Wildfire Spread Scenarios: Increasing Sample Diversity of Segmentation Diffusion Models with Training-Free Methods

Anonymous Full Paper
Submission 56

Abstract

Wildfire spread is an inherently stochastic process. To capture this stochasticity, we train a generative diffusion model to predict the wildfire spread. Such models can predict multiple different outcomes per input. However, seeing all possible outcomes may require hundreds of samples, since some of them have a low generation probability. To make this more efficient, we examine methods that bias the sampling process: away from the correct generation probabilities and towards higher sample diversity. To train this model, we introduce a simulation-based wildfire spread dataset called MMFire. Furthermore, we use a modified version of Cityscapes and the medical dataset LIDC, to ensure that our methodological findings transfer across domains. The diversity-encouraging methods we explore are particle guidance, SPELL, and our own clustering-based approach. All methods beat naive sampling, with SPELL proving to be best, increasing the HM IoU* metric by 7.5% on MMFire and 16.1% on Cityscapes with little cost to image quality and runtime.

The code and the MMFire dataset will be made publicly available upon acceptance.

1 Introduction

Recent papers on daily wildfire spread prediction [1–3] fail to achieve a high predictive performance, even though we know which variables are relevant for the physical processes at play. We believe that this might be related to the high uncertainty that is inherent to wildfire spreading, especially at the typically very low spatial and temporal resolution in these studies. Given this uncertainty, models should likely consider *several* outcomes to capture the range of likely options. We use diffusion models to generate this range of outcomes directly.

Although various papers explore how to use diffusion models to generate segmentation masks [4–12], they tend to simply aggregate the generated masks. This ignores the core advantage of generative models: their ability to generate *distinct* outputs. Some research examines how to train diffusion models to generate such distinct segmentation masks [10, 11] with well-calibrated probabilities. Yu et al. [13] even generate wildfire spread predictions with a diffusion model, but only compare the averaged predictions

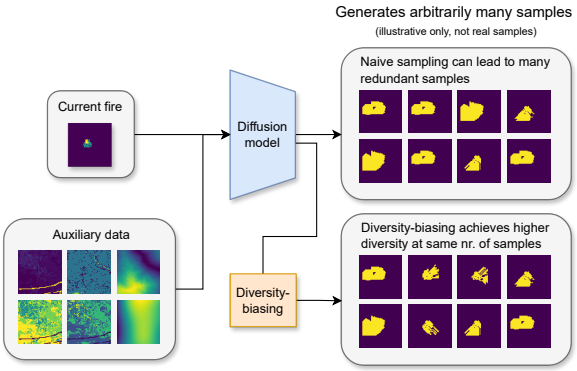

Figure 1. Diversity-biased sampling: We train a conditional diffusion model to generate different outputs for the same input data. If the goal is to find most, or all, different outputs for the current input, naive sampling can require a large number of samples, due to the redundancy in samples. To reduce this redundancy, we employ methods that bias the sampling towards higher diversity for the same number of samples.

to averaged simulated outcomes. We instead want to use the distinct predictions directly, to enable fire fighters to better plan for these different scenarios. To efficiently generate different predictions, instead of redundant ones, we bias the sampling process towards higher diversity.

Output diversity is relevant in various domains that have more than one possible target, e.g. different opinions of medical experts [11], or in temporal prediction: How will an active wildfire spread? Will a child chase a ball that rolls onto the street? In such applications, generating a diversity-biased set of scenarios can be much more useful than generating scenarios according to properly calibrated probabilities, or working with pixel-wise probabilities.

As our main motivation is wildfire spread, we first introduce MMFire as a benchmark dataset. For each starting condition, it contains multiple plausible outcomes of wildfire spread, based on uncertainty about the wind direction. In an application setting, it is unrealistic to not know the wind direction at all. However, this dataset is not meant to be perfectly realistic. It is meant to serve as a controlled environment for evaluating different diversity-encouraging methods. Trained on enough real-world data, or additional synthetic data, future models should be able to generate diverse wildfire spread predictions. Any knowledge gained from working on MMFire can

then be transferred to these models.

To draw conclusions that transfer across domains, we also use a binary variant of Cityscapes [14], and the medical dataset LIDC [15]. These datasets also contain *multiple* binary segmentation masks as targets for each input.

