODE trajectories and its applications. In Appendix B.1, we derive our estimator and draw connections to higher-order score matching (Meng et al., 2021).

Before presenting our FATE algorithm, we introduce a few notations. Along a trajectory that solves the ODE $\dot{x}_t = v_t(x_t)$, we can track the log-density rate using the chain rule $\mathrm{d}_t \log p_t^{v_t}(x_t) = \partial_t \log p_t(x_t) + \nabla_x \log p_t(x_t)^\top v_t(x_t)$. By abuse of notation, we extend that formula to any input x that need not be a solution of the ODE.

Remark 1 *Our formula is consistent, in that the total derivative estimate is exact when the denoiser is exact. Moreover, it is more efficient and does not introduce notable computational overhead over simply evaluating the denoisers, as empirically validated in Section 5.2. However, given imperfect denoiser, it could be biased, unlike the estimates obtained through Hutchinson's trace estimator, which remains an unbiased estimator of the model's log density.*

Remark 2 *This formula also applies to inner GLASS flows, where one replaces t, α_t, β_t with $s, \bar{\alpha}_s, \bar{\beta}_s$ and quantities associated with ordinary denoiser $\mathbb{E}[x_1|x_t]$ with the GLASS denoiser $\mathbb{E}[x_1|x_t, \bar{x}_s]$.*

Our formula is correct for all linear one-sided stochastic interpolants. While it can be evaluated analytically given a flow language model, it is not directly available in *ordinary* flow matching models. Though it can be recovered through the second-order Tweedie formula (Meng et al., 2021), it results in estimating the divergence of the denoiser, again needing Hutchinson's estimator.

The formula affords further simplifications for specific parameterizations. For instance, Batzolis et al. (2026) considered the case where the embeddings are exactly given by analogical bits (Chen et al., 2023), where the value at each position is either 0 or 1. One can interpret it as an instance of flow

Algorithm 1 FATE estimator of density evolution along an ODE trajectory

Require: Conditional moments $m_t(x) = \mathbb{E}[x_1 \mid x_t = x]$ and $n_t(x) = \mathbb{E}[\|x_1\|^2 \mid x_t = x]$; reference velocity v_t; evaluation velocity $\tilde{v}_t$; trajectory $\dot{\tilde{x}}_t = \tilde{v}_t(\tilde{x}_t)$

1: **Evaluate the Stein score** (Albergo et al., 2025)

$$\nabla_x \log p_t(\tilde{x}_t) = -\frac{\tilde{x}_t - \alpha_t m_t(\tilde{x}_t)}{\beta_t^2}. \tag{2}$$

2: **Estimate the reference log-density rate** (ours)

$$\mathrm{d}_t \log p_t^{v_t}(\tilde{x}_t) = -LD\frac{\dot{\beta}_t}{\beta_t} + \left(\frac{\alpha_t^2\dot{\beta}_t}{\beta_t^3} - \frac{\alpha_t\dot{\alpha}_t}{\beta_t^2}\right)\left(n_t(\tilde{x}_t) - \|m_t(\tilde{x}_t)\|^2\right). \tag{3}$$

3: **Return the evaluation log-density rate** (Skreta et al., 2025a)

$$\mathrm{d}_t \log p_t^{\tilde{v}_t}(\tilde{x}_t) = \mathrm{d}_t \log p_t^{v_t}(\tilde{x}_t) + \nabla_x \log p_t(\tilde{x}_t)^\top \big(\tilde{v}_t(\tilde{x}_t) - v_t(\tilde{x}_t)\big). \tag{4}$$

language model, where the *tokens* are given by the analogical bits, and the model learns the marginal distributions across the bits. For these models, L denotes the length of sequences under bits, $D = 1$, $V = 2$ and $n_t(x_t) = m_t(x_t)$.

While the formulas above were presented for the original x, t coordinates, in Appendix A we describe a coordinate transformation that was employed by Chen et al. (2026) for numerical integrations. Under such coordinates, both the sampling update and the density update simplify.

3.2 FATE+: Density tracking for combinations of flows

The previous discussion focuses on the case where we use the original denoiser. However, we may also define a velocity field using as denoiser a weighted linear combination of the denoisers of different models for inference-time steering applications as detailed later. We show that we can still form a consistent estimator for such compositions, terming this mechanism *FATE+*. Under the special case of using the original denoiser of a single model, it reduces to the estimator as in Algorithm 1.

Proposition 3 *[FATE+] Consider the combination of N flow language models. The ith model provides an estimate of the denoiser, given by $\tilde{m}_t^{(i)}(x_t)$. The Stein score of the ith model used to form the velocity is scaled by c^i. The final velocity is given by*

$$v_t(x_t) = \left(\left(1 - \sum_{i=1}^N c_i\right)\frac{\dot{\alpha}_t}{\alpha_t} + \sum_{i=1}^N c_i\frac{\dot{\beta}_t}{\beta_t}\right)x_t + \left(\dot{\alpha}_t - \frac{\alpha_t\dot{\beta}_t}{\beta_t}\right)\sum_{i=1}^N c_i\tilde{m}_t^{(i)}(x_t), \tag{5}$$

and the total time derivative given by the negative divergence can be evaluated as

$$\mathrm{d}_t \log p_t(x_t) = -LD\left(\left(1 - \sum_{i=1}^N c_i\right)\frac{\dot{\alpha}_t}{\alpha_t} + \sum_{i=1}^N c_i\frac{\dot{\beta}_t}{\beta_t}\right) + \left(\frac{\alpha_t\dot{\beta}_t}{\beta_t} - \dot{\alpha}_t\right)\sum_{i=1}^N c_i\nabla_x \cdot \tilde{m}_t^{(i)}(x_t). \tag{6}$$

The exact values of $\nabla_{x_t} \cdot \tilde{m}_t^{(i)}(x_t)$ can be estimated in closed-form without needing numerical differentiations for many practical scenarios, and we detail three cases below.

Case 1. For the original denoiser learned by the model, it is given by

$$\nabla_{x_t} \cdot \tilde{m}_t^{(i)}(x_t) = \nabla_{x_t} \cdot m_t^{(i)}(x_t) = \frac{\alpha_t}{\beta_t^2}\left(n_t^{(i)}(x_t) - \left\|m_t^{(i)}(x_t)\right\|^2\right) \tag{7}$$

Case 2. When the model parameterizes the logits for all the V possible tokens, we may want to temper the logits using some temperature T_i before passing them into softmax (Acquaviva, 2026). The denoiser induced by the tempered logits satisfies

$$\nabla_{x_t} \cdot \tilde{m}_t^{(i)}(x_t) = \nabla_{x_t} \cdot m_{t,T_i}^{(i)}(x_t) = \frac{\alpha_t}{\beta_t^2 T_i}\left(n_{t,T_i}^{(i)}(x_t) - \left\|m_{t,T_i}^{(i)}(x_t)\right\|^2\right), \tag{8}$$

where $n^{(i)}_{t,T_i}(x_t)$ and $m^{(i)}_{t,T_i}(x_t)$ denote the corresponding quantities using tempered logits.

Case 3. For the specific method of CoBit (Batzolis et al., 2026), we can similarly temper the logits passed into the sigmoid (Acquaviva, 2026). The denoiser induced by the tempered logits satisfies

$$\nabla_{x_t} \cdot \tilde{m}^{(i)}_t(x_t) = \nabla_{x_t} \cdot m^{(i)}_{t,T_i}(x_t) = \frac{\alpha_t}{T_i \beta_t^2} \sum_{j=1}^{L} m^{(i)}_{t,T_i}(x_t)_j \left(1 - m^{(i)}_{t,T_i}(x_t)_j\right). \tag{9}$$

4 Inference-time steering

We next explore how to steer a flow that was previously trained to generate data samples $p_1(x)$, so that it instead generates samples from a new target distribution $p_{\text{target}}(x)$. We remark that we do not propose fundamentally new steering algorithms, but instead extend these frameworks to ODE sampling when necessary, and show how our efficient density estimator can be employed by existing steering methods, enabling practical implementations of steering algorithms that were previously considered computationally challenging for diffusion models (Skreta et al., 2025b;a; He et al., 2026).

4.1 SuperDiff

SuperDiff (Skreta et al., 2025b) is one way to combine multiple diffusion models. While it has efficient implementations for SDE sampling, ODE version requires divergence estimate hence was considered computationally challenging. Here we show that our proposed FATE estimator extends SuperDiff to allow efficient ODE sampling. The velocity used to generate samples is given by the sum of scaled velocities of the individual models in the form of

$$v_t(x_t) = \sum_{i=1}^{N} k^{(i)}_t(x_t) v^{(i)}_t(x_t). \tag{10}$$

where the scalar-valued mixture weights $k^{(i)}_t(x_t)$ generally depend on model densities $p^{(i)}_t(x_t)$. We would like to find appropriate choices of $k^{(i)}_t(x_t)$, so as to sample from the desired distribution $p_{\text{target}}(x)$. SuperDiff has two variants which we next present.

Sampling from a mixture of different data distributions For the OR variant of SuperDiff, $p_{\text{target}}(x) = \sum_{i=1}^{N} w_i p^{(i)}_t(x)$, where w are non-negative mixture weights that sum to 1, which is a mixture of densities of all the individual models. The ODE version consists in choosing the velocity weights in Eq. 10 to generate that terminal density (Skreta et al., 2025b) as

$$k^{(i)}_t(x_t) = \text{softmax}\left(\log p^{(i)}_t(x_t) + \log w_i\right). \tag{11}$$

Sampling from the equal probability set of different data distributions For the AND variant of SuperDiff, $p_{\text{target}}(x)$ is a probability distribution supported on $S = \{x \in \mathbb{R}^d | p_1(x) = \ldots = p_N(x)\}$. It does so by enforcing that the infinitesimal changes in log-density of all models are equal along generation trajectories. Building upon Skreta et al. (2025b), we derive the exact velocity weights, where the desirable $k^{(i)}_t(x_t)$ can be solved using a linear equation as detailed in Appendix B.3.2.

4.2 Sequential Monte Carlo

An alternative to SuperDiff that allows sampling from different target distributions is Sequential Monte Carlo algorithm (see Algorithm 2) with specific choices of $\tilde{\gamma}_\tau(x^i_\tau)$, $q_\tau(x_\tau|x_{\tau-1})$ and $s_{\tau-1}(x_{\tau-1}|x_\tau)$. Similar to SuperDiff, these quantities can depend on density estimates. We focus on two popular target distributions having practical applications: the tempered data distribution and the composition of two different data distributions (Thornton et al., 2025; Skreta et al., 2025a; He et al., 2026). There is existing literature on SMC using reverse SDE as proposal (Wu et al., 2023; Singhal et al., 2025; He et al., 2026) as well as using ODE as proposal for certain restricted scenarios (Holderrieth et al., 2026). However, such machinery remains incapable of using ODE to efficiently sample from general target distributions that depend on densities, even though ODE sampling is generally more efficient than SDE sampling (Holderrieth et al., 2026). Our FATE estimator fills in this gap for flow language models, providing estimates of both transition densities and marginal densities.

Sampling from the tempered data distribution. Given pretrained flow models generating samples from $p_\tau(x_\tau)$, we would like to sample from $\gamma(x) = \frac{p(x)^\beta}{Z}$. For simplicity, we instantiate Algorithm 2 with the unnormalized density $\tilde{\gamma}_\tau(x_\tau) = p_\tau(x_\tau)^{\lambda_\tau}$, where λ_τ is the tempering schedule such that $\lambda_T = \beta$, with $q_1(x_1)$ standard normal tempered by λ_1. The forward and backward kernels $q_\tau(x_\tau|x_{\tau-1}) = p_{\tau|\tau-1}(x_\tau|x_{\tau-1})$ and $s_{\tau-1}(x_{\tau-1}|x_\tau) = p_{\tau-1|\tau}(x_{\tau-1}|x_\tau)$ are the noising and denoising kernel of the stochastic interpolant. We remark that many other choices can be employed here. The above choices result in the following weights as shown in Appendix B.4

$$\tilde{w}_\tau = \frac{p_\tau(x_\tau)^{\lambda_\tau - 1}}{p_{\tau-1}(x_{\tau-1})^{\lambda_{\tau-1}-1}}. \tag{12}$$

Sampling from the product of different data distributions. Here, we would like to sample from $\gamma(x) = \frac{\prod_{i=1}^N p^{(i)}(x)}{Z}$, having access to pretrained flow models sampling from the individual distributions $p_\tau^{(i)}(x_\tau)$. We instantiate Algorithm 2 using as the unnormalized density $\tilde{\gamma}_\tau(x_\tau) = \prod_{i=1}^N p_\tau^{(i)}(x_\tau)$ and as the backward kernel $s_{\tau-1}(x_{\tau-1}|x_\tau) = p_{\tau-1|\tau}(x_{\tau-1}|x_\tau)$, the shared noising kernel. As for the forward kernel, we focus on the following two variants, which we refer to as *Kernel Mixture (KM)* and *Averaged Scores (AS)*. While KM uses the mixture of proposal kernels of different models, AS uses the averaged scores of all models to form the final kernel, and they are given by

$$q_\tau(x_\tau|x_{\tau-1})_{\text{KM}} = \sum_{i=1}^N \frac{1}{N} p^{(i)}_{\tau-1|\tau}(x_{\tau-1}|x_\tau), \quad q_\tau(x_\tau|x_{\tau-1})_{\text{AS}} = p^{\sum_{i=1}^N \frac{1}{N} m_t^{(i)}(x_t)}_{\tau-1|\tau}(x_\tau|x_{\tau-1}), \tag{13}$$

where for AS we slightly abuse notations to indicate that the transition kernel is induced by the average of the denoisers of all the models. For both KM and AS, the proposal densities and the densities of the original models can be tracked using the analytical formulas as described in Section 3. The resulting SMC weights are given by

$$\tilde{w}_\tau = \frac{\prod_{i=1}^N p_\tau^{(i)}(x_\tau)}{\prod_{i=1}^N p_{\tau-1}^{(i)}(x_{\tau-1})} \frac{p_{\tau-1|\tau}(x_{\tau-1}|x_\tau)}{q_\tau(x_\tau|x_{\tau-1})}. \tag{14}$$

The role of densities in inference-time steering. In the above, the kernels and weights of the SMC algorithm require access to three special densities: the *noising* kernel $p_{\tau|\tau-1}(x_\tau|x_{\tau-1})$, the *denoising* kernel $p_{\tau-1|\tau}(x_{\tau-1}|x_\tau)$, as well as the *marginal* densities $p_\tau(x_\tau)$. The *noising* kernel is typically the Gaussian kernel corresponding to the forward process of diffusion. The *denoising* kernel $p_{\tau|\tau-1}(x_\tau|x_{\tau-1})$ is often constructed using the reverse SDE kernel (Song et al., 2021b; Skreta et al., 2025a; Singhal et al., 2025; He et al., 2026). Instead, we build upon GLASS flows (Holderrieth et al., 2026), which construct an inner ODE from $x_{\tau-1}$ to x_τ. We apply our estimator of the total derivative as in Eq. 43 instead of Hutchinson's estimator and track the density along the inner ODE, which gives an estimate of $p_{\tau|\tau-1}(x_\tau|x_{\tau-1})$. For marginal densites, using the Bayes' rule (He et al., 2026)

$$\log p_\tau(x_\tau) + \log p_{\tau|\tau-1}(x_\tau|x_{\tau-1}) = \log p_{\tau-1}(x_{\tau-1}) + \log p_{\tau|\tau-1}(x_\tau|x_{\tau-1}). \tag{15}$$

Because the noising and denoising kernels are known, we obtain the difference $\log p_\tau(x_\tau) - \log p_{\tau-1}(x_{\tau-1})$: summing these differences across times and utilizing the tractable initial density, we recover estimates of the marginal densities $\log p_\tau(x_\tau)$ with negligible overhead.

