Unconditional Sudoku Generation and Guided Solving with Continuous-Time Generative Models

Mariia Drozdova

University of Geneva
mariia.drozdova@unige.ch
*

January 5, 2026

Abstract

We study flow matching and score-based generative models in a highly constrained discrete setting, using Sudoku as a testbed. Starting from a continuous Gaussian probability path, we train neural vector fields and score models and sample from the resulting ordinary and stochastic differential equations. We first evaluate these models in a fully unconditional setting, measuring how often valid Sudoku grids are generated without any constraints. We find that stochastic sampling substantially outperforms deterministic flows, and that score-based SDEs with a path-dependent diffusion coefficient $\beta(t)$ achieve the highest success rates. We then show that the same models can be adapted to guided sampling, enabling stochastic Sudoku solving via repeated conditional generation and early stopping. While inefficient compared to classical solvers, this approach demonstrates that continuous-time generative models can assign non-zero probability mass to globally constrained combinatorial structures and can be repurposed for constraint satisfaction.

1 Introduction

Recent advances in generative modeling have demonstrated that complex data distributions can be learned by modeling the evolution of samples along continuous probability paths, starting from a simple distribution such as an isotropic Gaussian and gradually transforming it into the data distribution. In this work, we closely follow the formulation of flow matching and score matching models introduced in [1].

*Code available at https://github.com/MariiaDrozdova/sudoku_generation

We would like to study to what extent they can represent and sample from highly structured, discrete distributions. We investigate this question using Sudoku as a testbed, treating valid Sudoku grids as an extremely sparse subset of a continuous relaxation space.

1.1 Flow matching

We adopt a Gaussian probability path that interpolates between the data distribution and a standard Gaussian. For a data point $z \in \mathbb{R}^d$, we consider a conditional probability path of the form

$$p_t(x \mid z) = \mathcal{N}(x; \alpha_t z, \beta_t^2 I),$$

where α_t and β_t are continuous functions satisfying the boundary conditions $\alpha_0 = 0$, $\beta_0 = 1$, $\alpha_1 = 1$, and $\beta_1 = 0$. Sampling from this path can be written explicitly as

$$x_t = \alpha_t z + \beta_t \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I), \tag{1}$$

where x_1 corresponds to data.

A central result in flow-based generative modeling is that, given a probability path p_t, one can construct a corresponding vector field whose induced ordinary differential equation transports samples along this path. In particular, for the conditional Gaussian probability path defined above, there exists a conditional target velocity field $u_t^{\text{target}}(x \mid z)$ such that solutions of

$$\frac{dx_t}{dt} = u_t^{\text{target}}(x_t \mid z)$$

satisfy $x_t \sim p_t(\cdot \mid z)$ when initialized from $x_0 \sim \mathcal{N}(0, I)$.

For Gaussian probability paths, this conditional velocity field admits a closed-form expression,

$$u_t^{\text{target}}(x \mid z) = \left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t} \alpha_t \right) z + \frac{\dot{\beta}_t}{\beta_t} x,$$

where dots denote derivatives with respect to time. While the marginal velocity field obtained by integrating over the data distribution is generally intractable, an important result from flow matching theory shows that minimizing a regression loss against the conditional velocity field differs from minimizing the marginal loss only by a constant. As a consequence, learning to approximate the conditional velocity field is sufficient to recover the correct marginal dynamics.

In practice, we parameterize a neural network $u_\theta(x, t)$ and train it to approximate the conditional target velocity field under the Gaussian probability path. Once trained, generation proceeds by simulating the flow defined by the ordinary differential equation

$$\frac{dx_t}{dt} = u_\theta(x_t, t),$$

starting from Gaussian noise at $t = 0$ and integrating forward to $t = 1$. This constitutes the flow matching component of our approach and serves as the foundation for our experiments on unconditional Sudoku generation.

1.2 Score Matching and Diffusion Models

In addition to flow matching, the same probability path can be modeled using stochastic differential equations. Rather than describing the evolution of samples via a deterministic ordinary differential equation, diffusion-based models introduce stochasticity by adding a Brownian motion term. Concretely, given a probability path p_t, one considers stochastic dynamics of the form

$$dX_t = \left[u_t^{\text{target}}(X_t) + \frac{\sigma_t^2}{2} \nabla \log p_t(X_t) \right] dt + \sigma_t \, dW_t,$$

where σ_t is a time-dependent diffusion coefficient and W_t denotes a standard Brownian motion. From theory, it follows that this stochastic differential equation has marginal distributions $X_t \sim p_t$ for all $t \in [0, 1]$, provided the drift term is chosen as above.

The term $\nabla \log p_t(x)$ is known as the *score function*. While the marginal score function $\nabla \log p_t(x)$ is generally intractable, it can be expressed as an expectation over conditional score functions. As in the flow matching case, minimizing a loss against the conditional score function is sufficient, as the corresponding marginal objective differs only by an additive constant.

For the Gaussian probability path

$$p_t(x \mid z) = \mathcal{N}(x; \alpha_t z, \beta_t^2 I),$$

the conditional score function admits a closed-form expression,

$$\nabla \log p_t(x \mid z) = -\frac{x - \alpha_t z}{\beta_t^2}.$$

This expression can be used directly as a training target for a neural network approximating the score function.

For Gaussian paths, the velocity field and the score function are analytically related. As a result, one may train either component and derive the other explicitly. In our experiments, we consider both approaches and study their implications for sampling and stability.

$$s_\theta(x, t) = \frac{\alpha(t) \, u_\theta(x, t) - \dot{\alpha}(t) \, x}{\beta(t)^2 \, \dot{\alpha}(t) - \alpha(t) \, \dot{\beta}(t) \, \beta(t)}.$$

$$u_\theta(x, t) = \frac{\dot{\alpha}(t)}{\alpha(t)} \, x + \frac{\beta(t)^2 \dot{\alpha}(t) - \alpha(t)\beta(t)\dot{\beta}(t)}{\alpha(t)} \, s_\theta(x, t).$$

Once the score (and, if required, the velocity field) has been learned, sampling proceeds by numerically simulating the corresponding stochastic differential equation using Euler-Maruyama.