To cost-efficiently increase diversity, we focus on training-free methods: Particle guidance [16] and SPELL [17]. Both methods modify the sampling trajectory by repelling samples from each other, and thus increasing diversity. We also propose a simple clustering-based approach that starts with a large batch of samples and discards redundant ones very early on, to reduce computational cost.

Our key contributions are the following:

- We introduce MMFire, a simulated wildfire spread dataset with multiple valid future outcomes per input (subsection 4.1).
- We draw a connection between diversity statistics for segmentation masks and SPELL's critical parameter that eliminates the need for extensive hyperparameter search (subsection 5.4).
- We develop a clustering-based pruning approach. It beats naive sampling by 2.4% HM IoU* on MMFire (Table 2) and up to 15.5% on Cityscapes (Table 1), without modifying the original sampling trajectories.
- We demonstrate that particle guidance and SPELL both achieve superior quality-diversity trade-offs compared to naive sampling (Table 2, Table 3). For single batches, SPELL beats naive sampling by 7.5% HM IoU* on MMFire and 16.1% on Cityscapes.

2 Related work

Research on generating segmentation masks with diffusion models either uses the *Gaussian diffusion* framework or variants of *categorical diffusion*. When generating binary segmentation masks [4, 5, 12, 18], Gaussian diffusion can be used directly, followed by thresholding to binarize the real-valued outputs. To extend this from binary to multi-class segmentation masks, Analog Bits [7, 8] can be used with little change to the underlying mechanics.

In contrast to Gaussian diffusion, categorical diffusion [6, 9, 11] uses discrete state spaces, instead of real-valued ones. Empirically, Gaussian and categorical approaches perform similarly [9, 11].

In this work, we use Gaussian diffusion. This allows us to integrate diversity-related methods [16, 17], that have been developed for Gaussian diffusion, more easily. We focus on binary masks, assuming that the results can be transferred to the multi-class setting via Analog Bits.

Most studies on diffusion segmentation models focus on achieving a high segmentation performance, by aggregating multiple samples as a form of implicit ensembling [5], or improving segmentation and calibration scores of the mean-aggregated samples [4]. However, these improvements could conceptually also be achieved with discriminative methods. We instead want to focus on the unique ability of generative methods to generate multiple *different* predictions for the same input. We are only aware of two studies [10, 11] that investigate the performance of their model on a dataset with *multiple* correct annotations, also termed *ambiguous segmentation*.

Various methods have been proposed in the diffusion model literature to increase sample diversity, though they usually focus on the text-conditioned generation of natural images. CADS [19] adds a noise schedule to the conditioning. This is supposed to prevent samples from focusing on the most probable modes, and instead explore more of the latent space. In preliminary experiments, we found that CADS severely degrades the image quality and thus do not use it. This degradation is likely due to the need for the model to access conditioning information early in sampling, to establish the low-frequency information that represents the segmentation mask.

Instead of modifying the conditioning to increase diversity, most methods modify the sampling schedule. Particle guidance [16] computes a guidance term based on the pairwise distances between noise-free predictions of the current in-batch samples to repel them from each other. Motion modes [20] extends this to include several additional guidance terms, that encourage properties in the generated data that particle guidance might otherwise not preserve. We directly use particle guidance, since our domain does not lend itself as easily to additional guidance terms.

ProCreate [21] aims to generate samples that differ from existing samples. For a more accurate distance computation, the method 'looks ahead' by denoising for several steps. It then computes a guidance term similar to particle guidance. We also investigate the case of generating multiple batches of data with repellence from previously-sampled images. In contrast to ProCreate, we only use a single-step denoising for distance computations, since we find that initial predictions are rather close to the final samples for binary segmentation masks.

Contrary to guidance-based methods, SPELL [17] does not indiscriminately repel all close samples from each other. Instead, if two samples lie within a pre-defined L2-distance of each other (the *shield radius*), SPELL repels them just enough to ensure that the distance is maintained. We use SPELL as an alternative to particle guidance.

3 Method

We use the EDM diffusion framework [22] to generate segmentation masks, conditioned on an input image. At inference, we generate multiple masks by denoising multiple random noise samples with the trained diffusion model. Particle guidance[16] and SPELL [17] are used during the denoising process to increase the diversity among these generated masks. They work heuristically, by pushing the samples in a batch away from each other, thus increasing the diversity within a batch.