5 Experiments

In this section, we validate the accuracy and efficiency of FATE estimator on pretrained models and demonstrate its practical applications in three inference-time steering experiments.

5.1 Density estimation

We compare the proposed FATE estimator against Hutchinson's estimator for density estimation using the pretrained LangFlow (Chen et al., 2026) and CoBit (Batzolis et al., 2026) models on LM1B (Chelba et al., 2014) and OWT (Gokaslan et al., 2019) datasets, and present the results in

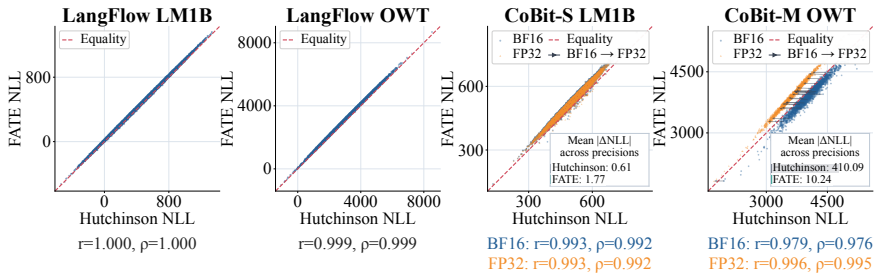


Figure 2: From left to right: density estimation results of LangFlow trained on LM1B and OWT, CoBit-S trained on LM1B and CoBit-M trained on OWT. Densities estimated using Hutchinson and FATE generally agree well, with high Pearson correlation coeffcient r and Spearman's rank correlation coefficient ρ. For CoBit-M using BF16 or FP32, while FATE is reasonably stable, Hutchinson's estimator yields notably different results, resulting in consistently larger or lower density estimates.

Figure 2. For density evaluations, the two estimators generally agree well with each other. While our estimator sometimes results in systematically larger or smaller NLLs, the correlations between the results of the two estimators are consistently high, often above 0.99. Up to numerical precision, the discrepancies between the two estimators can be attributed to both the randomness of Hutchinson and the models being imperfect. For perfect models, FATE matches Hutchinson in expectation.

Interestingly, we observe that NLLs estimated using Hutchinson can vary notably for CoBit models trained on OWT *across different data type*. While Batzolis et al. (2026) autocasted to BF16 for both training and sampling, using Hutchinson's estimator along with autocast results in consistently larger NLL values than FATE; the trend is reversed when using FP32 precision. While FP32 precision is in principle more accurate, it also deviates from the data type used in training and sampling, and it is unclear which precision we should prefer. We deem this as an undesirable property of Hutchinson's estimator, while our estimator is notably more stable across different precisions. We additionally verified that CoBit-S model trained on OWT behaved similar to CoBit-M trained on the same dataset. See Appendix C.1 for experimental details and Appendix D.1 for additional results.

5.2 Efficiency and Stochasticity

After verifying that FATE behaves similar to Hutchinson's, we now investigate the efficiency gain that we can achieve by replacing Hutchinson with FATE. We therefore use the same LangFlow and CoBit pre-trained models and benchmark their efficiency, in the form of runtime and memory overhead when evaluating the model. For self-conditioning, mimicking sampling, we feed in the network's prediction at the previous time point to the network at the current step. As shown in Figure 3, our estimator is at least 2.0 times faster and 1.8 times more memory efficient. Notably, for CoBit models, ours is up to 3.0 times faster (for CoBit-S trained on OWT, shown in Appendix D.2) and up to 31.7 times more memory efficient. Compared with simple denoiser evaluations, FATE only involves additional simple arithmetic expressions, hence is essentially *free* during sampling.

Furthermore, Hutchinson's trace estimator is, by definition, a stochastic estimator, as it directly depends on one or multiple stochastic probes, and it inevitably introduces some stochasticity in the resulting estimates. While Hutchinson is deterministic only at the limit of an infinite number of probes, our estimator is deterministic by definition. We empirically validated in Figure 3 that Hutchinson's variance is not negligible, despite decreasing as the number of probes increases.

5.3 Synthetic SMC

Consider the setting where both the underlying distributions and the optimal denoisers are analytically available. We keep $V = 3$, vary L between 1 and 5, and benchmark (1) SMC using reverse SDE with Euler-Maruyama integration as proposal with density estimated as in He et al. (2026) (`Reverse SDE`), (2) and (3) SMC using GLASS flows as proposal with Euler integration and density estimated using Hutchinson's estimator and our FATE estimator (`GLASS(Hutchinson)` and `GLASS(FATE)`), for both tempering and composition. Following Chen et al. (2026), all integrations are performed under the transformed coordinates. As shown in Figure 4, GLASS (FATE) consistently

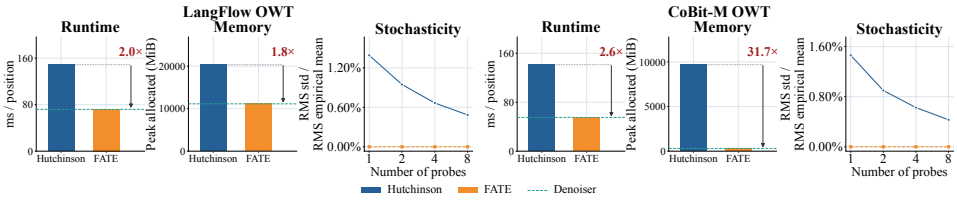


Figure 3: *Left*: results of LangFlow trained on OWT. *Right*: results of CoBit-M trained on OWT. Each side shows (from left to right) comparisons of Hutchinson and FATE for runtime, memory and stochasticity. FATE introduces minimal overheads over simple denoise evaluations, and is notably more efficient than using Hutchinson's estimator. The memory gain is especially significant for CoBit models, which could naturally translate to larger applicable batch sizes and higher throughputs given a fixed hardware budget. Hutchinson inevitably introduces variance due to stochastic probes.

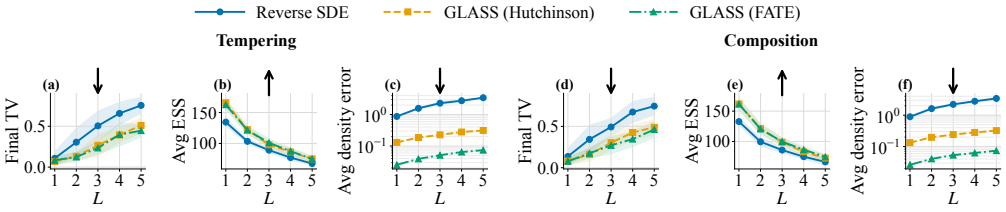


Figure 4: SMC metrics as a function of the sequence length, plotted as mean ± standard deviation. We report Final TV, Avg ESS, and Avg density error, where "Avg" denotes averages along the trajectory and "Final" denotes evaluation at its endpoint. GLASS flows with FATE consistently outperforms a classical baseline (Reverse SDE). Compared with Hutchinson's method, we perform better or similarly in terms of samples (a, b, d, e) while improving on density estimation (c, f).

achieves lower total variations to the ground truth, larger mean Effective Sample Size (ESS) and lower density estimation error along generation trajectories. Here we report composition results using KM proposal, but the overall conclusion remains for different proposals. See Appendix C.3 for details and Appendix D.3 for results of different composition proposals and SuperDiff variants.

5.4 Tempering

Inspired by recent discussion on flow language model evaluations (Acquaviva, 2026), we consider sampling from the tempered version of distributions learned by CoBit models (Batzolis et al., 2026). While previous work on flow language models typically heuristically tempers the logits (Acquaviva, 2026; Deschenaux & Gulcehre, 2026), we instead sample from the tempered distribution using SMC with 1024 particles over 5 random seeds. As shown in Figure 5, despite searching over a small range of temperatures, the resulting generative perplexity and entropy vary significantly, agreeing with the intuition that using lower temperature encourages sampling from high density regions and using higher temperature encourages more diverse samples. We provide numerical results in Appendix D.4.

5.5 Composition

We next consider composition of models. We employ the pre-trained models provided by Batzolis et al. (2026), and finetune them to obtain two separate experts each, given by i) CoBit-S pretrained on LM1B to model either (1) world and sports or (2) business and sports out of the 4 categories from AG news dataset (Zhang et al., 2015), ii) and iii) CoBit-S and CoBit-M pretrained on OWT to model either (1) positive reviews (4 or 5 stars) with balanced product categories or (2) reviews for electronic products with balanced review sentiments using Amazon Reviews 2023 (Hou et al., 2026). For sampling, we consider i) and ii) sampling using PF ODE of either experts, and iii) and iv) sampling using SuperDiff AND or OR. We then take the generated samples and feed them into pretrained classifiers to classify them according to i) categories for AG news, and ii) sentiments and categories for Amazon Reviews 2023. We generate 1024 samples using each sampler with 5 different random seeds and report the averaged results in Figure 6. SuperDiff AND achieves diverse samples that

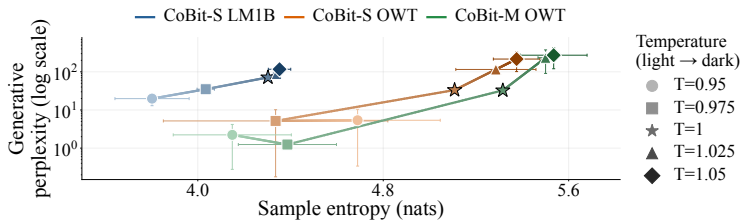


Figure 5: Sampling from a tempered version of the data distribution at inference-time using an Sequential Monte Carlo (SMC) algorithm. The algorithm requires density estimates provided by our FATE estimator (Section 4.2). This enables the user to decrease generative perplexity or increase sample diversity by varying the temperature.

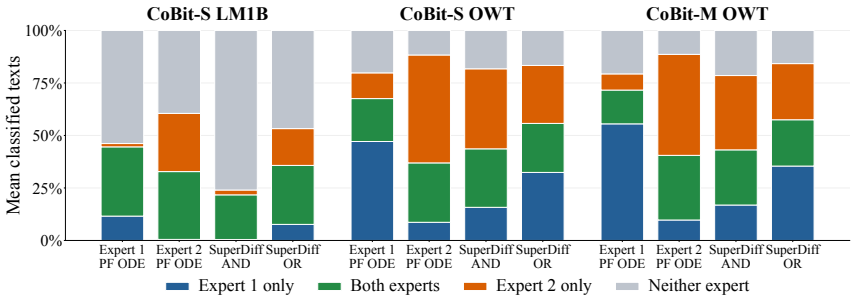


Figure 6: By sampling using PF ODE and tracking densities using our FATE estimator, we demonstrate successful model compositions using SuperDiff, interpolating between the two expert models.

differ from both experts, and SuperDiff OR successfully draws samples that appear approximately the equal mixtures of the two experts, validating that our density estimates enable practical model compositions. Details and additional results are provided in Appendix C.4 and Appendix D.5.

6 Discussion

Our key insight is that flow language models provide a natural estimate of $\mathrm{d}_t \log p_t(x_t)$. This estimate relies on a new formula for the partial time score $\partial_t \log p_t(x_t)$, which is key to the literature on infinitesimal density ratio estimation (Choi et al., 2022; Yu et al., 2025; Chen et al., 2025). While such time scores typically need to be learned from data using a different loss function than ordinary flow matching since ordinary loss only learns the posterior mean, flow language models can provide such estimates *for free* upon optimality since they learn to model the entire marginal distributions.

An alternative to constructing density estimates along sampling trajectory is to train the model to directly output energy estimates, in the form of an energy-based diffusion model (Du et al., 2023; Thornton et al., 2025; Guth et al., 2025; OuYang et al., 2026a; Yu et al., 2026), which can provide density estimates through a single network evaluation. However, for both training and sampling of such model family, one typically needs to evaluate the gradients of the energy network, leading to computational overheads over diffusion models that model the Stein scores. Despite the computational tradeoffs, it remains unclear which approach leads to more accurate density estimates.

We focused on inference-time steering. However, these density estimates have widespread applications apart from inference-time steering, from reinforcement learning (Liu et al., 2025) to building Boltzmann generators (Rehman et al., 2026; Ai et al., 2026; OuYang et al., 2026b).

7 Conclusion

We proposed a novel density estimator that is efficient and can be evaluated *for free* along generation trajectories, enabling practical applications of inference-time steering algorithms relying on density estimates along ODE generation trajectories.

AI USE STATEMENT

In this work, we used generative AI tools to provide critical ingredients for proving mathematical claims, assist in the writing of proofs, design or provide feedback on research methodology or experiments, implement methods, support qualitative and thematic data analysis and interpret results. We have not used generative AI tools to help develop theoretical models or conceptual frameworks, formulate mathematical claims or propose or refine hypotheses, and generating synthetic datasets, assisting with translation and cleaning and reformatting dataset are not applicable to this work. Additionally, we used generative AI tools to suggest experimental parameters and create or edit software code. We have reviewed all AI-assisted work, validated the mathematical results and reviewed LLM-generated code. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide code that can reproduce our results as supplementary materials, and describe experimental details in Appendix C.

REFERENCES

Sam Acquaviva. Fixing flow model evals, 2026. URL https://samacquaviva.com/projects/flow-evals/.

Xinyue Ai, Yutong He, Albert Gu, Ruslan Salakhutdinov, J Zico Kolter, Nicholas Matthew Boffi, and Max Simchowitz. Joint distillation for fast likelihood evaluation and sampling in flow-based models. In *The fourteenth international conference on learning representations*, 2026.

Michael Albergo, Nicholas M. Boffi, and Eric Vanden-Eijnden. Stochastic interpolants: a unifying framework for flows and diffusions. *Journal of Machine Learning Research*, 26(209):1–80, 2025.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in neural information processing systems*, 2021.

BIG-bench authors. Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856.

Georgios Batzolis, Mark Girolami, and Luca Ambrogioni. Towards closing the autoregressive gap in language modeling via entropy-gated continuous bitstream diffusion, 2026. arXiv: 2605.07013 [cs.CL].

Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic Differentiation in Machine Learning: a Survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

Nicholas Matthew Boffi, Michael Samuel Albergo, and Eric Vanden-Eijnden. How to build a consistency model: Learning flow maps via self-distillation. In *The thirty-ninth annual conference on neural information processing systems*, 2026.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in neural information processing systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020.