1.2.1 Connection to Denoising Diffusion Models

Langevin dynamics (static probability path). The [1] introduce an "SDE extension" of a probability path p_t via the Fokker–Planck equation. In particular, for an Itô SDE

$$dX_t = b_t(X_t)\,dt + \sigma_t\,dW_t,$$

the associated density p_t satisfies the Fokker–Planck equation

$$\partial_t p_t(x) = -\nabla \cdot \big(p_t(x)\, b_t(x)\big) + \frac{\sigma_t^2}{2}\Delta p_t(x).$$

The notes then show that for a (possibly time-dependent) path p_t, choosing the drift as

$$b_t(x) = u_t^{\text{target}}(x) + \frac{\sigma_t^2}{2}\nabla \log p_t(x)$$

yields an SDE whose marginals follow p_t. As a special case (Remark 16 in [1]), if the path is *static*, i.e. $p_t(x) = p(x)$ for all t (equivalently $\partial_t p_t(x) = 0$), one may set $u_t^{\text{target}} \equiv 0$ and obtain

$$dX_t = \frac{\sigma_t^2}{2}\nabla \log p(X_t)\,dt + \sigma_t\,dW_t,$$

which the notes identify as *Langevin dynamics*. In this static setting, p is a stationary distribution of the dynamics: if $X_0 \sim p$, then $X_t \sim p$ for all $t \geq 0$, and under suitable regularity conditions the dynamics converge to p from other initializations. Importantly, in the static Langevin setting σ_t can be chosen freely (it affects mixing speed rather than the stationary distribution).

Why static Langevin is not sufficient for our time-varying probability path. In our experiments we work with a time-varying Gaussian probability path p_t (Section 1), where p_t changes with t (e.g. through α_t and β_t). The static Langevin special case does not apply to such paths. In fact, if one attempts to remove the transport term while keeping a time-dependent target, i.e. sets

$$b_t(x) = \frac{\sigma_t^2}{2}\nabla \log p_t(x) \qquad \text{with} \qquad \sigma_t = \sigma(t),$$

then the Fokker–Planck equation forces the path to be static: using the identity $p_t(x)\nabla \log p_t(x) = \nabla p_t(x)$, one obtains

$$-\nabla \cdot \Big(p_t \frac{\sigma_t^2}{2}\nabla \log p_t\Big) + \frac{\sigma_t^2}{2}\Delta p_t = -\frac{\sigma_t^2}{2}\Delta p_t + \frac{\sigma_t^2}{2}\Delta p_t = 0,$$

hence $\partial_t p_t(x) = 0$. Therefore, *no choice of a purely time-dependent diffusion coefficient σ_t can compensate for the absence of transport when the desired marginals p_t vary with t.* This clarifies why the Langevin dynamics from [1] should be interpreted as a stationary sampler for a fixed distribution rather than as a finite-time noise-to-data bridge.

1.3 Diffusion discretization from a continuous Gaussian path

We train a score model

$$s_\theta(x,t) \approx \nabla_x \log p_t(x), \tag{2}$$

along the continuous path (1). For Gaussian perturbations, the conditional score is

$$\nabla_x \log p_t(x \mid x_0) = \frac{\alpha(t)x_0 - x}{\beta(t)^2}. \tag{3}$$

Rearranging (1) gives $x - \alpha(t)x_0 = \beta(t)\varepsilon$, hence

$$\nabla_x \log p_t(x \mid x_0) = -\frac{\varepsilon}{\beta(t)}.$$

This motivates converting a learned score into a noise predictor:

$$\varepsilon_\theta(x,t) = -\beta(t)\, s_\theta(x,t). \tag{4}$$

Even if training was performed in continuous time, we can apply a discrete sampler by evaluating s_θ at discrete t_k.

To obtain a practical sampler, we discretize the continuous time interval $t \in [0,1]$ into a grid $\{t_k\}_{k=0}^T$ with $t_0 = 0$ and $t_T = 1$, and define

$$\bar{\alpha}_k \;=\; \alpha(t_k)^2, \qquad 1 - \bar{\alpha}_k \;=\; \beta(t_k)^2. \tag{5}$$

At each discrete time, the marginal distribution satisfies

$$q(x_k \mid x_0) = \mathcal{N}\big(\sqrt{\bar{\alpha}_k}\, x_0,\ (1-\bar{\alpha}_k)I\big),$$

which coincides with the DDPM forward marginals. The learned continuous-time score $s_\theta(x,t)$ is therefore evaluated at $t = t_k$ and converted into a noise estimate via (4),

$$\varepsilon_\theta(x_k, t_k) = -\beta(t_k)\, s_\theta(x_k, t_k),$$

allowing us to apply standard DDPM reverse-time updates using the discrete coefficients $\{\bar{\alpha}_k\}$.

2 Methods

2.1 Dataset

We train on the same dataset as [2], consisting of 1M valid completed Sudoku grids. Each grid is represented as a 9×9 array with entries in $\{1, \ldots, 9\}$, and is flattened to a length-81 sequence of cells. For our continuous generative models, each cell is encoded as a 9-dimensional vector (logits), yielding a tensor of shape $(81, 9)$ per grid.

2.2 Probability path

Throughout this work we mostly use a simple linear Gaussian probability path, defined by

$$x_t = \alpha_t z + \beta_t \epsilon, \qquad \alpha_t = t, \quad \beta_t = 1 - t,$$

where z denotes a clean Sudoku grid encoded in $\mathbb{R}^{81\times 9}$, $\epsilon \sim \mathcal{N}(0, I)$, and $t \in [0, 1]$. This choice satisfies the boundary conditions $\alpha_0 = 0$, $\beta_0 = 1$ and $\alpha_1 = 1$, $\beta_1 = 0$, and corresponds to a linear interpolation between pure noise and the data distribution.

This linear path does not correspond to a diffusion forward process in the sense of denoising diffusion probabilistic models (DDPMs), since it does not satisfy $\alpha(t)^2 + \beta(t)^2 = 1$. Thus, for diffusion we use slightly different parametrization:

$$x_t = \alpha_t z + \beta_t \epsilon, \qquad \alpha(t) = \sin\Big(\frac{\pi}{2} t\Big), \qquad \beta(t) = \cos\Big(\frac{\pi}{2} t\Big),$$

More details are provided in Appendix B.