3.1 EDM diffusion framework

We follow the EDM framework [22] for our denoising diffusion models. The EDM model is based on the following ordinary differential equation (ODE):

$$\mathrm{d}\boldsymbol{x} = -t\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}; t)\mathrm{d}t, \tag{1}$$

where $\boldsymbol{x}$ is a noisy mask (also called *latent*) and t is the ODE time step. We also refer to t as the *noise level*, given that we use the default variance exploding formulation, where $\sigma(t) = t$. The ODE is solved via numerical integration with a 2nd order Heun solver [22]. This numerical integration starts from a pure noise mask with very high noise level $\sigma_{\max}$ and gradually removes all of the noise until a noise-free mask is reached.

We train a denoising neural network D_θ to remove noise by minimizing the objective:

$$\mathbb{E}_{\boldsymbol{y}\sim p_{\text{data}}}\mathbb{E}_{t\sim p_{\text{train}}}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},t^2\mathbf{I})}\|D(\boldsymbol{y}+\boldsymbol{\epsilon};t)-\boldsymbol{y}\|_2^2, \tag{2}$$

where a segmentation mask $\boldsymbol{y}$ is sampled from the data distribution p_{data} ; the current time step t is sampled from p_{train}; and a noise image $\boldsymbol{\epsilon}$ is sampled from an isotropic Gaussian with standard deviation t. The forward diffusion process then simply consists of adding the noise $\boldsymbol{\epsilon}$ to the ground truth segmentation mask $\boldsymbol{y}$. We refer to the noisy segmentation mask as $\boldsymbol{y}_t$. By optimizing this objective, the denoising model D_θ learns to predict the noise-free $\boldsymbol{y}$, given $\boldsymbol{y}_t$ and the current time step t.

After training D_θ, Equation 1 can be solved by approximating the score function:

$$\text{score}(x,t) = \nabla_{\boldsymbol{x}} \log p(\boldsymbol{x};t) = (D_\theta(\boldsymbol{x};t)-\boldsymbol{x})/t^2. \tag{3}$$

To use the EDM framework for segmentation, the generated sample needs to be conditioned on an input image that we want to segment. Therefore, we sample pairs ($\boldsymbol{y}$, $\boldsymbol{c}$) from the data distribution with segmentation mask $\boldsymbol{y}$ and input image, or *conditioning*, $\boldsymbol{c}$. We pass $\boldsymbol{c}$ to the denoising network D_θ as an additional input. We implement this by concatenating $\boldsymbol{c}$ to the current noisy segmentation mask $\boldsymbol{y}_t$ in the channel dimension.

3.2 Increasing sample diversity

When generating natural images, the reverse diffusion process first determines low-frequency features at high noise levels (e.g. where in the image we see a dog), and as the sample x_t moves towards lower noise levels, more high-frequency features are determined (e.g. the details of the face and then of the fur). However, in images that are binary segmentation masks, there are very few such high-frequency features, since all pixels take values of 0 or 1, without any fine-grained differences in between. This is highly relevant for any diversity-encouraging methods that modify the sampling process, since it means that the changes we care about are only possible near high noise levels.

Furthermore, in exploratory experiments, we found that the denoiser model's prediction $D_\theta(x_t, t_{\max})$ at the initial time step $t_{\max}$ is often relatively close to the final output already. This allows us to treat this first prediction as a proxy for the final sample. While this proxy is not perfect, it is a cost-efficient approximation that we employ.

3.2.1 Clustering-based sample-pruning

A straight-forward method to find all modes of a diffusion model's distribution is to simply generate a large number of samples. However, this will always incur a relatively high cost. Ideally, we would like to achieve this large-batch behavior, while keeping the cost low. To achieve this, we sample a large initial batch of pure noise, denoise it in a single step, and discard all samples that are deemed redundant. To decide which samples are redundant, we perform k-medians clustering, with k equal to the number of modes that we expect. We discard all but the medians determined by the clustering and finish the reverse diffusion process for the corresponding samples. The benefit of this approach is that it only uses unmodified sampling steps, thus avoiding any negative impacts on image quality that modifications to the sampling trajectory could have. As a distance metric for clustering, we use the chamfer distance instead of L2 distance, since the former proved slightly better in preliminary experiments.