Andrew Campbell, Joe Benton, Valentin De Bortoli, Thomas Rainforth, George Deligiannidis, and Arnaud Doucet. A continuous time framework for discrete denoising models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in neural information processing systems*, volume 35, pp. 28266–28279. Curran Associates, Inc., 2022.

Lohithsai Yadala Chanchu, Hany Abdulsamad, and Christian A Naesseth. Discrete diffusion inference-time control with nested sequential monte carlo. *arXiv preprint arXiv:2608.20123*, 2026.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling, 2014. arXiv: 1312.3005 [cs.CL].

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Ting Chen, Ruixiang ZHANG, and Geoffrey Hinton. Analog bits: Generating discrete data using diffusion models with self-conditioning. In *The eleventh international conference on learning representations*, 2023.

Wei Chen, Shigui Li, Jiacheng Li, Junmei Yang, John Paisley, and Delu Zeng. Dequantified diffusion-schrödinger bridge for density ratio estimation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 8427–8452. PMLR, 13–19 Jul 2025.

Yuxin Chen, Chumeng Liang, Hangke Sui, Ruihan Guo, Chaoran Cheng, Jiaxuan You, and Ge Liu. LangFlow: Continuous diffusion rivals discrete in language modeling, 2026. arXiv: 2604.11748 [cs.CL].

Kristy Choi, Chenlin Meng, Yang Song, and Stefano Ermon. Density ratio estimation via infinitesimal classification. In *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pp. 2552–2573. PMLR, 28–30 Mar 2022.

Chenguang Dai, Jeremy Heng, Pierre E. Jacob, and Nick Whiteley. An invitation to sequential monte carlo samplers. *Journal of the American Statistical Association*, 117(539):1587–1600, 2022. doi: 10.1080/01621459.2022.2087659. tex.eprint: https://doi.org/10.1080/01621459.2022.2087659.

Justin Deschenaux and Caglar Gulcehre. Language modeling with hyperspherical flows, 2026. arXiv: 2605.11125 [cs.LG].

Yilun Du, Conor Durkan, Robin Strudel, Joshua B. Tenenbaum, Sander Dieleman, Rob Fergus, Jascha Sohl-Dickstein, Arnaud Doucet, and Will Sussman Grathwohl. Reduce, Reuse, Recycle: Compositional Generation with Energy-Based Diffusion Models and MCMC. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 8489–8510. PMLR, July 2023.

F. Eijkelboom, H. Zimmermann, S. Vadgama, E. J. Bekkers, M. Welling, J-W. van de Meent*, and C. A. Naesseth*. Controlled generation with equivariant variational flow matching. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, Vancouver, Canada, Jul 2025. * equal contribution.

Floor Eijkelboom, Grigory Bartosh, Christian A. Naesseth, Max Welling, and Jan-Willem van de Meent. Variational Flow Matching for Graph Generation. In *Advances in Neural Information Processing Systems*, volume 37, pp. 11735–11764. Curran Associates, Inc., 2024.

Aaron Gokaslan, Vanya Cohen, Ellie Pavlick, and Stefanie Tellex. OpenWebText corpus, 2019.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, and David Duvenaud. Scalable reversible generative models with free-form continuous dynamics. In *International Conference on Learning Representations*, 2019.

Florentin Guth, Zahra Kadkhodaie, and Eero P Simoncelli. Learning normalized image densities via dual score matching. In *The thirty-ninth annual conference on neural information processing systems*, 2025.

Jiajun He, José Miguel Hernández-Lobato, Yuanqi Du, and Francisco Vargas. RNE: plug-and-play diffusion inference-time control and energy-based training. In *The fourteenth international conference on learning representations*, 2026.

Peter Holderrieth, Uriel Singer, Tommi Jaakkola, Ricky T. Q. Chen, Yaron Lipman, and Brian Karrer. GLASS flows: Efficient inference for reward alignment of flow and diffusion models. In *The fourteenth international conference on learning representations*, 2026.

Yupeng Hou, Jiacheng Li, Xiangjun Fu, Zhankui He, An Yan, Xiusi Chen, and Julian McAuley. Bridging language and items for retrieval and recommendation: Benchmarking LLMs as semantic encoders. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th annual meeting of the Association for Computational Linguistics (volume 1: Long papers)*, pp. 3251–3265, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6.

M.F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 19(2):433–450, 1990. tex.eprint: https://doi.org/10.1080/03610919008812866.

Chanhyuk Lee, Jaehoon Yoo, Manan Agarwal, Sheel Shah, Jerry Huang, Aditi Raghunathan, Seunghoon Hong, Nicholas M. Boffi, and Jinwoo Kim. Flow map language models: One-step language modeling via continuous denoising, 2026. arXiv: 2602.16813 [cs.CL].

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 7871–7880, Online, July 2020. Association for Computational Linguistics.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow Matching for Generative Modeling. In *The Eleventh International Conference on Learning Representations*, 2023.

Jie Liu, Gongye Liu, Jiajun Liang, Yangguang Li, Jiaheng Liu, Xintao Wang, Pengfei Wan, Di ZHANG, and Wanli Ouyang. Flow-GRPO: Training flow matching models via online RL. In *Advances in neural information processing systems*, volume 38, pp. 40783–40818. Curran Associates, Inc., 2025.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st international conference on machine learning*, volume 235 of *Proceedings of machine learning research*, pp. 32819–32848. PMLR, July 2024.

Chenlin Meng, Yang Song, Wenzhe Li, and Stefano Ermon. Estimating High Order Gradients of the Data Distribution by Denoising. In *Advances in Neural Information Processing Systems*, volume 34, pp. 25359–25369. Curran Associates, Inc., 2021.

John Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. TextAttack: a framework for adversarial attacks, data augmentation, and adversarial training in NLP. In *Proceedings of the 2020 conference on empirical methods in natural language processing: System demonstrations*, pp. 119–126, 2020.

Christian A Naesseth, Fredrik Lindsten, and Thomas B Schön. Elements of sequential monte carlo. *Foundations and Trends® in Machine Learning*, 12(3):187–306, 2019.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. arXiv: 2303.08774 tex.bibsource: dblp computer science bibliography, https://dblp.org tex.timestamp: Mon, 28 Aug 2023 21:26:19 +0200.

RuiKang OuYang, Louis Grenioux, and José Miguel Hernández-Lobato. A diffusive classification loss for learning energy-based generative models. In *Forty-third international conference on machine learning*, 2026a.

RuiKang OuYang, Hanlin Yu, Xinyue Ai, Yutong He, Nicholas Matthew Boffi, Pradeep Ravikumar, José Miguel Hernández-Lobato, Max Simchowitz, Benjamin Kurt Miller, and Omar Chehab. Few-step boltzmann generators via scalable likelihood flow maps. In *ICML 2026 workshop on structured probabilistic inference & generative modeling*, 2026b.

Tianyu Pang, Kun Xu, Chongxuan LI, Yang Song, Stefano Ermon, and Jun Zhu. Efficient learning of generative models via finite-difference score matching. In *Advances in Neural Information Processing Systems*, volume 33, pp. 19175–19188. Curran Associates, Inc., 2020.

Stefano Peluchetti. Non-denoising forward-time diffusions. *arXiv preprint arXiv:2312.14589*, 2023.

Peter Potaptchik, Jason Yim, Adhi Saravanan, Peter Holderrieth, Eric Vanden-Eijnden, and Michael S. Albergo. Discrete flow maps, 2026. arXiv: 2604.09784 [stat.ML].

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, and others. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Danyal Rehman, Tara Akhound-Sadegh, Artem Gazizov, Yoshua Bengio, and Alexander Tong. FALCON: Few-step accurate likelihoods for continuous flows. In *The fourteenth international conference on learning representations*, 2026.

Daan Roos, Oscar Davis, Floor Eijkelboom, Michael Bronstein, Max Welling, İsmail İlkan Ceylan, Luca Ambrogioni, and Jan-Willem van de Meent. Categorical flow maps, 2026. arXiv: 2602.12233 [cs.LG].

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in neural information processing systems*, volume 37, pp. 130136–130184. Curran Associates, Inc., 2024. doi: 10.52202/079017-4135.

Raghav Singhal, Zachary Horvitz, Ryan Teehan, Mengye Ren, Zhou Yu, Kathleen Mckeown, and Rajesh Ranganath. A general framework for inference-time scaling and steering of diffusion models. In *Proceedings of the 42nd international conference on machine learning*, volume 267 of *Proceedings of machine learning research*, pp. 55810–55827. PMLR, July 2025.

Marta Skreta, Tara Akhound-Sadegh, Viktor Ohanesian, Roberto Bondesan, Alan Aspuru-Guzik, Arnaud Doucet, Rob Brekelmans, Alexander Tong, and Kirill Neklyudov. Feynman-kac correctors in diffusion: Annealing, guidance, and product of experts. In *Forty-second international conference on machine learning*, 2025a.

Marta Skreta, Lazar Atanackovic, Joey Bose, Alexander Tong, and Kirill Neklyudov. The superposition of diffusion models using the itô density estimator. In *The thirteenth international conference on learning representations*, 2025b.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International conference on learning representations*, 2021a.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-Based Generative Modeling through Stochastic Differential Equations. In *International Conference on Learning Representations*, 2021b.

James Thornton, Louis Bethune, Ruixiang Zhang, Arwen Bradley, Preetum Nakkiran, and Shuangfei Zhai. Composition and control with distilled energy diffusion models and sequential monte carlo. In *AISTATS 2025*, 2025.

NLP Town. bert-base-multilingual-uncased-sentiment (Revision edd66ab), 2023. URL https://huggingface.co/nlptown/bert-base-multilingual-uncased-sentiment.

Masatoshi Uehara, Yulai Zhao, Chenyu Wang, Xiner Li, Aviv Regev, Sergey Levine, and Tommaso Biancalani. Inference-time alignment in diffusion models with reward-guided generation: Tutorial and review, 2025. arXiv: 2501.09685 [cs.AI].

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: System demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics.

Luhuan Wu, Brian Trippe, Christian Naesseth, David Blei, and John P Cunningham. Practical and asymptotically exact conditional sampling in diffusion models. *Advances in Neural Information Processing Systems*, 36:31372–31403, 2023.

Zhihan Yang, Wei Guo, Shuibai Zhang, Subham Sekhar Sahoo, Yongxin Chen, Arash Vahdat, Morteza Mardani, and John Thickstun. Continuous diffusion scales competitively with discrete diffusion for language, 2026. arXiv: 2605.18530 [cs.CL].

Hanlin Yu, Arto Klami, Aapo Hyvarinen, Anna Korba, and Omar Chehab. Density Ratio Estimation with Conditional Probability Paths. In *Forty-second International Conference on Machine Learning*, 2025.

Hanlin Yu, RuiKang OuYang, Partha Kaushik, Arto Klami, Michael U. Gutmann, and Omar Chehab. Learning energy-based models from stochastic interpolants using spatiotemporal differences, 2026. arXiv: 2605.26850 [cs.LG].

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, volume 28. Curran Associates, Inc., 2015.

APPENDIX

The appendix is organized as follows.

A Transformation for practical implementations **16**
A.1 Transformation of velocity and density 16
A.2 Transformation of SDE 17

B Theoretical results **19**
B.1 FATE 19
B.2 FATE+ 23
B.3 SuperDiff 26
B.4 Sequential Monte Carlo 27

C Experimental details **28**
C.1 Density estimation 28
C.2 Efficiency and stochasticity 28
C.3 Synthetic SMC 28
C.4 Composition 29

D Additional experimental results **30**
D.1 Density estimation 30
D.2 Efficiency and stochasticity 30
D.3 Synthetic SMC 30
D.4 Tempering 30
D.5 Composition 30

A TRANSFORMATION FOR PRACTICAL IMPLEMENTATIONS

In this section, we introduce the transformation scheme as used by Song et al. (2021a); Chen et al. (2026) which simplifies the expression of flow matching ODE, present the expressions of the corresponding reverse SDEs and show how to track densities under the resulting transformed coordinates. Note that these transformations are applicable across different choices of α and β. In the following discussion, when applicable we may simplify the notations, e.g. we may use m_t to refer to $m_t(x_t)$.

We can choose to reparameterize t using the log noise-to-signal ratio η given by $\eta = \log\left(\beta_t^2/\alpha_t^2\right)$ (Chen et al., 2026), in which case the relevant quantities can be largely re-written through simply replacing t with η.

We further employ the following reparameterization from x, t to y and r, given by

$$y = \frac{x}{\beta_t}, \quad r = \frac{\alpha_t}{\beta_t}. \tag{16}$$

A.1 TRANSFORMATION OF VELOCITY AND DENSITY

Let us derive how to track density along the sampling trajectory in the transformed coordinates.

Recall that $x = \beta_t y$, as such

$$p_r(y) = \beta_t^{LD} p_t(x), \tag{17}$$
$$\log p_r(y) = LD \log\left(\beta_t\right) + \log p_t(x). \tag{18}$$

Velocity Denote the velocity under x, t coordinates as $v_t(x)$ and the velocity under y, r coordinates as $u_r(y)$. Using the chain rule, we have

$$\frac{\mathrm{d}y}{\mathrm{d}t} = \frac{1}{\beta_t}\frac{\mathrm{d}x}{\mathrm{d}t} - \frac{\dot{\beta}_t}{\beta_t^2}x = \frac{1}{\beta_t}\left(v_t(x) - \frac{\dot{\beta}_t}{\beta_t}x\right), \tag{19}$$

and

$$u_r(y) = \frac{\mathrm{d}y}{\mathrm{d}r} = \frac{\mathrm{d}y}{\mathrm{d}t}\frac{\mathrm{d}t}{\mathrm{d}r} = \frac{\beta_t v_t(x) - \dot{\beta}_t x}{\dot{\alpha}_t\beta_t - \alpha_t\dot{\beta}_t}. \tag{20}$$

Recall that the velocity induced by the denoiser is given by

$$v_t(x) = \frac{\dot{\beta}_t}{\beta_t}x + \left(\dot{\alpha}_t - \frac{\alpha_t\dot{\beta}_t}{\beta_t}\right)m_t(x). \tag{21}$$

Plugging in the expression, the velocity under the transformed coordinates simplify greatly, and is given by

$$u_r(y) = m_t(x). \tag{22}$$

Stein score Recall that under the original coordinates the Stein score is given by

$$\nabla_x \log p_t(x) = -\frac{x - \alpha_t m_t(x)}{\beta_t^2}. \tag{23}$$

Under the transformation, we have

$$\nabla_y \log p_r(y) = \frac{\partial x}{\partial y}\nabla_x \log p_t(x) = -\beta_t\frac{x - \alpha_t m_t}{\beta_t^2} = -\frac{\beta_t y - \alpha_t m_t}{\beta_t} = r m_t - y. \tag{24}$$