2.3 Neural Parameterization

To parameterize the time-dependent vector field (or score-like update) required by flow/score matching, we use a lightweight Transformer of 3M parameters operating on the 81 Sudoku cells as tokens. The network takes as input a tensor $z \in \mathbb{R}^{81\times 9}$ (one 9-dimensional state per cell) together with a scalar time variable $t \in [0, 1]$, and outputs an update direction $\Delta z \in \mathbb{R}^{81\times 9}$ of the same shape. The model first projects each per-cell 9-dimensional state into a hidden token embedding, adds structured positional embeddings reflecting Sudoku geometry (row, column, and 3×3 box identity), injects a time-conditioning vector, and then applies several Transformer blocks with multi-head self-attention and MLP layers. Finally, the hidden states are normalized and projected back to $\mathbb{R}^9$ per cell to produce the update direction. More details are provided in Appendix A.

2.4 Training Objectives

Velocity and score parameterization. For the linear Gaussian probability path $x_t = \alpha_t z + \beta_t \epsilon$ with $\alpha_t = t$ and $\beta_t = 1 - t$, the target velocity field admits a closed-form expression

$$u_t^{\text{target}}(x \mid z) = \Big(\dot\alpha_t - \frac{\dot\beta_t}{\beta_t}\alpha_t\Big) z \; + \; \frac{\dot\beta_t}{\beta_t}\, x,$$

where dots denote derivatives with respect to t. In the flow-matching setting, we train a neural network $u_\theta(x, t)$ to directly approximate this target velocity field by minimizing a mean squared error objective between $u_\theta(x_t, t)$ and $u_t^{\text{target}}(x_t \mid z)$, with (x_t, z) sampled from the Gaussian path.

Score-based parameterization. The corresponding conditional score along the same path is given analytically by

$$\nabla_x \log p_t(x \mid z) = -\frac{x - \alpha_t z}{\beta_t^2}.$$

Directly learning this expression was found to be numerically unstable in our setting, particularly near $t \to 1$ where $\beta_t \to 0$ and the score magnitude diverges. Instead, we parameterize a rescaled score model by training a neural network to predict the bounded quantity $\beta_t^2 \nabla_x \log p_t(x \mid z) = -(x - \alpha_t z)$. During sampling, this rescaling avoids explicit division by β_t^2 and significantly improves numerical stability of the resulting SDE dynamics.

Time sampling and stabilization. To emphasize late-time behavior, which is particularly important for satisfying global Sudoku constraints, training times are sampled non-uniformly. Specifically, we draw $u \sim \mathcal{U}(0, 1)$ and set

$$t = 1 - (1 - u)^2,$$

which biases samples toward larger values of t. In addition, we clamp the time variable away from one by enforcing $t \leq 1 - 10^{-4}$ to avoid numerical issues. This was enough to avoid training collapse for flow-based training, however for score-based training we have to additionally perform scaling as described above.

2.5 Sampling procedures

We simulate both deterministic and stochastic dynamics using explicit time discretizations. Let $\{t_k\}_{k=0}^{K}$ be an increasing time grid on $[0, 1]$ with step size $h_k = t_{k+1} - t_k$.

ODE sampling (Euler). For deterministic generation we integrate the learned flow field

$$\frac{dx_t}{dt} = u_\theta(x_t, t)$$

using the forward Euler scheme

$$x_{t_{k+1}} = x_{t_k} + h_k \, u_\theta(x_{t_k}, t_k).$$

This corresponds to deterministic flow matching and serves as a baseline in our experiments.

SDE sampling (Euler–Maruyama). Following the formulation from [1], we consider stochastic dynamics obtained by adding a Brownian correction to the flow:

$$dX_t = \Big(u_\theta(X_t, t) + \frac{1}{2} \sigma^2 \, s_\theta(X_t, t) \Big) \, dt + \sigma \, dW_t,$$

where $s_\theta(x,t) \approx \nabla_x \log p_t(x)$ is a learned score function and $\sigma > 0$ is a constant diffusion coefficient. This construction guarantees that the marginal distributions of X_t follow the probability path p_t, provided the score is exact.

We discretize this SDE using the Euler–Maruyama scheme,

$$x_{t_{k+1}} = x_{t_k} + h_k\Big(u_\theta(x_{t_k}, t_k) + \frac{1}{2}\,\sigma^2\, s_\theta(x_{t_k}, t_k)\Big) + \sigma\sqrt{h_k}\,\xi_k, \qquad \xi_k \sim \mathcal{N}(0, I).$$

Empirical $\beta(t)$-diffusion sampler. In addition to the theoretically grounded construction above, we introduce an *empirical SDE sampler* in which the stochastic term is scaled using the path variance $\beta(t)$:

$$x_{t_{k+1}} = x_{t_k} + h_k\Big(u_\theta(x_{t_k}, t_k) + \frac{1}{2}\,\sigma^2\, s_\theta(x_{t_k}, t_k)\Big) + \beta(t_k)\sqrt{h_k}\,\xi_k, \qquad \xi_k \sim \mathcal{N}(0, I).$$

Unlike the constant-σ case, this update does not correspond to a probability-path SDE derived from the Fokker–Planck equation, since the diffusion coefficient in the noise term does not match the drift correction. Nevertheless, we found that injecting noise proportional to $\beta(t)$ dramatically improves numerical stability and success rates, especially for highly constrained problems such as Sudoku.

We do *not* claim that this sampler exactly preserves the target marginals. Rather, it should be viewed as a guided stochastic search process inspired by the probability-path framework. All experimental results explicitly distinguish between constant-σ and $\beta(t)$-based samplers.

2.6 Sudoku solving

Sudoku solving corresponds to sampling *guided* completions given a set of fixed digits. Let $z_{\text{given}} \in \mathbb{R}^{81\times 9}$ denote the sudoku input (partially filled sudoku where zero correspond to unknown), and let $m \in \{0,1\}^{81\times 1}$ be the corresponding mask indicating constrained cells:

$$m_i = 1 \text{ if cell } i \text{ is given}, \qquad m_i = 0 \text{ otherwise.}$$

We write $\odot$ for elementwise multiplication with broadcasting over the digit dimension.