3.2.2 Particle Guidance

A popular approach to increase the fidelity of generated natural images are guidance terms, like classifier guidance [23] or classifier-free guidance [24]. These modify the score function in Equation 1 by an additive term:

$$\mathrm{d}\boldsymbol{x} = -t\left(\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x};t) + \alpha\ \nabla_{\boldsymbol{x}}\ g(\boldsymbol{x};t)\right)\mathrm{d}t, \tag{4}$$

where we call $g(\boldsymbol{x};t)$ the *guidance function* and α is a scalar that we call the *guidance strength*.

While classifier-free guidance increases fidelity, it *decreases* diversity [25]. Particle guidance (PG) [16] does the opposite, by improving the diversity among samples (also called *particles*) in a batch, possibly at the cost of image quality. The basic mechanic is to compute a gradient that increases the pixel-wise L2-distances between images, based on radial basis function (RBF) kernels. This approach is purely heuristic: Samples are pushed apart from each other, but the directions in which they are pushed are not aligned with any information about the data.

Let $\{\boldsymbol{x}_i | 1 \leq i \leq B\}$ be a batch of B noisy masks. To compute the value of the guidance function $g(\boldsymbol{x}_i; t)$, the denoising model first estimates the noise-free masks $\tilde{\boldsymbol{x}}_i$ in a single step for all i: $\tilde{\boldsymbol{x}}_i = D_\theta(\boldsymbol{x}_i; t)$. Next, the pairwise RBF-kernels k between those noise-free masks are computed via

$$k\left(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j; t\right) = \exp\left(-\frac{\|\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{x}}_j\|_2^2}{h_t}\right), \tag{5}$$

with $h_t = m_t^2 / \log(B)$, where m_t is the median value of $\|\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{x}}_j\|_2^2$ within the current batch of masks.

The negative kernel sum aggregates all distance relationships from mask i to all other masks:

$$g(\boldsymbol{x_i}; t) = -\sum_{j=1}^{B} k\left(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j; t\right) \tag{6}$$

Finally, the gradient of this scalar sum is computed with regards to the noisy mask $\boldsymbol{x}_i$, backpropagating through D_θ. This gradient is then used as guidance in Equation 4.

3.2.3 SPELL: SParse repELLency

In contrast to particle guidance, SPELL [17] only repels samples that are too close to each other. The original authors use the metaphor of a shield of radius r around each sample. If a sample enters this protected area around another sample, it is pushed away to an L2 distance of r.

Furthermore, SPELL is not a guidance method. Instead of adding a term to the score function, it modifies the score function in Equation 3 by changing the noise-free prediction with an additive term Δ. The modified score function for $\boldsymbol{x}_i$ becomes:

$$\text{score}_{\text{mod}}(x_i, t) = (D(\boldsymbol{x}_i; t) + \Delta_i - \boldsymbol{x}_i)/t^2, \tag{7}$$

with the additive term Δ_i computed as:

$$\Delta_i = \sum_{b, b \neq i} \sigma_{\text{relu}}\left(\frac{r}{\|\tilde{\boldsymbol{x}}_{0,i} - \tilde{\boldsymbol{x}}_{0,b}\|_2} - 1\right) \cdot (\tilde{\boldsymbol{x}}_{0,i} - \tilde{\boldsymbol{x}}_{0,b}) \tag{8}$$

This approach has the advantage of avoiding costly backward passes like in particle guidance. But since it changes the target of the denoising process at all sampling steps, there is a high risk for generating lower-quality samples. Therefore, we will limit SPELL to high-noise areas of the sampling process, to give the score function the chance to correct potential image quality issues caused by the repellence.

3.2.4 Inter-batch diversity

When generating *multiple* batches of samples, we want to encourage diversity across these batches. For this reason, we keep a memory bank containing the already-generated samples for the current input. For both particle guidance and SPELL, we add two more method variants: one that only repels samples in the current batch from the samples in the memory bank, and one that repels both from the memory bank and from samples in the current batch. This is also used in the original SPELL paper [17].