Time score Recall that in the original coordinate the time score is given by

$$\partial_t \log p_t(x) = -LD\frac{\dot{\beta}_t}{\beta_t} + \frac{\dot{\beta}_t}{\beta_t^3}\|x\|^2 + \frac{1}{\beta_t^2}\left(\dot{\alpha}_t - 2\frac{\alpha_t\dot{\beta}_t}{\beta_t}\right)x^\top m_t(x) + \frac{\alpha_t}{\beta_t^2}\left(\frac{\alpha_t\dot{\beta}_t}{\beta_t} - \dot{\alpha}_t\right)n_t(x). \tag{25}$$

Under the transformation, we have

$$\frac{\partial}{\partial r}\log p_r(y) = \frac{\partial t}{\partial r}\frac{\partial}{\partial t}\log p_r(y), \tag{26}$$

where

$$\frac{\partial r}{\partial t} = \frac{\dot{\alpha}_t\beta_t - \alpha_t\dot{\beta}_t}{\beta_t^2}, \tag{27}$$

$$\frac{\partial t}{\partial r} = 1 \Big/ \frac{\partial r}{\partial t}\ . \tag{28}$$

Observe that

$$\frac{\partial}{\partial t}\log p_r(y) = \partial_t \log p_t(x) + \nabla_x \log p_t(x)^\top \dot{x} + LD\frac{\dot{\beta}_t}{\beta_t}. \tag{29}$$

Plugging in the expressions of $\partial_t \log p_t(x)$ and $\nabla_x \log p_t(x)^\top$ and simplifying, we have

$$\partial_r \log p_r(y) = y^\top m_t - r n_t. \tag{30}$$

Total time derivative along learned velocity We can use the chain rule and plug in the expressions of the scores:

$$\mathrm{d}_r \log p_r(y) = \partial_r \log p_r(y) + \nabla_y \log p_r(y)^\top m = -r\left(n - m^\top m\right). \tag{31}$$

Alternatively, this can be derived directly from the expression of the total time derivative under the original coordinates using change of variables. We have

$$\mathrm{d}_r \log p_r(y_r) = \frac{1}{\dot{r}}\left(LD\frac{\dot{\beta}_t}{\beta_t} + \mathrm{d}_t \log p_t(x_t)\right), \tag{32}$$

where

$$\dot{r} = \frac{\dot{\alpha}_t\beta_t - \alpha_t\dot{\beta}_t}{\beta_t^2}. \tag{33}$$

A.2 Transformation of SDE

SDE with diffusion convention of time Using $\bar{t}$ to denote time following diffusion convention, the forward SDE corresponding to the variance-preserving interpolation employed by Chen et al. (2026) is given by

$$\mathrm{d}x_{\bar{t}} = -\frac{1}{2}b_{\bar{t}}x_{\bar{t}}\mathrm{d}\bar{t} + \sqrt{b_{\bar{t}}}\mathrm{d}W_{\bar{t}}, \tag{34}$$

and the reverse SDE is given by

$$\mathrm{d}x_{\bar{t}} = \left[-\frac{1}{2}b_{\bar{t}}x_{\bar{t}} - b_{\bar{t}}\nabla_{x_{\bar{t}}}\log p_{\bar{t}}(x_{\bar{t}})\right]\mathrm{d}\bar{t} + \sqrt{b_{\bar{t}}}\mathrm{d}\overline{W}_{\bar{t}}, \tag{35}$$

where $b(\bar{t})$ is SDE coefficient.

SDE reparameterized using log noise-to-signal ratio One can show that $\frac{\partial \bar{t}}{\partial \eta} = \frac{\beta_{\bar{t}}^2}{b_{\bar{t}}}$.

As such, the forward SDE is given as

$$\mathrm{d}x_\eta = -\frac{1}{2}\beta_\eta^2 x_\eta \mathrm{d}\eta + \beta_\eta \mathrm{d}W_\eta. \tag{36}$$

The reverse SDE is given as

$$\mathrm{d}x_\eta = \left[-\frac{1}{2}\beta_\eta^2 x_\eta - \beta_\eta^2\nabla_{x_\eta}\log p_\eta(x_\eta)\right]\mathrm{d}\eta + \beta_\eta\mathrm{d}\overline{W}_\eta \tag{37}$$

$$= \left[\left(1 - \frac{1}{2}\beta_\eta^2\right)x_\eta - \alpha_\eta m_\eta(x_\eta)\right]\mathrm{d}\eta + \beta_\eta\mathrm{d}\overline{W}_\eta, \tag{38}$$

where we express the Stein score using the denoiser.

SDE under transformed coordinates We next derive what the resulting SDEs are after the transformation from x, η to y and r. Noting that $\mathrm{d}y = \frac{1}{\beta_\eta}\mathrm{d}x - \frac{x\dot{\beta}_\eta}{\beta_\eta^2}\mathrm{d}\eta$ and $\mathrm{d}\eta = -\frac{2}{r}\mathrm{d}r$, the forward SDE is given by

$$\mathrm{d}y_r = \frac{y_r}{r}\mathrm{d}r + \sqrt{\frac{2}{r}}\mathrm{d}W_r. \tag{39}$$

The reverse SDE is given by

$$\mathrm{d}y_r = \left[2m_r(y_r) - \frac{y_r}{r}\right]\mathrm{d}r + \sqrt{\frac{2}{r}}\mathrm{d}\overline{W}_r. \tag{40}$$

B THEORETICAL RESULTS

B.1 FATE

B.1.1 PROPOSITION AND PROOF

Proposition 4 (FATE estimator) *Consider two velocity fields $v_t, \tilde{v}_t$, with x_t and $\tilde{x}_t$ the associated solutions of the respective ODEs . Denote p_t the marginal distribution of x_t. Using the chain rule:*

$$\mathrm{d}_t \log p_t(\tilde{x}_t) = \partial_t \log p_t(\tilde{x}_t) + \nabla_x \log p_t(\tilde{x}_t)^\top \tilde{v}_t(\tilde{x}_t). \tag{41}$$

Let us denote $m_t(x) := \mathbb{E}[x_1 | x_t = x]$ and $n_t(x) := \mathbb{E}\left[\|x_1\|^2 | x_t = x\right]$. For any $x \in \mathbb{R}^{LD}$ and $t \in [0,1]$, one has (naturally) $\nabla_x \log p_t(x) = -\frac{x - \alpha_t m_t(x)}{\beta_t^2}$, and also

$$\partial_t \log p_t(x) = -LD\frac{\dot{\beta}_t}{\beta_t} + \frac{\dot{\beta}_t}{\beta_t^3}\|x\|^2 + \frac{1}{\beta_t^2}\left(\dot{\alpha}_t - 2\frac{\alpha_t\dot{\beta}_t}{\beta_t}\right) x^\top m_t(x) + \frac{\alpha_t}{\beta_t^2}\left(\frac{\alpha_t\dot{\beta}_t}{\beta_t} - \dot{\alpha}_t\right) n_t(x). \tag{42}$$

As such, both scores are closed-form expressions of $m_t(x)$ and $n_t(x)$ that can be computed analytically given a pre-trained model of $p_t(x_1^i | x_t)$. In particular, when $\tilde{v}_t \equiv v_t$ (and thus $\tilde{x}_t \equiv x_t$),

$$\mathrm{d}_t \log p_t(x_t) = -LD\frac{\dot{\beta}_t}{\beta_t} + \left(\frac{\alpha_t^2\dot{\beta}_t}{\beta_t^3} - \frac{\alpha_t\dot{\alpha}_t}{\beta_t^2}\right)\left(n_t(x_t) - \|m_t(x_t)\|^2\right). \tag{43}$$

Proof. Recall that by the chain rule, the total derivative along a trajectory $\tilde{x}_t$ defined by $\dot{\tilde{x}}_t = \tilde{v}_t(\tilde{x}_t)$ with a velocity $\tilde{v}_t$ expands as

$$\mathrm{d}_t \log p_t(\tilde{x}_t) = \partial_t \log p_t(\tilde{x}_t) + \nabla_x \log p_t(\tilde{x}_t)^\top \tilde{v}_t(\tilde{x}_t). \tag{44}$$

In the following, we derive closed-form expressions for the space score $\nabla_x \log p_t(\cdot)$ and the time score $\partial_t \log p_t(\cdot)$, as well as a closed-form expression of the total time derivative along the trajectory of x_t.

1st step: Closed-form expression for the space score.

This is a direct consequence of Tweedie's formula as for any $x \in \mathbb{R}^{DL}$ and for any $t < 1$, one has

$$\nabla_x \log p_t(x) = -\frac{x - \alpha_t \mathbb{E}[x_1 \mid x_t = x]}{\beta_t^2}. \tag{45}$$

2nd step: Closed-form expression for the time score.

Let us show that for any $x \in \mathbb{R}^{LD}$ and any $t < 1$, one has:

$$\begin{aligned}\partial_t \log p_t(x) = &-LD\frac{\dot{\beta}_t}{\beta_t} + \frac{\dot{\beta}_t}{\beta_t^3}\|x\|^2 + \frac{1}{\beta_t^2}\left(\dot{\alpha}_t - 2\frac{\alpha_t\dot{\beta}_t}{\beta_t}\right) x^\top \mathbb{E}[x_1 | x_t = x] \\ &+ \frac{\alpha_t}{\beta_t^2}\left(\frac{\alpha_t\dot{\beta}_t}{\beta_t} - \dot{\alpha}_t\right)\mathbb{E}\left[\|x_1\|^2 | x_t = x\right]\end{aligned} \tag{46}$$

Our derivation is similar to Yu et al. (2025) and Guth et al. (2025), relying on the Conditional Time Score Matching identity. One has

$$\begin{aligned}\partial_t \log p_t(x) &= \mathbb{E}_{p_t(x_1 | x_t = x)}\left[\partial_t \log p_t(x | x_1)\right] \\ &= \sum_{i=1}^{L} \mathbb{E}_{p_t(x_1^i | x_t = x)}\left[\partial_t \log p_t(x^i | x_1^i)\right],\end{aligned} \tag{47}$$

as $\partial_t \log p_t(x|x_1) = \sum_{i=1}^{L} \partial_t \log p_t(x^i | x_1^i)$, since the dimensions are conditionally independent given x_1 due to the diagonal isotropic covariance matrix $\beta_t^2 I$.

Note that

$$\log p_t(x^i|x_1^i) = \log \mathcal{N}\left(x^i; \alpha_t x_1^i, \beta_t^2 I_D\right) = -\frac{D}{2}\log\left(2\pi\beta_t^2\right) - \frac{\left\|x^i - \alpha_t x_1^i\right\|^2}{2\beta_t^2}. \tag{48}$$

Taking derivative with respect to t:

$$\partial_t \log p_t(x^i|x_1^i) = -D\frac{\dot\beta_t}{\beta_t} + \frac{\dot\beta_t}{\beta_t^3}\left\|x^i - \alpha_t x_1^i\right\|^2 + \frac{1}{\beta_t^2}\left(x^i - \alpha_t x_1^i\right)^\top \dot\alpha_t x_1^i. \tag{49}$$

Then as

$$\left\|x^i - \alpha_t x_1^i\right\|^2 = \left\|x^i\right\|^2 - 2\alpha_t\langle x^i, x_1^i\rangle + \alpha_t^2\left\|x_1^i\right\|^2, \tag{50}$$

$$(x^i - \alpha_t x_1^i)^\top x_1^i = \langle x^i, x_1^i\rangle - \alpha_t\left\|x_1^i\right\|^2, \tag{51}$$

one gets

$$\partial_t \log p_t(x^i|x_1^i) \tag{52}$$

$$\begin{aligned}
&= -D\frac{\dot\beta_t}{\beta_t} + \frac{\dot\beta_t}{\beta_t^3}\left\|x^i - \alpha_t x_1^i\right\|^2 + \frac{1}{\beta_t^2}\left(x^i - \alpha_t x_1^i\right)^\top \dot\alpha_t x_1^i \\
&= -D\frac{\dot\beta_t}{\beta_t} + \frac{\dot\beta_t}{\beta_t^3}\left(\left\|x^i\right\|^2 - 2\alpha_t\langle x^i, x_1^i\rangle + \alpha_t^2\left\|x_1^i\right\|^2\right) + \frac{1}{\beta_t^2}\dot\alpha_t\left(\langle x^i, x_1^i\rangle - \alpha_t\left\|x_1^i\right\|^2\right) \\
&= -D\frac{\dot\beta_t}{\beta_t} + \frac{\dot\beta_t}{\beta_t^3}\left\|x^i\right\|^2 + \frac{\dot\alpha_t\beta_t - 2\dot\beta_t\alpha_t}{\beta_t^3}\langle x^i, x_1^i\rangle + \frac{\alpha_t^2\dot\beta_t - \alpha_t\dot\alpha_t\beta_t}{\beta_t^3}\left\|x_1^i\right\|^2.
\end{aligned} \tag{53}$$

Then by plugging Eq. 52 in Eq. 47 one gets:

$$\begin{aligned}
&\partial_t \log p_t(x) \\
&= \sum_{i=1}^{L}\left(\mathbb{E}_{p_t(x_1^i|x_t=x)}\left[-D\frac{\dot\beta_t}{\beta_t} + \frac{\dot\beta_t}{\beta_t^3}\left\|x^i\right\|^2 + \frac{\dot\alpha_t\beta_t - 2\dot\beta_t\alpha_t}{\beta_t^3}\langle x^i, x_1^i\rangle + \frac{\alpha_t^2\dot\beta_t - \alpha_t\dot\alpha_t\beta_t}{\beta_t^3}\left\|x_1^i\right\|^2\right]\right) \\
&= -LD\frac{\dot\beta_t}{\beta_t} + \frac{\dot\beta_t}{\beta_t^3}\|x\|^2 + \sum_{i=1}^{d}\left(\frac{\dot\alpha_t\beta_t - 2\dot\beta_t\alpha_t}{\beta_t^3}\langle x^i, \mathbb{E}_{p_t(x_1^i|x_t=x)}\left[x_1^i\right]\rangle\right. \\
&\quad \left. + \frac{\alpha_t^2\dot\beta_t - \alpha_t\dot\alpha_t\beta_t}{\beta_t^3}\mathbb{E}_{p_t(x_1^i|x_t=x)}\left[\left\|x_1^i\right\|^2\right]\right) \\
&= -LD\frac{\dot\beta_t}{\beta_t} + \frac{\dot\beta_t}{\beta_t^3}\|x\|^2 + \frac{\dot\alpha_t\beta_t - 2\dot\beta_t\alpha_t}{\beta_t^3}x^\top\mathbb{E}\left[x_1 \mid x_t = x\right] + \frac{\alpha_t^2\dot\beta_t - \alpha_t\dot\alpha_t\beta_t}{\beta_t^3}\mathbb{E}\left[\|x_1\|^2|x_t = x\right] \\
&= -LD\frac{\dot\beta_t}{\beta_t} + \frac{\dot\beta_t}{\beta_t^3}\|x\|^2 + \frac{1}{\beta_t^2}\left(\dot\alpha_t - 2\frac{\alpha_t\dot\beta_t}{\beta_t}\right)x^\top\mathbb{E}\left[x_1 \mid x_t = x\right] \\
&\quad + \frac{\alpha_t}{\beta_t^2}\left(\frac{\alpha_t\dot\beta_t}{\beta_t} - \dot\alpha_t\right)\mathbb{E}\left[\|x_1\|^2 \mid x_t = x\right],
\end{aligned}$$

which gives Eq. 46 and completes this step.