Hard clamping (inpainting). The simplest constraint mechanism clamps givens after each stochastic step:

$$\Pi_{\text{hard}}(x) \;=\; m \odot z_{\text{given}} + (1 - m) \odot x.$$

Two-stage soft $\to$ hard constraints (path-consistent injection). Hard clamping at early (high-noise) times can be too restrictive because it forces clean digits into a highly noisy state. To address this, we use a two-stage rule controlled by a threshold $\tau \in [0,1]$:

- **Soft (path-consistent) injection** for $t \le \tau$: we replace givens by a noisy version consistent with the forward path,

$$x_t^{\text{given}} = \alpha(t)\, z_{\text{given}} + \beta(t)\, \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I),$$

and set

$$\Pi_{\text{soft}}(x, t) \;=\; m \odot x_t^{\text{given}} + (1 - m) \odot x.$$

- **Hard clamping** for $t > \tau$: we enforce exact givens via Π_{hard}.

2.7 Evaluation metrics

Validity rate. For unconditional generation, we report the fraction of generated samples that correspond to fully valid Sudoku grids. A grid is considered valid only if it satisfies all Sudoku constraints exactly (rows, columns, and 3×3 subgrids). Partial or nearly-valid grids are not counted.

Entropy. To measure model confidence during sampling, we compute the mean per-cell entropy of the relaxed output distribution,

$$H = \frac{1}{81} \sum_{i=1}^{81} \Big(- \sum_{k=1}^{9} p_{i,k} \log p_{i,k} \Big),$$

where $p_{i,k}$ denotes the predicted probability of digit k in cell i. Entropy provides a local measure of uncertainty, with low entropy indicating near one-hot predictions. We also visualize entropy trajectories over time to study convergence behavior.

Constraint violations. After discretizing the relaxed output via a per-cell argmax, we count the number of violations of Sudoku constraints. Specifically, we measure the number of duplicate digits in each row, column, and 3×3 box. This metric captures the degree of global constraint satisfaction even when full validity is not achieved.

2.8 Implementation details.

The flow and score models have exactly same architecture and 3.3M parameters. Both flow and score models were trained separately for 300,000 iterations using a batch size of 2000 on an NVIDIA RTX 3090 GPU. All other architectural components and optimization settings were kept identical across models.

3 Results

3.1 Unconditional Sudoku Generation

Evaluation protocol. For unconditional generation we fix a batch size of 500 samples. For each model and each noise level σ, we run the sampler independently

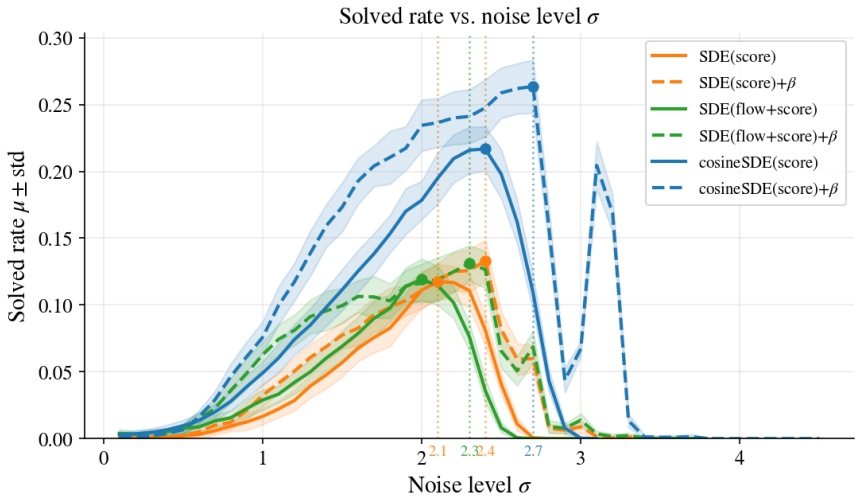

Figure 1: Comparison of the solved rate metric as a function of the noise level σ for SDE with only a trained score model(flow derived analytically from it), and SDE with both score and flow trained; both either with diffusion coefficient σ or $\beta(t)$. Markers indicate the maximum value for each curve, with the corresponding σ annotated.

50 times. At the end of each run we evaluate all 500 samples and count the fraction of *fully valid* Sudoku grids (satisfying all row, column, and 3×3 box constraints). We report the mean success rate across the 50 runs together with its standard deviation.

Compared sampling strategies. We train flow models and score models independently and evaluate all sampling procedures that can be constructed from them.

From a trained *flow* model we consider deterministic ODE sampling and stochastic SDE sampling, where the score term is reconstructed analytically from the learned flow, using either a constant diffusion coefficient σ or the path-dependent choice $\beta(t)$ in the simulator. From a trained *score* model we analogously consider the induced ODE, standard SDE sampling, and SDE sampling with $\beta(t)$. Finally, we evaluate SDEs constructed from *both* a learned flow and a learned score. These configurations isolate the effects of deterministic vs. stochastic sampling, flow-based vs. score-based parameterizations, and the choice of diffusion coefficient in the simulator. For the cosine schedule we restrict attention to score-based models, as this setting is primarily used to study diffusion-style discretizations (DDPM/DDIM).

Dependence on noise level. Figure 1 shows that SDE-based samplers exhibit a pronounced dependence on the diffusion scale σ, with a clear optimal region for

each method. Empirically, too little noise tends to produce premature collapse into invalid configurations, while excessive noise prevents convergence to discrete Sudoku solutions. Across methods, replacing a constant diffusion coefficient with the path-dependent choice $\beta(t)$ consistently shifts the curves to the left and increases the peak success rate. We also observe an immediate drop after the top performance, with the recovery afterwards. SDEs derived purely from a flow model perform poorly across all noise levels (Table 1); we omit them from Figure 1.