4 Multi-modal datasets

We use three different multi-modal datasets (i.e. having multiple targets per input). For MMFire and Cityscapes, we know all targets and explicitly set their probabilities. These datasets are thus very useful for evaluating different methods, but the way their annotations are generated does not represent a real-life use case. LIDC, on the other hand, offers a case of real-life ambiguous segmentation, where different annotations represent differing opinions between domain experts. The datasets also greatly differ in their inter-mode variances: MMFire's modes always overlap in the initial burned area with a medium amount of difference between modes. In Cityscapes, there are large-scale differences between modes, but also large-scale overlaps between some modes. In LIDC, there is often a large amount of overlap between different annotations, with differences being rather small. These differences will become relevant when setting the hyperparameters for SPELL in subsection 5.4.

4.1 MMFire

We generated MMFire (MM = multi-modal) with the help of the Simfire [26] simulator. For a given geolocation, it downloads from the LANDFIRE program [27, 28] real-world data that is relevant for predicting how wildfires spread, namely: dead fuel moisture at extinction, fuel bed depth, oven-dry fuel load, surface to volume ratio, and elevation. Initial wind speed and wind direction are randomly generated. An initial fire is set near the center of the image. The well-known Rothermel model [29] then deterministically predicts the spread of the fire for a desired time, based on these initial conditions.

To generate a dataset with multiple different outcomes for each initial condition, we first randomly

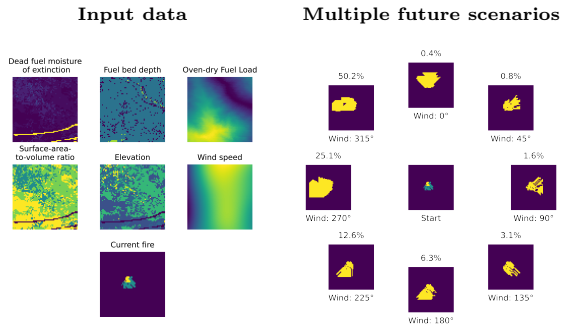

Figure 2. MMFire: We use a wildfire spread simulator to generate multiple plausible outcomes based on the current state of the fire. This is done by setting the wind direction to one of eight values across the whole 64×64 image. We impose a highly skewed probability distribution on the eight outcomes during training (see the probabilities above). This represents a difficult situation where naively sampling from the diffusion model is a very slow strategy for finding all modes.

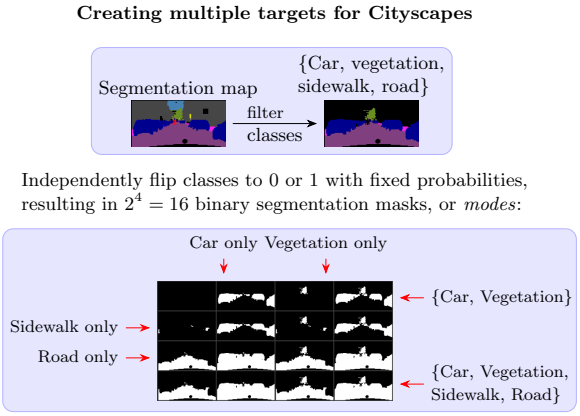

Figure 3. Multi-modal binary Cityscapes: The classes *road*, *sidewalk*, *vegetation*, and *car* are randomly flipped to the positive or negative class, with fixed probabilities, resulting in $2^4 = 16$ separate *modes* per image.

pick a geolocation in the western USA, where fuel is abundant and LANDFIRE provides data. We simulate a fire at that location for 10 minutes to have a non-trivial initial state for the fire. From this initial state, we branch into eight different futures by setting the wind direction to $i \times 45°, i \in \{0, 1, \ldots, 7\}$, constant across the simulation area of 64×64 pixels. With these eight wind directions, we then simulate another respective 10 minutes of fire spread. We use the eight final states as the targets of the dataset. Figure 2 shows an example pair of input data and eight different futures.

The models we train never get access to the wind direction. Instead, they are supposed to pick a different wind direction according to the probabilities that we set. For this, we set a highly skewed distribution on the different modes (and their associated wind direction), weighting mode i with a weight of 2^i, to create a dataset in which it is challenging to find all modes of the distribution. With naive sampling, the expected number of samples to see each mode at least once is about 307.