Given the closed-form expression of $\nabla_{x_t}\log p_t(\cdot)$ ***and*** $\partial_t \log p_t(\cdot)$***, we can already estimate*** $\mathrm{d}_t \log p_t(\tilde x_t)$ ***using the chain rule as in Eq. 44 by evaluating the time and space scores in*** $\tilde x_t$ ***and using*** $\tilde v_t$***.***

In the following, we show that expressions simplify greatly when $\tilde v_t(x_t)$ is given by the learned velocity field $v_t(x_t)$.

3rd step: Closed-form expression for the total time derivative along the trajectory x_t. Now we consider $\tilde v_t \equiv v_t$, and thus $\tilde x_t \equiv x_t$. By the chain rule as in Eq. 44 and the first step, we only need to have a closed-form expression of $\nabla_x \log p_t(x_t)^\top v_t(x_t)$, and then sum this term with Eq. 46 evaluated in x_t

First, let us show that:

$$\nabla_{x_t} \log p_t(x_t)^\top v_t(x_t) = \frac{\dot\beta_t}{\beta_t^3}\|x_t\|^2 - \frac{1}{\beta_t^2}\left(\dot\alpha_t - 2\frac{\alpha_t\dot\beta_t}{\beta_t}\right) x_t^\top \mathbb{E}\left[x_1 \mid x_t\right] + \frac{\alpha_t}{\beta_t^2}\left(\dot\alpha_t - \frac{\alpha_t\dot\beta_t}{\beta_t}\right)\left\|\mathbb{E}\left[x_1 \mid x_t\right]\right\|^2. \tag{54}$$

As $x_t = \alpha_t x_1 + \beta_t x_0$ one has $x_0 = \frac{x_t - \alpha_t x_1}{\beta_t}$ and thus

$$\begin{aligned} v_t(x_t) &= \mathbb{E}\left[\dot\alpha_t x_1 + \dot\beta_t x_0 \mid x_t\right] \\ &= \mathbb{E}\left[\dot\alpha_t x_1 + \dot\beta_t \frac{x_t - \alpha_t x_1}{\beta_t} \mid x_t\right] \\ &= \mathbb{E}\left[\frac{\dot\beta_t}{\beta_t} x_t + \left(\dot\alpha_t - \frac{\alpha_t\dot\beta_t}{\beta_t}\right) x_1 \mid x_t\right] \\ &= \frac{\dot\beta_t}{\beta_t} x_t + \left(\dot\alpha_t - \frac{\alpha_t\dot\beta_t}{\beta_t}\right)\mathbb{E}\left[x_1 \mid x_t\right]. \end{aligned} \tag{55}$$

Moreover, by Tweedie's formula one has

$$\nabla_{x_t} \log p_t(\tilde x_t) = -\frac{\tilde x_t - \alpha_t \mathbb{E}\left[x_1 \mid \tilde x_t\right]}{\beta_t^2}. \tag{56}$$

Thus by combining Eq. 55 and Eq. 56 one gets

$$v_t(x_t)^\top \nabla_{x_t} \log p_t(x_t) \tag{57}$$

$$\begin{aligned} &= \left(\frac{\dot\beta_t}{\beta_t} x_t + \left(\dot\alpha_t - \frac{\alpha_t\dot\beta_t}{\beta_t}\right)\mathbb{E}\left[x_1 | x_t\right]\right)^\top \left(-\frac{x_t - \alpha_t \mathbb{E}\left[x_1 \mid x_t\right]}{\beta_t^2}\right) \\ &= -\frac{\dot\beta_t x_t^\top \left(x_t - \alpha_t \mathbb{E}\left[x_1 \mid x_t\right]\right)}{\beta_t^3} - \frac{\left(\dot\alpha_t\beta_t - \alpha_t\dot\beta_t\right)\mathbb{E}\left[x_1 \mid x_t\right]^\top \left(x_t - \alpha_t \mathbb{E}\left[x_1 \mid x_t\right]\right)}{\beta_t^3} \\ &= -\frac{\dot\beta_t}{\beta_t^3} x_t^\top x_t + x_t^\top \mathbb{E}\left[x_1 \mid x_t\right]\frac{2\alpha_t\dot\beta_t - \dot\alpha_t\beta_t}{\beta_t^3} + \mathbb{E}\left[x_1 \mid x_t\right]^\top \mathbb{E}\left[x_1 \mid x_t\right]\frac{\alpha_t}{\beta_t^3}\left(\dot\alpha_t\beta_t - \alpha_t\dot\beta_t\right) \\ &= -\frac{\dot\beta_t}{\beta_t^3}\|x_t\|^2 - \frac{1}{\beta_t^2}\left(\dot\alpha_t - 2\frac{\alpha_t\dot\beta_t}{\beta_t}\right) x_t^\top \mathbb{E}\left[x_1 \mid x_t\right] + \frac{\alpha_t}{\beta_t^2}\left(\dot\alpha_t - \frac{\alpha_t\dot\beta_t}{\beta_t}\right)\left\|\mathbb{E}\left[x_1 \mid x_t\right]\right\|^2, \end{aligned} \tag{58}$$

which gives Eq. 54.

Finally let us sum Eq. 54 with Eq. 46 to get the total time derivative along x_t.

One has

$$\mathrm{d}_t \log p_t(x_t) \tag{59}$$

$$\begin{aligned} &= \partial_t \log p_t(x_t) + \nabla_{x_t} \log p_t(x_t)^\top v_t(x_t) \\ &= -LD\frac{\dot\beta_t}{\beta_t} + \frac{\dot\beta_t}{\beta_t^3}\|x_t\|^2 + \frac{1}{\beta_t^2}\left(\dot\alpha_t - 2\frac{\alpha_t\dot\beta_t}{\beta_t}\right) x_t^\top \mathbb{E}\left[x_1 \mid x_t\right] + \frac{\alpha_t}{\beta_t^2}\left(\frac{\alpha_t\dot\beta_t}{\beta_t} - \dot\alpha_t\right)\mathbb{E}\left[\|x_1\|^2 | x_t\right] \\ &\quad - \frac{\dot\beta_t}{\beta_t^3}\|x_t\|^2 - \frac{1}{\beta_t^2}\left(\dot\alpha_t - 2\frac{\alpha_t\dot\beta_t}{\beta_t}\right) x_t^\top \mathbb{E}\left[x_1 \mid x_t\right] + \frac{\alpha_t}{\beta_t^2}\left(\dot\alpha_t - \frac{\alpha_t\dot\beta_t}{\beta_t}\right)\left\|\mathbb{E}\left[x_1 \mid x_t\right]\right\|^2 \\ &= -LD\frac{\dot\beta_t}{\beta_t} + \frac{\alpha_t}{\beta_t^2}\left(\dot\alpha_t - \frac{\alpha_t\dot\beta_t}{\beta_t}\right)\left(\left\|\mathbb{E}\left[x_1 \mid x_t\right]\right\|^2 - \mathbb{E}\left[\|x_1\|^2 | x_t\right]\right) \\ &= -LD\frac{\dot\beta_t}{\beta_t} + \left(\frac{\alpha_t^2\dot\beta_t}{\beta_t^3} - \frac{\alpha_t\dot\alpha_t}{\beta_t^2}\right)\left(\mathbb{E}\left[\|x_1\|^2 \mid x_t\right] - \left\|\mathbb{E}\left[x_1 \mid x_t\right]\right\|^2\right), \end{aligned} \tag{60}$$

which completes the proof. □

Connection to Skreta et al. (2025b) Skreta et al. (2025b) noted the following identity when the particle $\tilde{x}_t$ moves along velocity $v_t(\tilde{x}_t)$

$$\mathrm{d}_t \log p_t(\tilde{x}_t) = -\nabla_{\tilde{x}_t} \cdot v_t(\tilde{x}_t) - \langle \nabla_{\tilde{x}_t} \log p_t(\tilde{x}_t), v_t(\tilde{x}_t) - \tilde{v}_t(\tilde{x}_t) \rangle. \tag{61}$$

This can be recovered from Eq. 41 by noting that

$$-\nabla_{\tilde{x}_t} \cdot v_t(\tilde{x}_t) = \partial_t \log p_t(\tilde{x}_t) + \nabla_x \log p_t(\tilde{x}_t)^\top v_t(\tilde{x}_t). \tag{62}$$

B.1.2 Alternative proof for Eq. 43 using higher order score matching

We now highlight a connection to higher order score matching (Meng et al., 2021; OuYang et al., 2026b), which allows us to recover the formula of the total time derivative as in Eq. 43.

Proof. Note that

$$v_t(x_t) = \frac{\dot{\beta}_t}{\beta_t} x_t + \left(\dot{\alpha}_t - \frac{\alpha_t \dot{\beta}_t}{\beta_t} \right) \mathbb{E}\left[x_1 \mid x_t \right], \tag{63}$$

$$\nabla_x \log p_t(x_t) = -\frac{x_t - \alpha_t \mathbb{E}\left[x_1 \mid x_t \right]}{\beta_t^2} = \frac{\alpha_t \mathbb{E}\left[x_1 \mid x_t \right] - x_t}{\beta_t^2}, \tag{64}$$

as such

$$\mathbb{E}\left[x_1 \mid x_t \right] = \frac{x_t + \beta_t^2 \nabla_x \log p_t(x_t)}{\alpha_t}, \tag{65}$$

and

$$v_t(x_t) = \frac{\dot{\beta}_t}{\beta_t} x_t + \left(\dot{\alpha}_t - \frac{\alpha_t \dot{\beta}_t}{\beta_t} \right) \frac{x_t + \beta_t^2 \nabla_x \log p_t(x_t)}{\alpha_t} = \frac{\dot{\alpha}_t}{\alpha_t} x_t + \beta_t^2 \left(\frac{\dot{\alpha}_t}{\alpha_t} - \frac{\dot{\beta}_t}{\beta_t} \right) \nabla_x \log p_t(x_t). \tag{66}$$

Computing the divergence, we have

$$\nabla_x \cdot v_t(x_t) = LD \frac{\dot{\alpha}_t}{\alpha_t} + \beta_t^2 \left(\frac{\dot{\alpha}_t}{\alpha_t} - \frac{\dot{\beta}_t}{\beta_t} \right) \mathrm{Tr}\left(\nabla_x^2 \log p_t(x_t) \right). \tag{67}$$

Introduce the simplifying notation that $s_t = \nabla_x \log p_t(x_t)$ and $H_t = \nabla_x^2 \log p_t(x_t)$. Observe that the stochastic interpolant is given by $x_t = \alpha_t x_1 + \beta_t x_0 = (\alpha_t x_1) + \beta_t x_0$, i.e. it can be interpreted as adding Gaussian noise of covariance $\beta_t^2 I$ to scaled data $\alpha_t x_1$. Applying Theorem 1 from Meng et al. (2021), we have

$$\mathbb{E}\left[\alpha_t^2 x_1 x_1^\top \mid x_t \right] = x_t x_t^\top + \beta_t^2 \left(x_t s_t^\top + s_t x_t^\top \right) + \beta_t^4 H_t + \beta_t^4 s_t s_t^\top + \beta_t^2 I. \tag{68}$$

Taking the trace, we have

$$\alpha_t^2 \mathbb{E}\left[\|x_1\|^2 \mid x_t \right] = \|x_t\|^2 + 2\beta_t^2 x_t^\top s_t + \beta_t^4 \mathrm{Tr}\left(H_t \right) + \beta_t^4 \|s_t\|^2 + \beta_t^2 LD, \tag{69}$$

$$= \left\| x_t + \beta_t^2 s_t \right\|^2 + \beta_t^4 \mathrm{Tr}\left(H_t \right) + \beta_t^2 LD, \tag{70}$$

$$= \alpha_t^2 \|\mathbb{E}\left[x_1 \mid x_t \right]\|^2 + \beta_t^4 \mathrm{Tr}\left(H_t \right) + \beta_t^2 LD, \tag{71}$$

$$\mathrm{Tr}\left(H_t \right) = \frac{\alpha_t^2}{\beta_t^4} \mathbb{E}\left[\|x_1\|^2 \mid x_t \right] - \frac{\alpha_t^2}{\beta_t^4} \|\mathbb{E}\left[x_1 \mid x_t \right]\|^2 - \frac{1}{\beta_t^2} LD, \tag{72}$$

by using Eq. 66 for the last equality. Plugging the expression into Eq. 67, we have

$$\nabla_x \cdot v_t(x_t) = LD \frac{\dot{\alpha}_t}{\alpha_t} + \beta_t^2 \left(\frac{\dot{\alpha}_t}{\alpha_t} - \frac{\dot{\beta}_t}{\beta_t} \right) \left(\frac{\alpha_t^2}{\beta_t^4} \mathbb{E}\left[\|x_1\|^2 \mid x_t \right] - \frac{\alpha_t^2}{\beta_t^4} \|\mathbb{E}\left[x_1 \mid x_t \right]\|^2 - \frac{1}{\beta_t^2} LD \right) \tag{73}$$

$$= LD \frac{\dot{\beta}_t}{\beta_t} + \frac{\alpha_t \dot{\alpha}_t \beta_t - \alpha_t^2 \dot{\beta}_t}{\beta_t^3} \left(\mathbb{E}\left[\|x_1\|^2 \mid x_t \right] - \|\mathbb{E}\left[x_1 \mid x_t \right]\|^2 \right). \tag{74}$$

Thus

$$\mathrm{d}_t \log p_t(x_t) = -\nabla_x \cdot v_t(x_t) = -LD \frac{\dot{\beta}_t}{\beta_t} + \frac{\alpha_t^2 \dot{\beta}_t - \alpha_t \dot{\alpha}_t \beta_t}{\beta_t^3} \left(\mathbb{E}\left[\|x_1\|^2 \mid x_t \right] - \|\mathbb{E}\left[x_1 \mid x_t \right]\|^2 \right),$$

i.e. we recover the formula from Eq. 43. □

B.2 FATE+

B.2.1 Proof of Proposition 3

Proposition 3 *[FATE+] Consider the combination of N flow language models. The ith model provides an estimate of the denoiser, given by $\tilde{m}_t^{(i)}(x_t)$. The Stein score of the ith model used to form the velocity is scaled by c^i. The final velocity is given by*

$$v_t(x_t) = \left(\left(1 - \sum_{i=1}^{N} c_i\right)\frac{\dot{\alpha}_t}{\alpha_t} + \sum_{i=1}^{N} c_i \frac{\dot{\beta}_t}{\beta_t}\right) x_t + \left(\dot{\alpha}_t - \frac{\alpha_t \dot{\beta}_t}{\beta_t}\right) \sum_{i=1}^{N} c_i \tilde{m}_t^{(i)}(x_t), \tag{5}$$

and the total time derivative given by the negative divergence can be evaluated as

$$\mathrm{d}_t \log p_t(x_t) = -LD\left(\left(1 - \sum_{i=1}^{N} c_i\right)\frac{\dot{\alpha}_t}{\alpha_t} + \sum_{i=1}^{N} c_i \frac{\dot{\beta}_t}{\beta_t}\right) + \left(\frac{\alpha_t \dot{\beta}_t}{\beta_t} - \dot{\alpha}_t\right) \sum_{i=1}^{N} c_i \nabla_x \cdot \tilde{m}_t^{(i)}(x_t). \tag{6}$$

Proof. Consider the composition of scores

$$\widetilde{\nabla \log p_t^{(i)}(x_t)} = c_i \frac{\alpha_t m_t^{(i)}(x_t) - x_t}{\beta_t^2} \tag{75}$$

$$\nabla \log p_t(x_t) = \sum_{i=1}^{N} \widetilde{\nabla \log p_t^{(i)}(x_t)} = \sum_{i=1}^{N} c_i \frac{\alpha_t m_t^{(i)}(x_t) - x_t}{\beta_t^2}. \tag{76}$$

We have

$$v_t(x_t) = \frac{\dot{\alpha}_t}{\alpha_t} x_t + \beta_t^2 \left(\frac{\dot{\alpha}_t}{\alpha_t} - \frac{\dot{\beta}_t}{\beta_t}\right) \nabla \log p_t(x_t) \tag{77}$$

$$= \left(\left(1 - \sum_i c_i\right)\frac{\dot{\alpha}_t}{\alpha_t} + \sum_i c_i \frac{\dot{\beta}_t}{\beta_t}\right) x_t + \left(\frac{\dot{\alpha}_t}{\alpha_t} - \frac{\dot{\beta}_t}{\beta_t}\right) \alpha \sum_i c_i m_t^{(i)}(x_t) \tag{78}$$

$$-\nabla_{x_t} \cdot v_t(x_t) = -LD\left(\left(1 - \sum_i c_i\right)\frac{\dot{\alpha}_t}{\alpha_t} + \sum_i c_i \frac{\dot{\beta}_t}{\beta_t}\right) + \frac{\alpha_t^2 \dot{\beta}_t - \dot{\alpha}_t \alpha_t \beta_t}{\beta_t^3} \sum_i c_i \nabla_{x_t} \cdot m_t^{(i)}(x_t). \tag{79}$$

□

We next derive how they behave under transformations of coordinates.