Table 1: Linear schedule for probability path coefficients $\alpha(t)$ and $\beta(t)$

Metric	SDE+Score+β	SDE+Score	SDE+Score+Flow+β	SDE+Score+Flow	SDE+Flow+β	SDE+Flow
Rate (50 runs)	0.1332	0.1173	0.1312	0.1191	0.0041	0.0048
Std. of rate	0.0155	0.0132	0.0134	0.0155	0.00248	0.00294
Pearson corr	-0.8838	-0.2978	-0.9077	-0.2921	-0.2339	-0.2907
p-value	$\approx 10^{-166}$	$\approx 10^{-11}$	$\approx 10^{-190}$	$\approx 10^{-10}$	$\approx 10^{-7}$	$\approx 10^{-10}$
Steps	200	200	200	200	200	200
Optimal σ	2.4	2.1	2.3	2.0	0.4	0.3

Linear probability path. Table 1 summarizes results under the linear schedule. SDEs derived from learned score models achieve success rates around 12%–13%, substantially outperforming both deterministic ODE sampling and SDEs derived purely from learned flow models (rightmost columns). Combining a learned score with a learned flow does not yield further improvements in this setting: the joint flow+score SDE performs comparably to the score-only SDE at their respective optimal noise levels. Across all SDE variants, using the path-dependent diffusion strength $\beta(t)$ consistently improves both the mean success rate and stability, yielding the strongest linear-path performance for `SDE+Score+`β and `SDE+Score+Flow+`β.

We additionally report the Pearson correlation between entropy and constraint violations as a diagnostic measure. Entropy is averaged over a late-time window spanning the last 30 to 15 integration steps, and constraint violations are averaged over the corresponding outputs. A stronger negative correlation indicates that lower entropy is predictive of fewer constraint violations, suggesting that entropy can serve as a proxy for convergence quality and for the model's uncertainty about constraint satisfaction.

Entropy dynamics. Figure 2 visualizes the evolution of mean per-cell entropy for linear-schedule SDEs at their respective optimal noise levels. Across SDE variants, a recurring pattern is observed in which entropy initially increases, then decreases as structure emerges, followed by a small late-stage increase

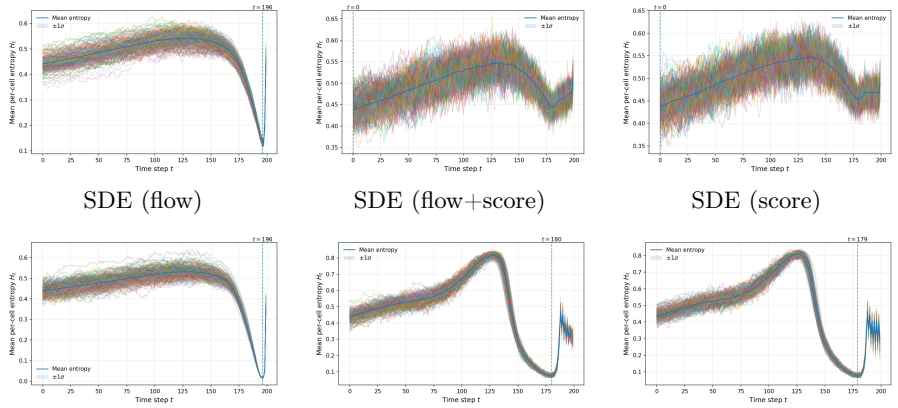

Figure 2: Entropy under linear schedule. For each plot we chose σ which had the best success rate. Top row shows SDE variants with canonical Brownian noise; bottom row shows the corresponding variants with β_t instead of σ in the simulator. Columns compare having only score model trained or only flow trained, or combining both flow and score models trained separately.

before convergence. Variants using the path-dependent diffusion coefficient $\beta(t)$ typically reach a lower minimum entropy than their constant-noise counterparts. This lower minimum can be partially explained by the relatively large noise levels selected for SDE samplers (cf. Figure 1), which promote broader exploration. For constant diffusion coefficient it means large noise during all the inference. For β however the noise diminishes when t approaches 1. After reaching this minimum, mild entropy oscillations are observed for $\beta(t)$-samplers. These oscillations may have a numerical origin or reflect the fact that this choice of diffusion no longer exactly follows the prescribed probability path. Nevertheless, despite this deviation, $\beta(t)$-based samplers consistently achieve superior final success rates, indicating that strict adherence to the probability path is not required for effective convergence in this setting.

Table 2: Cosine schedule for probability path coefficients $\alpha(t)$ and $\beta(t)$ and diffusion discretizations.

Metric	SDE+β	SDE	DDIM	DDPM	ODE
Rate (50 runs)	0.26356	0.21676	0.00560	0.03120	0.00150
Std. of rate	0.0199169	0.0167040	0.0019596	0.0041183	0.0008660
Pearson corr	-0.6507	-0.6129	-0.2260	-0.1888	-0.1392
p-value	$\approx 10^{-61}$	$\approx 10^{-53}$	$\approx 10^{-7}$	$\approx 10^{-5}$	$\approx 10^{-3}$
Steps	200	200	200	200	200
Optimal σ	2.7	2.4	–	–	–

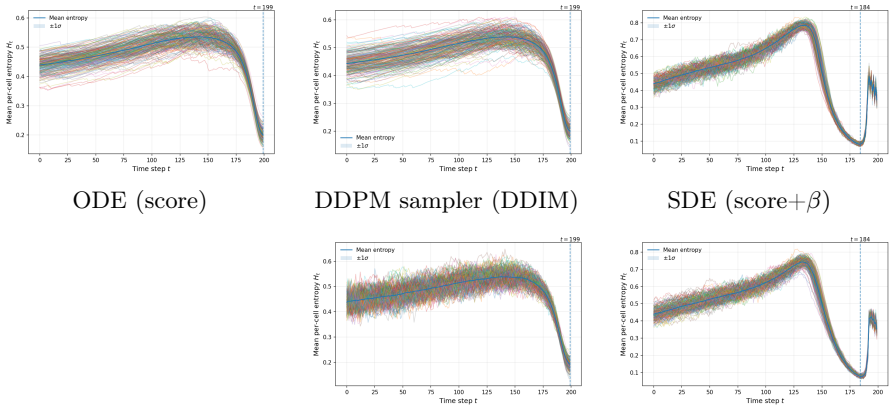

Figure 3: Entropy under cosine schedule for the Gaussian probability path. Right column compares SDE score with diffusion coefficient equal to either σ or $\beta(t)$. Middle column compares DDPM samplers (DDIM vs DDPM). ODE results (with a velocity field derived from the trained score) are shown adjacent to DDIM for reference.