4.2 Cityscapes: Multi-modal, binary version

Cityscapes [14] is a semantic segmentation benchmark dataset. Inspired by previous studies [30, 31], we synthetically make the annotations multi-modal. Unlike these previous studies, we stay within the binary regime, to stay closer to the data we are interested in, namely wildfire progression data.

To make the Cityscapes dataset multi-modal, previous studies split up one class into two new, synonymous, classes, e.g. *road* becomes $road_1$ and $road_2$. During training, we randomly choose whether all *road* pixels in the current image become $road_1$ or all become $road_2$. To stay binary, we instead flip those classes between the positive and the negative class. We do this with the classes *road*, *sidewalk*, *vegetation* and *car*. All other classes are always set to negative. We flip the classes to positive with respective probabilities 5%, 25%, 75%, 95%. Each class is individually flipped on or off, thus the combination of these flip decisions for all four classes leads to $2^4 = 16$ modes for each image, assuming all four classes are present. This creates a skewed distribution, where naive sampling is a bad strategy to find all modes. Figure 3 shows all modes for an example image. While previous studies also used the class *person*, we generate segmentation masks at 64×128 pixels for faster experimentation. At this resolution, correctly annotating people is very difficult, so we drop this class.

4.3 LIDC

The LIDC dataset [15] contains CT scans of lungs and corresponding expert annotations of lung nodules. Each scan is annotated with four binary segmentation masks, that oftentimes disagree with each other. Crucially, if segmentation masks differ from each other, this represents actual disagreement between experts, stemming from epistemic uncertainty. The challenge when working with this dataset is that neither the full set of modes is not available.

5 Experiments

For more detailed information on implementation and experimental setup, please refer to the appendix.

5.1 Evaluation criteria

To evaluate the generated samples, we compute the Hungarian-Matched IoU (HM IoU), which finds the

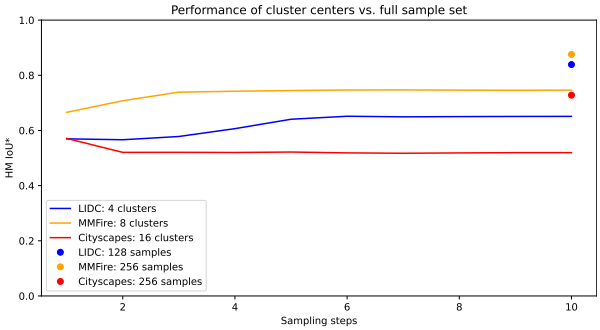

Figure 4. Applying clustering-based sample-pruning at different sampling steps: Varying after which sampling step the clustering and pruning is performed influences the final performance. For all datasets, there is large gap between evaluating only the cluster centers and evaluating the full set of generated samples. The x-axis represents the index of the sampling step. However, the noise levels at each step do not decrease linearly. See appendix for details on the noise schedule.

best match between generated and ground truth masks and computes the mean IoU across the matched masks. We modify this metric, indicated by HM IoU*, by removing duplicate ground truth masks, since our goal is to *avoid* the generation of duplicates. This is especially relevant for LIDC where most images have duplicate targets.

For MMFire and Cityscapes, we know the ground truth set of modes. With this, we additionally compute image quality and the number of distinct generated modes (see Appendix D). Results are averaged over five runs with different random seeds.

5.2 Clustering-based sample-pruning

Figure 4 shows the performance of clustering-based sample-pruning, applied at different sampling steps. For all datasets, we see a large difference between evaluating only the cluster centers (illustrated by lines) and evaluating the whole set of generated samples (illustrated by dots). This difference persists across all steps, even when the final denoised samples are chosen for clustering at the last step. For Cityscapes, the difference between clustering at the final step and using the fully denoised batch of samples at this step is 20.8% HM IoU*. This indicates that the clustering algorithm used is not able to reliably detect the modes of the distribution.

We believe that this clustering failure is the result of two interacting factors: First, the modeling error. In approximating the conditional distributions over segmentation masks, the models produce imperfect outputs. Some of them are outliers and have a distorting influence on which clusters are chosen. Second, the asymmetry in the distributions. The MMFire and Cityscapes distribution over modes have been purposefully chosen to be highly asymmetrical, to represent a challenging benchmark. The

Table 1. Clustering-based sample pruning: Varying initial batch size. We investigate the trade-off between runtime and quality when varying the number of initial samples on Cityscapes. Naive sampling generates a number of samples equal to the batch size, while clustering always generates 16 samples, but starts with a higher number of samples that are clustered.