For velocity, recall that

$$u_r(y_r) = \frac{\beta_t v_t(x_t) - \dot{\beta}_t x_t}{\dot{\alpha}_t \beta_t - \alpha_t \dot{\beta}_t}, \tag{80}$$

as such

$$u_r(y_r) = \left(1 - \sum_i c_i\right)\frac{y_r}{r} + \sum_i c_i m_t^{(i)}(x_t). \tag{81}$$

For density, using

$$\mathrm{d}_r \log p_r(y) = \partial_r \log p_r(y) + \nabla_y \log p_r(y)^\top m = -r\left(n - m^\top m\right), \tag{82}$$

we have

$$\mathrm{d}_r \log p_r(y_r) = -LD\frac{1 - \sum_i c_i}{r} - \beta_t \sum_i c_i \nabla_{x_t} m_t^{(i)}(x_t). \tag{83}$$

B.2.2 SPECIFIC DENOISERS

Original denoiser Recall that

$$\mathrm{d}_t \log p_t(x_t) = -LD\frac{\dot{\beta}_t}{\beta_t} + \frac{\alpha_t^2\dot{\beta}_t - \alpha_t\dot{\alpha}_t\beta_t}{\beta_t^3}\left(n_t(x_t) - \|m_t(x_t)\|^2\right). \tag{84}$$

and

$$v_t(x_t) = \frac{\dot{\beta}_t}{\beta_t}x_t + \left(\dot{\alpha}_t - \frac{\alpha_t\dot{\beta}_t}{\beta_t}\right)m_t(x_t) \tag{85}$$

$$-\nabla_{x_t}\cdot v_t(x_t) = -LD\frac{\dot{\beta}_t}{\beta_t} - \left(\dot{\alpha}_t - \frac{\alpha_t\dot{\beta}_t}{\beta_t}\right)\nabla_{x_t}\cdot m_t(x_t). \tag{86}$$

Combining Eq. 84 and Eq. 86, we have

$$\nabla_{x_t}\cdot m_t(x_t) = \frac{\alpha_t}{\beta_t^2}\left(n_t(x_t) - \|m_t(x_t)\|^2\right). \tag{87}$$

Denoiser induced by class logits Recent works (Acquaviva, 2026) look into the effect of temperature on evaluations of flow language models. Instead of tempering the scores, the categorical logits used to form the denoisers are tempered:

$$p_{t,T}(x_1^i|x_t) = \mathrm{softmax}\left(\frac{\log p_t\left(x_1^i|x_t\right)}{T}\right), \tag{88}$$

and

$$p_{t,T}(x_1^i = k|x_t) = \frac{\exp\left(\log p_t(x_1^i = e_k|x_t)/T\right)}{\sum_{j=1}^V \exp\left(\log p_t(x_1^i = e_j|x_t)/T\right)}. \tag{89}$$

We would like to derive the expression of $\nabla_{x_t^i}\log p_{t,T}(x_1^i = e_k|x_t)$.

Using Bayes' rule,

$$\nabla_{x_t^i}\log p(x_t|x_1^i = e_k) = \frac{\alpha_t e_k - x_t^i}{\beta_t^2}, \tag{90}$$

$$\nabla_{x_t^i}\log p(x_t) = \frac{\alpha_t m_t(x_t)_i - x_t^i}{\beta_t^2}, \tag{91}$$

$$\nabla_{x_t^i}\log p_t(x_1^i = k|x_t) = \nabla_{x_t^i}\log p(x_t|x_1^i = e_k) - \nabla_{x_t^i}\log p(x_t) \tag{92}$$

$$= \frac{\alpha_t}{\beta_t^2}\left(e_k - m_t(x_t)_i\right), \tag{93}$$

and

$$\nabla_{x_t^i}\log p_{t,T}(x_1^i = e_k|x_t) = \frac{1}{T}\nabla_{x_t^i}\log p(x_1^i = e_k|x_t) \tag{94}$$

$$-\frac{1}{T}\sum_{j=1}^V p_{t,T}(x_1^i = e_j|x_t)\nabla_{z_i}\log p_t(x_1^i = e_j|x_t) \tag{95}$$

$$= \frac{\alpha_t}{T\beta_t^2}\left(e_k - m_{t,T}(x_t)_i\right), \tag{96}$$

$$\nabla_{x_1^i}p_{t,T}(x_1^i = e_k|x_t) = \frac{\alpha_t}{T\beta_t^2}p_{t,T}(x_1^i = e_k|x_t)\left(e_k - m_{t,T}(x_t)_i\right). \tag{97}$$

We have

$$m_{t,T}(x_t)_i = \sum_k p_{t,T}(x_1^i = e_k) e_k, \tag{98}$$

$$\nabla_{x_t^i} m_{t,T}(x_t)_i = \sum_k \nabla_{z_i} p_{t,T}(x_1^i = e_k | x_t) e_k^\top \tag{99}$$

$$= \frac{\alpha_t}{T\beta_t^2} \left(\sum_k p_{t,T}(x_1^i = e_k) e_k e_k^\top - m_t(x_t)_i m_t(x_t)_i^\top \right), \tag{100}$$

$$\nabla_{x_t^i} \cdot m_{t,T}(x_t) = \frac{\alpha_t}{T\beta_t^2} \left(n_{t,T}(x_t) - \|m_{t,T}(x_t)\|^2 \right). \tag{101}$$

As such, using tempered denoiser accounts to replacing $\left(n_t(x_t) - \|m_t(x_t)\|^2\right)$ with $\frac{1}{T}\left(n_{t,T}(x_t) - \|m_{t,T}(x_t)\|^2\right)$.

Denoiser induced by binary logits We next consider how the formulation of Batzolis et al. (2026) behaves under similar tempering as in Acquaviva (2026).

We use $\ell(x,t,i)$ to denote the logit given by $\log p_t(x_1^i = 1|x_t) - \log p_t(x_1^i = 0|x_t)$. We have

$$m_{t,T}(x_t)_i = \text{sigmoid}\left(\frac{\ell(x,t,i)}{T}\right), \tag{102}$$

$$\nabla_{x_t^i} m_{t,T}(x_t)_i = \text{sigmoid}\left(\frac{\ell(x,t,i)}{T}\right)\left(1 - \text{sigmoid}\left(\frac{\ell(x,t,i)}{T}\right)\right)\frac{1}{T}\nabla_{x_1^i}\log \ell(x,t,i), \tag{103}$$

$$\nabla_x \cdot m_{t,T}(x_t) = \frac{1}{T}\sum_i m_{t,T}(x_t)_i \left(1 - m_{t,T}(x_t)_i\right) \nabla_{x_t^i} \ell(x,t,i). \tag{104}$$

Using Bayes' rule and use $-i$ to denote all entries apart from the ith entry, we have

$$\frac{p(x_1^i = 1|x_t)}{p(x_1^i = 0|x_t)} \tag{105}$$

$$= \frac{p(x_t|x_1^i = 1)p(x_1^i = 1)}{p(x_t|x_1^i = 0)p(x_1^i = 0)} \tag{106}$$

$$= \frac{p(x_t^i|x_1^i = 1)p(x_t^{-i}|x_1^i = 1)p(x_1^i = 1)}{p(x_t^i|x_1^i = 0)p(x_t^{-i}|x_1^i = 0)p(x_1^i = 0)} \tag{107}$$

$$= \frac{p(x_t^i|x_1^i = 1)p(x_1^i = 1|x_t^{-i})}{p(x_t^i|x_1^i = 0)p(x_1^i = 0|x_t^{-i})}. \tag{108}$$

As such,

$$\ell(x,t,i) \tag{109}$$

$$= \log p(x_1^i = 1|x_t) - \log p(x_1^i = 0|x_t) \tag{110}$$

$$= \left(\log p(x_t^i|x_1^i = 1) - \log p(x_t^i|x_1^i = 0)\right) + \left(\log p(x_1^i = 1|x_t^{-i}) - \log p(x_1^i = 0|x_t^{-i})\right) \tag{111}$$

Observe that the second term does not depend on x_i. When $x_t = \alpha_t x_1 + \beta_t \epsilon$, we have

$$\log p(x_t^i|x_1^i = 1) - \log p(x_t^i|x_1^i = 0) \tag{112}$$

$$= -\frac{\left(x_t^i - \alpha_t\right)^2}{2\beta_t^2} + \frac{\left(x_t^i\right)^2}{2\beta_t^2} \tag{113}$$

$$= \frac{\alpha_t}{\beta_t^2}\left(x_t^i - \frac{\alpha_t}{2}\right) \tag{114}$$

As such,

$$\nabla_{x_t^i} \ell(x,t,i) = \frac{\alpha_t}{\beta_t^2}, \tag{115}$$

and

$$\nabla_{x_t} \cdot m_t(x_t) = \frac{\alpha_t}{T\beta_t^2} \sum_{i=1}^{L} m_{t,T}(x_t)_i \left(1 - m_{t,T}(x_t)_i\right). \tag{116}$$

B.3 SUPERDIFF

Generally, we would like to use as velocity the combination of the velocities of the different models, which we write as

$$v_t(x) = \sum_i k^i v_t^i(x). \tag{117}$$

When there are two models, we instead consider $v_t(x) = kv_t^1(x) + (1-k)\, v_t^2(x)$.

B.3.1 LOGICAL OR

Following Skreta et al. (2025b), the weights can be evaluated as

$$k = \text{softmax}\left(\log p_t^{(i)}(x_t) + \log w_i\right). \tag{118}$$

When there are two models with equal weights, we have

$$k = \text{sigmoid}\left(\log p_t^{(1)}(x) - \log p_t^{(2)}(x)\right). \tag{119}$$

B.3.2 LOGICAL AND

Following Skreta et al. (2025b), we would like to draw samples that have equal densities under the different models. We achieve this by matching the infinitesimal change of densities of all models.

Under the transformed coordinates, the infinitesimal change of density for the ith model is given as

$$\mathrm{d}_r \log p_r^{(i)}(y_r) \tag{120}$$

$$= -r\left(n^{(i)} - \left\|m^{(i)}\right\|^2\right) + \left(rm^{(i)} - y\right)^\top \left(m - m^{(i)}\right) \tag{121}$$

$$= -r\left(n^{(i)} - \left\|m^{(i)}\right\|^2\right) + r\left(m^{(i)}\right)^\top \left(\sum_j k^{(j)} m^{(j)} - m^{(i)}\right) - y^\top \left(\sum_j k^{(j)} m^{(j)} - m^{(i)}\right) \tag{122}$$

$$= y^\top m^{(i)} - rn^i(i) + \sum_j \left(rm^{(i)} - y\right)^\top m^{(j)} k^{(j)}. \tag{123}$$

where we plug in $m = \sum_j k^{(j)} m^{(j)}$.

Similar to Skreta et al. (2025b), the changes of densities can be written as a linear transformation

$$\begin{bmatrix} \mathrm{d}\log p^1(x_t) \\ \mathrm{d}\log p^2(x_t) \ldots \mathrm{d}\log p^N(x_t) \end{bmatrix} = Ak + b, \tag{124}$$

where

$$A_{ij} = \left(rm^{(i)} - y\right)^\top m^{(j)}, \quad b_{(i)} = y^\top m^{(i)} - rn^{(i)}, \tag{125}$$

and the weights can be solved using the following linear equation

$$\begin{bmatrix} A & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} k \\ \mathrm{d}\log p \end{bmatrix} = \begin{bmatrix} -b \\ 1 \end{bmatrix}. \tag{126}$$

When there are two models,

$$k = \frac{n^{(1)} - n^{(2)} - \left(m^{(1)} - m^{(2)}\right)^\top m^{(2)} - \frac{1}{\alpha_t}\left(m^{(1)} - m^{(2)}\right)^\top x}{\left\|m^{(1)} - m^{(2)}\right\|^2}. \tag{127}$$

Algorithm 2 Sequential Monte Carlo (Dai et al., 2022; Naesseth et al., 2019). Iteration is subscript τ, particle index is superscript i.

Require: Number of particles N, unnormalized target distributions $\tilde{\gamma}_\tau(x_\tau)$, forward kernels $q_\tau(x_\tau|x_{\tau-1})$, backward kernels $s_{\tau-1}(x_{\tau-1}|x_\tau)$, initial particles $(x_0^i)_{i\in[N]}$ and weights $w_0^i = N^{-1}$.