Cosine schedule for probability path coefficients and diffusion discretizations. Results under the cosine schedule are reported in Table 2 and Figure 3. In this setting we train a score model in continuous time along the cosine probability path and then evaluate both continuous-time samplers (ODE/SDE) and discrete-time diffusion samplers (DDPM/DDIM) obtained by discretization. Continuous-time SDE sampling substantially outperforms DDPM and DDIM in this highly constrained domain. Among all evaluated unconditional methods, the cosine schedule combined with an SDE using $\beta(t)$ achieves the highest success rate, exceeding 25%.

Discrete DDPM and DDIM samplers perform markedly worse than continuous-time SDEs in our implementation. A plausible explanation is that discretization effects and sensitivity to early commitment errors are amplified in highly constrained combinatorial spaces, where a small number of wrong early decisions can lead to unavoidable constraint violations later. We emphasize that in our setup DDPM/DDIM are applied only at sampling time by discretizing a continuously trained score model, which may further accentuate this mismatch.

Across all experiments, stochastic SDE sampling consistently outperforms deterministic ODE sampling. Score-based SDEs are substantially more effective than flow-derived SDEs, while combining flow and score does not improve performance under the linear schedule. Cosine schedules outperform linear schedules for unconditional Sudoku generation, and the strongest overall method is the cosine SDE with diffusion coefficient $\beta(t)$. Finally, discrete DDPM/DDIM discretizations underperform continuous-time SDE samplers in this highly con-

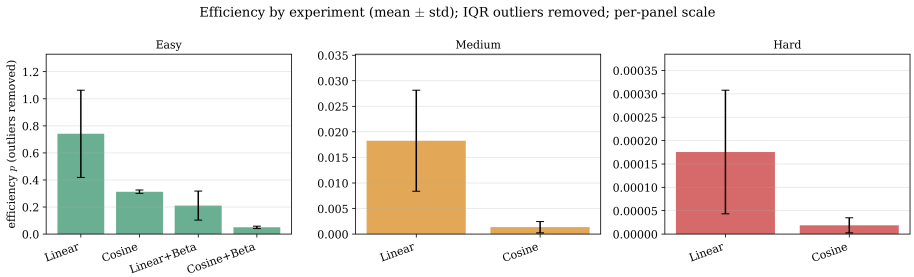

Figure 4: Efficiency comparison across noise schedules and diffusion parameterizations for Easy, Medium, and Hard Sudoku instances. Bars show mean efficiency over runs with one standard deviation. Linear schedules consistently outperform cosine schedules, motivating their use in subsequent experiments.

strained setting, suggesting that the continuous-time dynamics are more robust to the early commitment errors that arise when sampling valid Sudoku grids from noise.

3.2 Guided Sudoku Solving

In addition to unconditional generation, we evaluate whether the learned continuous-time generative models can be used to solve partially observed Sudoku puzzles by conditioning on a subset of fixed digits. In this setting, the objective is to generate any configuration consistent with a given set of hard constraints.

Guided sampling setup. Given a partially filled Sudoku grid, we construct a binary mask indicating fixed (given) cells and free variables. Sampling is performed using a guided SDE, where constraints are enforced through masked updates during the simulator steps. We employ a two-stage constraint strategy controlled by a threshold $t_{\text{hard}} \in [0, 1]$, which determines when constraints transition from a noise-consistent (soft) enforcement to exact clamping.

Sampling protocol. For each Sudoku instance, we repeatedly sample batches of 512 trajectories. After each batch, all samples are discretized via per-cell argmax and checked for Sudoku validity. If at least one valid solution is found, sampling terminates early; otherwise, a new batch is generated. The threshold t_{hard} is sampled independently for each run from the set $\{0.0, 0.45, 0.5, 0.55\}$. This choice reflects empirical observations that different puzzles and models favor different transition points between soft and hard constraint enforcement.

Observed behavior. Across a range of difficult Sudoku instances, this procedure consistently assigns non-zero probability mass to valid completions. As a consequence, repeated sampling eventually produces a correct solution. The number of required batches varies substantially across puzzles: some instances

converge quickly, while others require many sampling rounds and correspondingly longer runtimes.

Model and schedule choice. While cosine-scheduled SDEs have been shown to perform strongly in unconditional generation, their effectiveness in the guided Sudoku-solving setting is less clear. We therefore conducted a focused empirical comparison using an SDE-based score model with two noise schedules (linear and cosine) and two diffusion parameterizations: a constant diffusion coefficient σ and a time-dependent coefficient $\beta(t)$.

To quantify efficiency, we define a normalized success probability per sampled instance:

$$p \;=\; \frac{N_{\text{valid}}}{(r+1)\,N_{\text{batch}}},$$

where r is the number of failed sampling attempts before the first successful run, N_{batch} is the batch size (number of candidate sudokus evaluated per attempt), and N_{valid} denotes the number of valid sudokus obtained *in the first successful run*. All previous runs are unsuccessful and therefore contribute zero valid solutions. This metric measures the fraction of valid solutions among all evaluated samples up to the first success, jointly accounting for stochastic retries and batch evaluation cost.

On easy puzzles, this preliminary evaluation already reveals a clear separation: linear probability paths substantially outperform cosine schedules across both diffusion choices. In particular, the linear schedule with $\beta(t)$-scaled diffusion achieves significantly higher efficiency than all other configurations (Figure 4). Cosine-based variants, especially without $\beta(t)$ scaling, require substantially more sampling and frequently fail to produce a valid solution within a reasonable budget.

Based on these observations, we restrict subsequent medium and hard puzzle experiments to the linear schedule with $\beta(t)$-scaled diffusion. Other configurations are excluded, as they often require excessive computation and do not reliably find a solution even after 1,000 repetitions with a batch size of 512. Notably, the linear schedule continues to outperform alternative choices consistently across increasing difficulty levels.

Hard Sudoku instances and runtime. We further evaluated the guided solver on hard Sudoku instances. For these puzzles, the solver consistently succeeds but requires multiple stochastic sampling rounds. With a batch size of 512, we observe an average of 8.9 batches before the first valid solution is obtained, corresponding to approximately 4,500 evaluated candidate grids per puzzle. While some instances converge more quickly, others require substantially more sampling, leading to high variance in runtime.