Method	Batch size B	Image quality ↑	Distinct modes ↑	HM IoU* ↑	Runtime ↓
Naive sampling	16	0.956	12.2	0.416	0h41m
	32	0.956	13.3	0.497	1h19m
	64	0.956	14.1	0.586	2h39m
Clustering [B→16]	32	0.953	5.129	0.469	0h46m
	64	0.951	5.680	0.517	0h53m
	128	0.949	5.993	0.552	1h9m
	256	0.948	6.122	0.571	1h47m

rare modes will only show up very seldom. In the light of outliers created by the imperfect models, it is then impossible for the clustering algorithm to decide which outliers are rare modes and which are just noise to be ignored.

We conclude from these results that clustering and pruning immediately after the first sampling step is the best option. Further steps increase the performance on MMFire and LIDC, but also incur the high cost of denoising all samples in the large batch again, while the performance gain is small.

Even with only one denoising step of the full batch, our method incurs a much higher computation cost than naive sampling, while generating the same amount of outputs. We reduce the number of samples in the initial large batch, to investigate the trade-off between performance and runtime. Runtime and performance drop step-wise with reduced batch size, as shown in Table 1. However our approach comes within 1.5% HM IoU* of naively sampling 64 samples, with only 16 generated samples. Depending on the use case, this superior sample-efficiency can be a very desirable property.

The strength of our clustering approach is not a low runtime, but a high sample quality. Thus, when comparing this method with others, we decide to use 256 samples and ignore the high runtime. Table 2 shows that clustering outperforms the probabilistic UNet and naive sampling by at least 2.4% on MMFire, and 15.5% on Cityscapes. On Cityscapes it gets within 0.6% of the best performance by SPELL, even beating particle guidance by 1.8%. However, on MMFire, it is clearly outperformed by both particle guidance and SPELL. We assume that this is caused by the two factors mentioned earlier. However, on LIDC, our approach reaches the first place with a performance almost equal to that of the probabilistic UNet, outperforming the next-best diffusion-based method by 3.9%. This is likely because the rather uniform distribution of modes on LIDC makes it easier to determine correct cluster centers.

To compare with methods that generate *multiple* batches, we increase the number of clusters k to the total number of desired samples, effectively over-clustering. Table 3 shows that clustering still beats

Table 2. Single-batch performance: We generate a batch of N samples per input; N is the number of modes of the respective dataset. Methods should produce a diverse set of samples while retaining a high image quality. PG: Particle Guidance. LIDC's ground truth data does not permit the computation of the image quality and distinct modes metrics.

Method	Image quality ↑	Distinct modes ↑	HM IoU* ↑	Runtime ↓
MMFire - 1 batch × 8 samples				
Naive sampling	0.999	3.4	0.638	0h26m
Prob. UNet	0.999	3.1	0.581	0h6m
Clustering [256→8]	0.999	3.8	0.662	1h50m
PG: batch	0.999	3.9	0.690	0h29m
SPELL: batch	0.999	4.4	**0.713**	0h25m
Cityscapes - 1 batch × 16 samples				
Naive sampling	0.956	12.2	0.416	41m
Prob. UNet	0.960	3.6	0.292	6m
Clustering [256→16]	0.948	6.1	0.571	1h47m
PG: batch	0.825	15.3	0.553	46m
SPELL: batch	0.936	7.3	**0.577**	40m
LIDC - 1 batch × 4 samples				
Naive sampling			0.523	0h50m
Prob. UNet			0.573	0h5m
Clustering [128→4]	n/a	n/a	**0.574**	3h19m
PG: batch			0.528	0h57m
SPELL: batch			0.535	0h50m

Table 3. Multi-batch performance: We generate two or four batches of N samples per input; N is the number of modes of the respective dataset. For clustering, we only generate one batch, but increase the number of clusters accordingly. PG: Particle Guidance. LIDC's ground truth data does not permit the computation of the image quality and distinct modes metrics.