1: **for** $\tau = 1$ to T **do**
2: Resample particles $x_{\tau-1}^i \leftarrow x_{\tau-1}^{a_{\tau-1}^i}$ where $a_{\tau-1}^i \sim \text{Multinom}(w_{\tau-1}^1, \ldots, w_{\tau-1}^N)$ for $i \in [N]$
3: Propose particles $x_\tau^i \sim q_\tau(x_\tau^i|x_{\tau-1}^i)$ for $i \in [N]$
4: Compute weights $w_\tau^i = \frac{\tilde{w}_\tau^i}{\sum_j \tilde{w}_\tau^j}$ where $\tilde{w}_\tau^i = \frac{\tilde{\gamma}_\tau(x_\tau^i) s_{\tau-1}(x_{\tau-1}^i|x_\tau^i)}{\tilde{\gamma}_{\tau-1}(x_{\tau-1}^i) q_\tau(x_\tau^i|x_{\tau-1}^i)}$ for $i \in [N]$
5: **end for**
6: **return** a particle-based approximation $\hat{\gamma}_\tau(\cdot) = \sum_{i=1}^N w_\tau^i \delta_{x_\tau^i}(\cdot)$ of the intermediate targets $\gamma_\tau(\cdot)$

B.4 Sequential Monte Carlo

First, we recall the Sequential Monte Carlo algorithm.

We next derive the simplified weights used in the Sequential Monte Carlo Algorithm to perform tempering as described in Section 4.2.

Tempering First, we recall the choices made for the Sequential Monte Carlo algorithm:

$$\tilde{\gamma}_\tau(x_\tau) = p_\tau(x_\tau)^{\lambda_\tau} \tag{128}$$

$$q_\tau(x_\tau|x_{\tau-1}) = p_{\tau|\tau-1}(x_\tau|x_{\tau-1}), \tag{129}$$

$$s_{\tau-1}(x_{\tau-1}|x_\tau) = p_{\tau-1|\tau}(x_{\tau-1}|x_\tau). \tag{130}$$

where $p_{\tau-1|\tau}(x_{\tau-1}|x_\tau)$ and $p_{\tau|\tau-1}(x_\tau|x_{\tau-1})$ are, respectively, the noising and denoising kernels associated with the stochastic interpolant, i.e., the forward and reverse conditionals of the same joint distribution induced by the interpolant, and λ_τ is the tempering schedule such that $0 < \lambda_1 < \ldots < \lambda_T = 1$.

With these choices, we can then simplify the formula for the weights:

$$\tilde{w}_\tau = \frac{\tilde{\gamma}_\tau(x_\tau) s_{\tau-1}(x_{\tau-1}|x_\tau)}{\tilde{\gamma}_{\tau-1}(x_{\tau-1}) q_\tau(x_\tau|x_{\tau-1})} = \frac{p_\tau(x_\tau)^{\lambda_\tau} p_{\tau-1|\tau}(x_{\tau-1}|x_\tau)}{p_{\tau-1}(x_{\tau-1})^{\lambda_{\tau-1}} p_{\tau|\tau-1}(x_\tau|x_{\tau-1})}. \tag{131}$$

Noting that by Bayes' rule:

$$p_{\tau-1}(x_{\tau-1}) p_{\tau|\tau-1}(x_\tau|x_{\tau-1}) = p_\tau(x_\tau) p_{\tau-1|\tau}(x_{\tau-1}|x_\tau), \tag{132}$$

we obtain

$$\tilde{w}_\tau = \frac{p_\tau(x_\tau)^{\lambda_\tau} p_{\tau-1}(x_{\tau-1})}{p_{\tau-1}(x_{\tau-1})^{\lambda_{\tau-1}} p_\tau(x_\tau)} = \frac{p_\tau(x_\tau)^{\lambda_\tau - 1}}{p_{\tau-1}(x_{\tau-1})^{\lambda_{\tau-1}-1}}. \tag{133}$$

C EXPERIMENTAL DETAILS

We always use the transformed coordinate system as described in Appendix A. The integration rules are derived under the transformed y, r coordinates, and mapped back to the original coordinates. Following previous work (Chen et al., 2026; Batzolis et al., 2026), the reported generative perplexity values are computed under GPT2-large (Radford et al., 2019) provided by Huggingface Transformers (Wolf et al., 2020).

C.1 DENSITY ESTIMATION

For discrete data, the natural density is the one of the discrete distribution. When we integrate the flow in continuous space, we evaluate the density of the continuous distribution. Fortunately, we can relate the continuous densities $p_t(x_t)$ and categorical densities $p_{\text{data}}(c)$ when the model integrates between $t \in [0, 1-\epsilon]$ for a small ϵ using the following bound (Chen et al., 2026; Yang et al., 2026)

$$\log p_{\text{data}}(c) \geq \mathbb{E}_{x_{1-\epsilon}|c}\left[\log p_0(x_0) + \int_{t=0}^{1-\epsilon} \mathrm{d}_s \log p_s(x_s) ds + \log p\left(r|x_{1-\epsilon}\right)\right] \tag{134}$$

$$- \mathbb{E}_{x_{1-\epsilon}|c}\left[\log p(x_{1-\epsilon}|r)\right]. \tag{135}$$

In practice, one may use slightly different time intervals, reparameterize t using the log noise-to-signal ratio $\log\left(\frac{\beta_t^2}{\alpha_t^2}\right)$ and use simplifying expression to express the divergence to be with respect to the denoiser. Following Chen et al. (2026), given a text, we use a single Monte Carlo sample of $x_{1-\epsilon}$ to evaluate the resulting bound on the discrete density, where the divergence is estimated either using the Hutchinson's trace estimator with Gaussian probes, as in Chen et al. (2026), or using our proposed estimator.

C.2 EFFICIENCY AND STOCHASTICITY

We evaluate the efficiency of our estimator in relation to Hutchinson's, using the pretrained models provided by Chen et al. (2026) and Batzolis et al. (2026) on LM1B and OWT, where the clean data is generated randomly and interpolated using the stochastic interpolant. Depending on the usage of self-conditioning, there are generally two variants, which we refer to as `same` and `previous`, respectively. Given x_t and t, `same` variant first computes a forward pass of the network with no self-conditioning to get the output, then use the resulting output as self-conditioning to feed the network. As such, it takes two network evaluations to get the final output; this variant mimics the computations when evaluating perplexities of the models. `Previous` variant instead caches the network's output given the previous input, and use it as self-conditioning given the current input; this variant mimics the computations when performing sampling.

We consider the following three estimates: `denoiser`, which refers to merely forming the denoiser by querying the model, `FATE`, which is our analytical estimator, and `Hutchinson`, which is using Hutchinson's estimator to estimate the divergence.

C.3 SYNTHETIC SMC

We always use 256 particles, where the SMC samplers have 4 outer transitions, each containing 40 inner GLASS ODE steps or 40 reverse SDE steps. We run 25 trials with seeds $1, 2, \ldots, 25$.

The default data distribution is given by

$$p_{\text{data}}(c) \propto \exp\left(0.4 \sum_{i=0}^{L-1} \cos\left(\frac{2\pi c_i}{V} - 0.7i\right) + 0.65 \sum_{i=0}^{L-2} \cos\left(\frac{2\pi\left(c_{i+1} - c_i\right)}{K}\right)\right). \tag{136}$$

We use one-hot embedding for each token to embed the input sequences into a continuous space, and adapt the same stochastic interpolant as Chen et al. (2026), directly using the log noise-to-signal ratio η as integration coordinates without using Gumbel CDF.

For tempering, we choose $\beta = 2$. For composition, we construct the second data distribution as

$$p_{\text{data}}^{(2)}(c) = \frac{1}{2} p_{\text{data}}(c) + \frac{1}{2} \text{flip last axis}\left(p_{\text{data}}(c)\right). \tag{137}$$

We compute total variation between the final weighted samples and the ground truth target distributions. The Effective Sample Sizes (ESSs) are computed as the mean of the effective sample sizes across the four resampling stages, where the ESS in the τ-th stage is given by

$$\mathrm{ESS}_j = \frac{1}{\sum_{i=1}^{N} \left(\tilde{w}_\tau^i\right)^2}. \tag{138}$$

Finally, we report the density error as the mean absolute difference of the estimated densities of the particles against the ground truth. For composition, the reported density error is the larger one out of the two data distributions.

For Reverse SDE, the marginal densities are tracked using RNE formula with standard normal reference (He et al., 2026). For GLASS (Hutchinson), as we use analytical denoiser, we emulate the behavior of Hutchinson's estimator by computing the Jacobian vector product and forming the resulting stochastic estimate, where the probes of Hutchinson's estimator are sampled from a standard normal distribution.

For composition, when we select one model to output the velocities, we still need to evaluate the change of density along the other model. These are straightforward to implement with Reverse SDE, where one evaluates the denoiser of the other model. For GLASS (Hutchinson) and GLASS (FATE), we use (Skreta et al., 2025b)

$$\mathrm{d}_t \log p_t(\tilde{x}_t) = \mathrm{d}_t \log p_t(x_t) + \nabla_{x_t} \log p_t(x_t)^\top \left(\tilde{v}_t(\tilde{x}_t) - v_t(x_t)\right), \tag{139}$$

where $\mathrm{d}_t \log p_t(x_t)$ is estimated either using Hutchinson's estimator or FATE.

C.4 Composition

For classifying texts generated using the models, we use pre-trained models on Huggingface Transformers (Wolf et al., 2020). For classifying AG news categories, we use `bert-base-uncased-ag-news` provided by Morris et al. (2020). For Amazon reviews, we use Town (2023) to classify sentiments and use Lewis et al. (2020) to classify product categories.

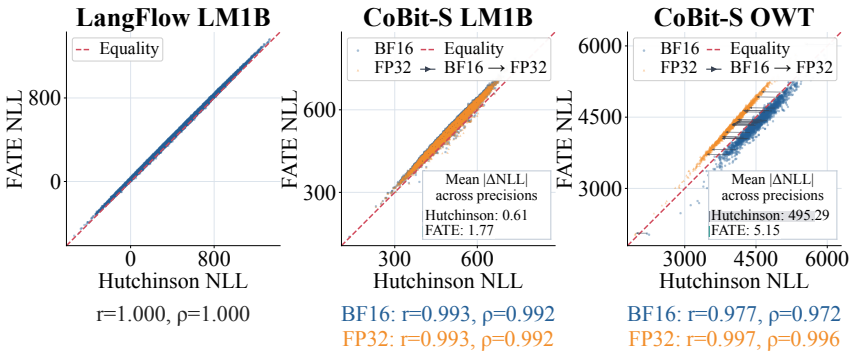


Figure 7: Density estimation results of LangFlow trained on LM1B, CoBit-S trained on LM1b and CoBit-S trained on OWT. The general observation is largely consistent with the main paper.

D Additional Experimental Results

D.1 Density Estimation

Here we report the density estimation results using CoBit-S trained on OWT. As shown in Figure 7, the density estimates behave rather similar to CoBit-M trained on OWT as reported in the main paper.

D.2 Efficiency and Stochasticity

In terms of efficiency, we report the runtime results in Table 1 and memory results in Table 2. For stochasticity, we report the results in Table 3.

D.3 Synthetic SMC

The Kernel Mixture composition proposal presented in the main paper considers transition kernels that are mixtures of denoising kernels of each model. An alternative is to make each individual transition a mixture of two steps, which is applicable for both Reverse SDE and GLASS, and we refer to this variant as Inner Kernel Mixture (IKM).

In the main paper, we reported composition results using KM variant. Here we additionally report results using AS variant and IKM variant in Figure 8. The relative comparisons between the different estimators remain largely unchanged.

We further report the results of SuperDiff AND and SuperDiff OR in Figure 9, comparing sampling using PF ODE and Reverse SDE. For SuperDiff AND, we report the average log-density differences between the two models as well as the average density estimation error. For SuperDiff OR, we report the TV distance to the ground truth mixture as well as the average density error. Similar to composition, the reported average density errors are the larger of the two models. For SuperDiff AND, PF ODE suffers from difficulties in estimating the densities, but results in smaller log-density gap between the two models than using Reverse SDE, the goal of the AND logic. For SuperDiff OR, PF ODE results in notably more accurate density estimates, while both PF ODE and Reverse SDE successfully capture the target mixture distribution.

D.4 Tempering

We report the tempering results in Table 4. Using a temperature that further deviates from 1.0 generally leads to larger mismatch between proposals and targets and smaller numbers of unique texts.

D.5 Composition

We report the composition results of finetuned models based on CoBit-S trained on LM1B in Table 5, based on CoBit-S trained on OWT in Table 6 and based on CoBit-M trained on OWT in Table 7.

Table 1: Runtime as measured as ms per position.

Model	Precision	Position	Denoiser	Hutchinson	FATE	Hutchinson/FATE
LangFlow LM1B	BF16	Previous	45.872	95.283	46.433	2.052
LangFlow LM1B	BF16	Same	91.683	141.127	92.266	1.530
LangFlow OWT	BF16	Previous	71.832	148.660	72.669	2.046
LangFlow OWT	BF16	Same	143.552	220.345	144.415	1.526
CoBit-S LM1B	BF16	Previous	84.560	235.906	84.565	2.790
CoBit-S LM1B	BF16	Same	169.403	321.019	169.285	1.896
CoBit-S LM1B	FP32	Previous	155.839	419.716	155.813	2.694
CoBit-S LM1B	FP32	Same	311.619	573.389	311.665	1.840
CoBit-S OWT	BF16	Previous	48.891	148.798	48.920	3.042
CoBit-S OWT	BF16	Same	97.818	197.803	97.843	2.022
CoBit-S OWT	FP32	Previous	150.393	449.728	150.471	2.989
CoBit-S OWT	FP32	Same	300.960	600.400	300.796	1.996
CoBit-M OWT	BF16	Previous	55.148	141.293	55.171	2.561
CoBit-M OWT	BF16	Same	110.190	196.383	110.234	1.782
CoBit-M OWT	FP32	Previous	144.427	388.158	144.496	2.686
CoBit-M OWT	FP32	Same	288.936	532.592	288.901	1.844

Table 2: Maximum incremental peak allocated GPU memory in MiB across runs.

Model	Precision	Position	Denoiser	Hutchinson	FATE	Hutchinson/FATE
LangFlow LM1B	BF16	Previous	6767.490	14103.468	6815.490	2.069
LangFlow LM1B	BF16	Same	6815.490	14151.792	6863.490	2.062
LangFlow OWT	BF16	Previous	11143.839	20351.623	11191.839	1.818
LangFlow OWT	BF16	Same	11191.839	20399.367	11239.839	1.815
CoBit-S LM1B	BF16	Previous	1920.809	28633.908	1920.809	14.907
CoBit-S LM1B	BF16	Same	1932.059	28633.908	1932.059	14.820
CoBit-S LM1B	FP32	Previous	3841.559	56810.732	3841.559	14.788
CoBit-S LM1B	FP32	Same	3856.559	56810.732	3856.559	14.731
CoBit-S OWT	BF16	Previous	1024.054	14541.740	1024.054	14.200
CoBit-S OWT	BF16	Same	1030.054	14541.740	1030.054	14.117
CoBit-S OWT	FP32	Previous	2048.101	28681.134	2048.101	14.004
CoBit-S OWT	FP32	Same	2056.101	28681.134	2056.101	13.949
CoBit-M OWT	BF16	Previous	305.621	9685.408	305.621	31.691
CoBit-M OWT	BF16	Same	307.121	9685.408	307.121	31.536
CoBit-M OWT	FP32	Previous	600.695	18178.946	600.695	30.263
CoBit-M OWT	FP32	Same	602.695	18178.946	602.695	30.163

Table 3: Percentages of the variability of Hutchinson measured as RMS sample standard deviation divided by RMS empirical mean as a function number of probes. FATE is deterministic by definition.