These results confirm that the guided SDE assigns non-zero probability mass to valid completions even in the hardest cases. However, they also highlight the computational cost of relying purely on stochastic sampling without explicit

search or symbolic constraint propagation. Reducing the number of required samples is an important direction for future work.

4 Discussion

This work explored continuous-time generative models based on flow matching and score matching as a testbed for highly structured discrete data, using Sudoku as a challenging benchmark. Our primary focus was unconditional generation, with guided Sudoku solving studied as an application that probes the learned geometry of the solution manifold.

Unconditional generation. Across all experiments, stochastic SDE-based samplers consistently outperformed deterministic ODE sampling. Score-based SDEs achieved the highest unconditional success rates, indicating that explicitly modeling the score provides a stronger inductive bias for navigating the constrained Sudoku manifold than learning a velocity field alone. Using a path-dependent diffusion coefficient $\beta(t)$ further improved stability and performance, even though this choice does not correspond to a classical diffusion process for the linear probability path.

Among all evaluated methods, the cosine schedule combined with score-based SDE sampling achieved the best unconditional performance, exceeding 25% valid Sudoku grids per batch. Discrete DDPM and DDIM samplers underperformed continuous-time SDEs in this setting, suggesting that discretization artifacts and early commitment errors are particularly detrimental for highly constrained combinatorial problems.

Guided Sudoku solving. For guided Sudoku solving, we observed a different empirical behavior. While cosine-based SDEs performed best for unconditional generation, they were less effective for solving difficult partially observed puzzles under our guided sampling scheme. In contrast, linear probability paths combined with score-based SDE sampling and $\beta(t)$-scaled diffusion consistently produced solutions more reliably and often faster.

It suggests that the smoother, more uniform noise decay of the linear path may be better suited for constraint satisfaction under hard conditioning, whereas the cosine schedule excels at unconditional generation. Understanding this tradeoff remains an interesting direction for future work.

Interpretation. The guided solver does not perform symbolic reasoning or explicit constraint propagation. Instead, it relies on repeated stochastic exploration of a learned continuous relaxation of the Sudoku solution space. Although individual samples have low probability of being valid, the model assigns non-zero probability mass to correct solutions, allowing valid grids to emerge through repeated sampling and early stopping.

From an algorithmic perspective, this approach is inefficient compared to classical Sudoku solvers. However, it demonstrates that continuous generative

models, trained purely on completed solutions, can be repurposed as stochastic constraint solvers without task-specific heuristics or search logic. This positions generative modeling as a complementary paradigm for structured reasoning tasks, rather than a replacement for classical algorithms.

Limitations and future work. This study is exploratory and has several limitations. We did not perform ablation studies on the guided solver, nor did we systematically tune the threshold schedule or sampling parameters. Training and evaluation were limited to a single dataset and model architecture. Future work could investigate principled constraint injection strategies, adaptive threshold selection, and extensions to other combinatorial problems. A deeper theoretical understanding of why certain diffusion choices outperform probability-path consistent dynamics is also an open question.

5 Conclusions

Overall, our results show that continuous-time generative models can assign meaningful probability mass to extremely sparse, highly constrained discrete structures. Even when individual samples are unlikely to be correct, repeated stochastic sampling enables both unconditional generation and guided problem solving. Sudoku serves as a compact but demanding benchmark, highlighting both the promise and current limitations of diffusion-based generative modeling for structured reasoning tasks.

References

[1] Peter Holderrieth and Ezra Erives. Introduction to flow matching and diffusion models. `https://diffusion.csail.mit.edu/`, 2025. MIT Course 6.S184: Generative AI with Stochastic Differential Equations.

[2] Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks, 2025.

A Model

Time embedding. We condition the network on the continuous time variable t using Fourier features. Concretely, we map t to a vector of sinusoidal components at exponentially spaced frequencies:

$$\phi(t) = [\sin(2\pi f_1 t), \ldots, \sin(2\pi f_K t),\ \cos(2\pi f_1 t), \ldots, \cos(2\pi f_K t)],$$

where $(f_k)_{k=1}^{K}$ spans a range from low to high frequencies. This embedding allows the model to represent complex time dependencies and is widely used in diffusion/score-based models to make the network sensitive to both coarse

and fine variations across the noise levels. The Fourier features are then passed through a small MLP to produce a time-conditioning vector in the Transformer hidden dimension, which is added to every token embedding.

Transformer backbone. The backbone consists of L standard Transformer blocks. Each block applies layer normalization, multi-head self-attention over the 81 cell tokens, and a position-wise MLP with GELU nonlinearity. Residual connections are used around both the attention sublayer and the MLP sublayer, with dropout applied to stabilize training. This architecture enables global interactions between cells, which is essential for Sudoku due to its non-local row/column/box constraints.

Sudoku-aware positional encoding. Unlike generic sequence tasks, Sudoku tokens have a known 2D structure and a third type of locality induced by the 3×3 subgrids. We therefore use three learned embedding tables for row index, column index, and box index, and add them to the token representation of each cell. For a cell at position (r, c) with box index $b(r, c)$, the positional vector is

$$e_{\text{pos}}(r, c) = e_{\text{row}}(r) + e_{\text{col}}(c) + e_{\text{box}}(b(r, c)).$$

This biases attention toward respecting Sudoku geometry and provides the model explicit access to the constraint structure.

B DDPM from continuous score models

From $\bar{\alpha}_k$ to (α_k, β_k). DDPM assumes a Markov forward chain of the form

$$q(x_k \mid x_{k-1}) = \mathcal{N}\big(\sqrt{\alpha_k}\, x_{k-1},\ \beta_k I\big), \qquad \beta_k = 1 - \alpha_k,$$

with independent Gaussian noise increments. Expanding the recursion yields

$$x_k = \sqrt{\alpha_k} x_{k-1} + \sqrt{\beta_k}\, \epsilon_k = \sqrt{\alpha_k \alpha_{k-1}} x_{k-2} + (\cdots) = \Big(\prod_{i=1}^{k} \sqrt{\alpha_i}\Big) x_0 + \text{Gaussian noise.}$$

Thus the coefficient multiplying x_0 in $q(x_k \mid x_0)$ equals $\prod_{i=1}^{k} \sqrt{\alpha_i}$, and comparing with (8) gives

$$\sqrt{\bar{\alpha}_k} = \prod_{i=1}^{k} \sqrt{\alpha_i} \quad \Longrightarrow \quad \boxed{\bar{\alpha}_k = \prod_{i=1}^{k} \alpha_i}.$$

Consequently the per-step coefficients must satisfy

$$\boxed{\alpha_k = \frac{\bar{\alpha}_k}{\bar{\alpha}_{k-1}},} \qquad \boxed{\beta_k = 1 - \alpha_k.} \tag{6}$$

This is exactly the construction implemented in code via $a_k = \bar{\alpha}_k / \bar{\alpha}_{k-1}$ and $b_k = 1 - a_k$.

Gaussian posterior and $\tilde{\beta}_k$ (derivation). DDPM sampling uses the exact posterior $q(x_{k-1} \mid x_k, x_0)$. Conditioned on x_0, the pair (x_{k-1}, x_k) is jointly Gaussian:

$$x_{k-1} \mid x_0 \sim \mathcal{N}\big(\sqrt{\bar{\alpha}_{k-1}}x_0,\ (1-\bar{\alpha}_{k-1})I\big), \qquad x_k \mid x_{k-1} \sim \mathcal{N}\big(\sqrt{\alpha_k}x_{k-1},\ \beta_k I\big).$$

For jointly Gaussian variables, the conditional covariance is given by the Schur complement,

$$\mathrm{Var}(X \mid Y) = \mathrm{Var}(X) - \mathrm{Cov}(X,Y)\,\mathrm{Var}(Y)^{-1}\,\mathrm{Cov}(Y,X).$$

Applying this identity with $X = x_{k-1}$ and $Y = x_k$ (conditioning throughout on x_0) gives

$$\mathrm{Var}(x_{k-1} \mid x_k, x_0) = \mathrm{Var}(x_{k-1} \mid x_0) - \mathrm{Cov}(x_{k-1}, x_k \mid x_0)\,\mathrm{Var}(x_k \mid x_0)^{-1}\,\mathrm{Cov}(x_k, x_{k-1} \mid x_0).$$

Compute the required covariances (all are scalar multiples of I):

$$\mathrm{Var}(x_{k-1} \mid x_0) = (1-\bar{\alpha}_{k-1})I, \qquad \mathrm{Var}(x_k \mid x_0) = (1-\bar{\alpha}_k)I,$$

and using $x_k = \sqrt{\alpha_k}x_{k-1} + \sqrt{\beta_k}\epsilon_k$,

$$\mathrm{Cov}(x_{k-1}, x_k \mid x_0) = \sqrt{\alpha_k}\,(1-\bar{\alpha}_{k-1})I.$$

Substituting yields the intermediate form

$$\begin{aligned}\mathrm{Var}(x_{k-1} \mid x_k, x_0) &= (1-\bar{\alpha}_{k-1})I - \frac{\alpha_k(1-\bar{\alpha}_{k-1})^2}{1-\bar{\alpha}_k}\,I \\ &= \frac{(1-\bar{\alpha}_{k-1})\big(1-\bar{\alpha}_k-\alpha_k(1-\bar{\alpha}_{k-1})\big)}{1-\bar{\alpha}_k}\,I.\end{aligned}$$

Finally, using $\bar{\alpha}_k = \alpha_k\bar{\alpha}_{k-1}$ and $\beta_k = 1-\alpha_k$,

$$1-\bar{\alpha}_k-\alpha_k(1-\bar{\alpha}_{k-1}) = 1-\alpha_k\bar{\alpha}_{k-1}-\alpha_k+\alpha_k\bar{\alpha}_{k-1} = 1-\alpha_k = \beta_k,$$

and therefore

$$\boxed{\tilde{\beta}_k = \frac{1-\bar{\alpha}_{k-1}}{1-\bar{\alpha}_k}\,\beta_k, \qquad \sigma_k = \sqrt{\tilde{\beta}_k}.} \tag{7}$$

In practice, one often avoids the singular endpoints $\bar{\alpha}_0 = 1$ and $\bar{\alpha}_T = 0$ by slightly truncating the time grid, which prevents division by zero in the expressions above.

Cosine schedulers for probability paths parameters. We introduced previously Gaussian probability path

$$x_t = \alpha_t z + \beta_t \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I).$$

A DDPM forward process is defined so that the marginal at (discrete) step k satisfies

$$q(x_k \mid x_0) \;=\; \mathcal{N}\big(\sqrt{\bar{\alpha}_k}\,x_0,\ (1-\bar{\alpha}_k)I\big), \tag{8}$$

for some schedule $\{\bar{\alpha}_k\}_{k=0}^{T}$ with $\bar{\alpha}_0 = 1$ (clean) and $\bar{\alpha}_T \approx 0$ (noise). Matching (1) with (8) at discrete times t_k gives the identification

$$\sqrt{\bar{\alpha}_k} = \alpha(t_k), \qquad \sqrt{1-\bar{\alpha}_k} = \beta(t_k), \tag{9}$$

hence a necessary condition for a continuous Gaussian path (1) to coincide with diffusion marginals at all t_k is

$$\boxed{\alpha(t)^2 + \beta(t)^2 = 1} \quad \text{for all } t.$$

The linear choice $\alpha(t) = t$, $\beta(t) = 1 - t$ violates this condition for $t \in (0, 1)$, and therefore does *not* correspond to any diffusion forward process (even though it defines a valid Gaussian interpolation). A diffusion-compatible alternative is obtained by parameterizing (α, β) on the unit circle, e.g.

$$\alpha(t) = \sin\Big(\frac{\pi}{2}t\Big), \qquad \beta(t) = \cos\Big(\frac{\pi}{2}t\Big),$$

which satisfies $\alpha(t)^2 + \beta(t)^2 = 1$ identically and induces $\bar{\alpha}_k = \alpha(t_k)^2$. In our implementation we apply time reversal $t \leftarrow 1 - t$ to align the loop direction with the simulator; this changes only the indexing convention, not the underlying derivations.