Model	Precision	Position	Hutchinson (1 probe)	Hutchinson (2 probes)	Hutchinson (4 probes)	Hutchinson (8 probes)	FATE
LangFlow LM1B	BF16	Previous	4.285	3.005	2.122	1.498	0
LangFlow LM1B	BF16	Same	4.786	3.270	2.356	1.641	0
LangFlow OWT	BF16	Previous	1.390	0.947	0.665	0.484	0
LangFlow OWT	BF16	Same	1.356	0.981	0.725	0.505	0
CoBit-S LM1B	BF16	Previous	3.901	2.712	1.916	1.343	0
CoBit-S LM1B	BF16	Same	4.073	2.820	1.952	1.358	0
CoBit-S LM1B	FP32	Previous	3.910	2.716	1.919	1.344	0
CoBit-S LM1B	FP32	Same	4.090	2.830	1.958	1.361	0
CoBit-S OWT	BF16	Previous	1.296	0.929	0.681	0.482	0
CoBit-S OWT	BF16	Same	1.592	1.092	0.768	0.532	0
CoBit-S OWT	FP32	Previous	1.298	0.926	0.679	0.479	0
CoBit-S OWT	FP32	Same	1.465	1.017	0.727	0.507	0
CoBit-M OWT	BF16	Previous	1.460	0.898	0.626	0.430	0
CoBit-M OWT	BF16	Same	1.520	1.022	0.708	0.484	0
CoBit-M OWT	FP32	Previous	1.398	0.883	0.617	0.426	0
CoBit-M OWT	FP32	Same	1.420	0.979	0.687	0.466	0

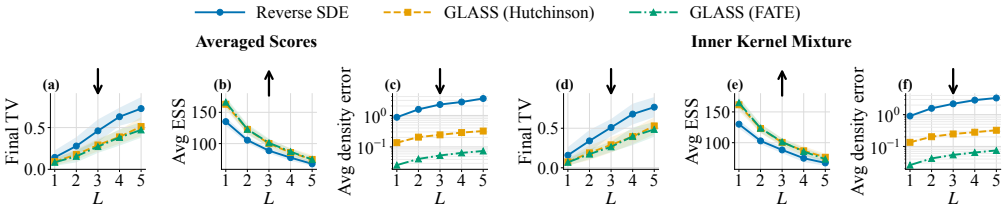


Figure 8: Results of Averaged Scores and Inner Kernel Mixture proposal kernels for composition. The overall conclusion is consistent with the main paper.

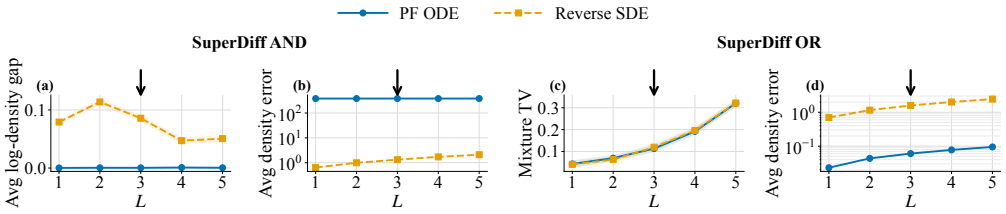


Figure 9: Results of SuperDiff AND and OR.

Table 4: Tempering results in the form of mean $\pm$ standard deviation over 5 runs with different random seeds. T=1 uses GLASS generation; other temperatures use SMC. Unique texts indicate the number of unique text samples obtained using SMC, and is not applicable for other samplers.

Model	Temperature	Sampler	Seeds	GenPPL	Entropy (nats)	Unique texts
CoBit-S LM1B	0.95	SMC	5	19.86 ± 6.90	3.803 ± 0.160	205.20 ± 14.38
CoBit-S LM1B	0.975	SMC	5	35.02 ± 5.49	4.034 ± 0.037	436.00 ± 13.25
CoBit-S LM1B	1	GLASS	5	71.51 ± 1.27	4.303 ± 0.005	–
CoBit-S LM1B	1.025	SMC	5	87.91 ± 3.22	4.340 ± 0.014	482.80 ± 14.29
CoBit-S LM1B	1.05	SMC	5	116.78 ± 10.60	4.353 ± 0.048	264.40 ± 15.44
CoBit-S OWT	0.95	SMC	5	5.38 ± 5.04	4.689 ± 0.357	13.20 ± 10.57
CoBit-S OWT	0.975	SMC	5	5.17 ± 4.99	4.336 ± 0.484	123.40 ± 47.03
CoBit-S OWT	1	GLASS	5	33.31 ± 0.53	5.108 ± 0.012	–
CoBit-S OWT	1.025	SMC	5	115.38 ± 19.16	5.286 ± 0.173	123.60 ± 26.50
CoBit-S OWT	1.05	SMC	5	216.21 ± 114.75	5.375 ± 0.100	20.60 ± 13.79
CoBit-M OWT	0.95	SMC	5	2.23 ± 1.95	4.149 ± 0.255	17.00 ± 17.59
CoBit-M OWT	0.975	SMC	5	1.24 ± 0.11	4.387 ± 0.212	57.60 ± 46.68
CoBit-M OWT	1	GLASS	5	32.67 ± 0.59	5.316 ± 0.007	–
CoBit-M OWT	1.025	SMC	5	233.51 ± 142.65	5.500 ± 0.033	122.60 ± 20.84
CoBit-M OWT	1.05	SMC	5	272.82 ± 152.36	5.535 ± 0.144	27.60 ± 15.79

SMC algorithms, despite consistency guarantees, here suffer from notable diversity issues, likely due to the lack of a principled proposal.

Table 5: Composition results of CoBit-S trained on LM1B in the form of mean ± standard deviation over 5 runs with different random seeds. Panel A reports sample quality metrics as well as the proportion of categories as identified using the classifiers. Unique texts indicate the number of unique text samples obtained using SMC, and is not applicable for other samplers. Panel B reports proportions of classifier probabilities.

A. Quality, diversity, and proportions of classifier predictions

Method	GenPPL	Entropy (nats)	Unique texts	World (%)	Sports (%)	Business (%)	Sci/Tech (%)
Expert 1 (World/Sports), GLASS	2.29 ± 0.02	2.01 ± 0.02	–	14.61 ± 0.92	35.47 ± 1.80	1.80 ± 0.29	48.12 ± 1.55
Expert 1 (World/Sports), PF ODE	2.40 ± 0.02	2.04 ± 0.02	–	11.60 ± 0.91	32.97 ± 1.61	1.58 ± 0.42	53.85 ± 1.08
Expert 2 (Business/Sports), GLASS	2.31 ± 0.02	2.00 ± 0.02	–	0.51 ± 0.17	33.18 ± 1.85	31.00 ± 2.76	35.31 ± 2.82
Expert 2 (Business/Sports), PF ODE	2.45 ± 0.02	2.06 ± 0.02	–	0.51 ± 0.22	32.29 ± 0.85	27.71 ± 0.77	39.49 ± 1.10
SMC: Kernel Mixture	1.94 ± 0.15	1.68 ± 0.30	1.80 ± 1.30	0.00 ± 0.00	40.00 ± 54.77	0.00 ± 0.00	60.00 ± 54.77
SMC: Averaged Scores	2.12 ± 0.14	1.74 ± 0.27	1.60 ± 0.89	0.00 ± 0.00	20.00 ± 44.72	0.00 ± 0.00	80.00 ± 44.72
SuperDiff AND	4.16 ± 0.05	2.36 ± 0.01	–	0.47 ± 0.14	21.25 ± 2.21	2.30 ± 0.36	75.98 ± 2.17
SuperDiff OR	2.41 ± 0.02	2.05 ± 0.01	–	7.66 ± 0.51	28.09 ± 1.23	17.52 ± 0.56	46.74 ± 0.46

B. Proportions of classifier probabilities

Method	World prob. (%)	Sports prob. (%)	Business prob. (%)	Sci/Tech prob. (%)
Expert 1 (World/Sports), GLASS	14.53 ± 0.82	35.38 ± 1.93	2.19 ± 0.27	47.90 ± 1.49
Expert 1 (World/Sports), PF ODE	11.74 ± 0.70	32.93 ± 1.63	2.16 ± 0.32	53.16 ± 1.13
Expert 2 (Business/Sports), GLASS	0.56 ± 0.19	33.00 ± 1.66	30.64 ± 2.30	35.80 ± 2.06
Expert 2 (Business/Sports), PF ODE	0.58 ± 0.15	32.19 ± 0.73	27.47 ± 0.67	39.76 ± 1.19
SMC: Kernel Mixture	0.05 ± 0.11	40.40 ± 54.23	4.34 ± 8.22	55.22 ± 50.71
SMC: Averaged Scores	0.68 ± 1.49	14.81 ± 32.93	0.21 ± 0.22	84.30 ± 32.78
SuperDiff AND	0.51 ± 0.13	21.53 ± 1.91	2.91 ± 0.43	75.05 ± 1.74
SuperDiff OR	7.72 ± 0.53	28.23 ± 0.95	17.33 ± 0.39	46.71 ± 0.29

Table 6: Composition results of CoBit-S trained on OWT in the form of mean ± standard deviation over 5 runs with different random seeds. Panel A reports sample quality metrics as well as the proportion of categories as identified using the classifiers. Unique texts indicate the number of unique text samples obtained using SMC, and is not applicable for other samplers. Panel B reports proportions of classifier probabilities.

A. Quality, diversity, and proportions of classifier predictions

Method	GenPPL	Entropy (nats)	Unique texts	Positive (%)	Electronics (%)	Both (%)
Expert 1 (Positive), GLASS	70.99 ± 3.50	3.75 ± 0.01	–	73.73 ± 1.62	30.14 ± 1.81	20.80 ± 1.50
Expert 1 (Positive), PF ODE	82.01 ± 3.86	3.88 ± 0.02	–	67.56 ± 1.67	32.68 ± 0.84	20.43 ± 0.39
Expert 2 (Electronics), GLASS	100.34 ± 3.59	3.72 ± 0.01	–	39.80 ± 0.99	78.87 ± 1.69	30.51 ± 1.05
Expert 2 (Electronics), PF ODE	121.52 ± 3.54	3.85 ± 0.01	–	36.93 ± 2.49	79.67 ± 1.85	28.28 ± 1.67
SMC: Kernel Mixture	344.61 ± 258.64	3.33 ± 0.48	1.20 ± 0.45	79.39 ± 44.40	20.00 ± 44.72	19.39 ± 43.37
SMC: Averaged Scores	77.69 ± 30.23	3.40 ± 0.45	1.00 ± 0.00	40.00 ± 54.77	60.00 ± 54.77	40.00 ± 54.77
SuperDiff AND	246.12 ± 4.60	4.01 ± 0.02	–	43.63 ± 1.00	65.96 ± 1.69	27.85 ± 1.53
SuperDiff OR	88.43 ± 3.61	3.84 ± 0.01	–	55.74 ± 1.27	50.94 ± 1.27	23.34 ± 1.60

B. Proportions of classifier probabilities

Method	Positive prob. (%)	Electronics prob. (%)	Empty texts
Expert 1 (Positive), GLASS	66.22 ± 0.97	26.42 ± 0.51	0.00 ± 0.00
Expert 1 (Positive), PF ODE	59.82 ± 1.02	27.12 ± 0.34	0.20 ± 0.45
Expert 2 (Electronics), GLASS	40.42 ± 0.46	46.60 ± 0.60	0.20 ± 0.45
Expert 2 (Electronics), PF ODE	37.78 ± 1.51	46.79 ± 0.58	0.20 ± 0.45
SMC: Kernel Mixture	63.03 ± 32.93	29.91 ± 11.63	0.00 ± 0.00
SMC: Averaged Scores	41.35 ± 36.15	41.28 ± 23.09	0.00 ± 0.00
SuperDiff AND	42.97 ± 0.74	40.82 ± 0.56	0.00 ± 0.00
SuperDiff OR	51.13 ± 1.11	34.84 ± 0.40	0.40 ± 0.55

Table 7: Composition results of CoBit-M trained on OWT in the form of mean $\pm$ standard deviation over 5 runs with different random seeds. Panel A reports sample quality metrics as well as the proportion of categories as identified using the classifiers. Unique texts indicate the number of unique text samples obtained using SMC, and is not applicable for other samplers. Panel B reports proportions of classifier probabilities.

A. Quality, diversity, and proportions of classifier predictions

Method	GenPPL	Entropy (nats)	Unique texts	Positive (%)	Electronics (%)	Both (%)
Expert 1 (Positive), GLASS	69.72 ± 0.56	3.77 ± 0.01	–	74.77 ± 2.52	23.16 ± 2.06	15.64 ± 1.35
Expert 1 (Positive), PF ODE	72.70 ± 1.59	3.88 ± 0.01	–	71.62 ± 1.04	23.79 ± 1.49	16.09 ± 0.83
Expert 2 (Electronics), GLASS	89.30 ± 2.32	3.74 ± 0.02	–	41.56 ± 1.55	79.20 ± 1.24	32.75 ± 1.94
Expert 2 (Electronics), PF ODE	89.83 ± 2.48	3.84 ± 0.02	–	40.53 ± 0.83	78.87 ± 1.13	30.78 ± 0.79
SMC: Kernel Mixture	238.56 ± 159.53	3.30 ± 0.65	1.40 ± 0.89	40.00 ± 54.77	79.92 ± 44.68	39.92 ± 54.67
SMC: Averaged Scores	148.09 ± 132.27	3.90 ± 0.46	1.40 ± 0.55	79.59 ± 44.50	100.00 ± 0.00	79.59 ± 44.50
SuperDiff AND	200.17 ± 9.48	4.18 ± 0.03	–	43.14 ± 1.95	61.66 ± 1.30	26.27 ± 1.05
SuperDiff OR	80.36 ± 4.75	3.86 ± 0.01	–	57.48 ± 1.36	48.79 ± 0.92	22.05 ± 1.08

B. Proportions of classifier probabilities

Method	Positive prob. (%)	Electronics prob. (%)	Empty texts
Expert 1 (Positive), GLASS	66.51 ± 1.29	22.41 ± 0.68	0.20 ± 0.45
Expert 1 (Positive), PF ODE	62.80 ± 0.66	22.52 ± 0.63	0.80 ± 0.84
Expert 2 (Electronics), GLASS	41.18 ± 1.09	47.07 ± 0.38	0.20 ± 0.45
Expert 2 (Electronics), PF ODE	40.48 ± 0.44	47.09 ± 0.48	0.80 ± 0.84
SMC: Kernel Mixture	46.00 ± 46.64	41.51 ± 16.86	0.00 ± 0.00
SMC: Averaged Scores	60.05 ± 29.77	47.42 ± 4.20	0.00 ± 0.00
SuperDiff AND	42.02 ± 1.28	39.46 ± 0.08	0.40 ± 0.55
SuperDiff OR	53.21 ± 1.10	33.43 ± 0.52	0.40 ± 0.55