Method	Image quality ↑	Distinct modes ↑	HM IoU* ↑	Runtime ↓
MMFire - 2 batches ×8 samples				
Naive sampling	0.999	4.1	0.705	0h44m
ProbUNet	0.999	3.6	0.639	0h12m
Clustering [256→16]	0.999	4.4	0.728	2h8m
PG: batch	0.999	4.7	0.746	1h3m
PG: memory bank	0.999	4.5	0.733	1h4m
PG: batch & memory bank	0.999	4.6	0.743	1h4m
SPELL: batch	0.999	5.2	0.781	0h55m
SPELL: memory bank	0.999	5.0	0.765	0h55m
SPELL: batch & memory bank	0.999	5.3	**0.784**	0h55m
MMFire - 4 batches ×8 samples				
Naive sampling	0.999	4.7	0.751	1h28m
ProbUNet	0.999	4.1	0.681	0h27m
Clustering [256→32]	0.999	5.0	0.770	2h55m
PG: batch	0.999	5.3	0.789	2h35m
PG: memory bank	0.999	5.0	0.774	2h35m
PG: batch & memory bank	0.999	5.1	0.781	2h38m
SPELL: batch	0.999	6.0	0.826	2h17m
SPELL: memory bank	0.999	5.5	0.801	2h19m
SPELL: batch & memory bank	0.999	6.0	**0.830**	2h17m
Cityscapes - 2 batches ×16 samples				
Naive sampling	0.956	13.3	0.497	1h19m
ProbUNet	0.959	4.2	0.326	0h12m
Clustering [256→32]	0.948	7.3	0.661	2h23m
PG: batch	0.827	10.3	0.654	1h37m
PG: memory bank	0.892	8.9	0.642	1h37m
PG: batch & memory bank	0.867	9.6	0.679	1h36m
SPELL: batch	0.936	8.7	**0.690**	1h26m
SPELL: memory bank	0.946	7.7	0.635	1h26m
SPELL: batch & memory bank	0.936	8.7	**0.690**	1h26m
Cityscapes - 4 batches ×16 samples				
Naive sampling	0.956	14.1	0.586	2h39m
ProbUNet	0.960	4.7	0.354	0h27m
Clustering [256→64]	0.949	8.1	0.699	3h44m
PG: batch	0.827	11.8	0.704	3h41m
PG: memory bank	0.919	9.8	0.705	3h40m
PG: batch & memory bank	0.904	10.3	0.723	3h41m
SPELL: batch	0.936	9.7	**0.735**	3h19m
SPELL: memory bank	0.951	8.3	0.680	3h19m
SPELL: batch & memory bank	0.936	9.8	**0.735**	3h19m
LIDC - 2 batches ×4 samples				
Naive sampling			0.660	1h41m
ProbUNet			**0.715**	0h7m
Clustering [128→8]			0.695	4h1m
PG: batch			0.677	1h54m
PG: memory bank	n/a	n/a	0.675	1h54m
PG: batch & memory bank			0.682	1h54m
SPELL: batch			0.686	1h40m
SPELL: memory bank			0.696	1h41m
SPELL: batch & memory bank			0.697	1h41m
LIDC - 4 batches ×4 samples				
Naive sampling			0.727	3h32m
ProbUNet			**0.785**	0h13m
Clustering [128→16]			0.732	5h23m
PG: batch			0.736	3h58m
PG: memory bank	n/a	n/a	0.739	3h58m
PG: batch & memory bank			0.743	3h58m
SPELL: batch			0.740	3h30m
SPELL: memory bank			0.751	3h31m
SPELL: batch & memory bank			0.749	3h30m

naive sampling, but falls behind the other methods.

In summary, while the computational cost of this method is higher than that of competing methods, it can be useful if the goal is only to produce a low number of samples that is as representative as possible, and the underlying distribution is not heavily skewed. Furthermore, particle guidance and SPELL might be more cost-efficient at achieving a high HM IoU; but this clustering-based approach only follows the original sampling trajectory, thus avoiding potential image quality issues stemming from the interference with the sampling process. Application cases for this method could be situations where a low number of high-quality scenarios are presented to a human operator for analysis or planning, but runtime is not a big concern.

5.3 Particle guidance

Following our intuition that the determination of modes for binary segmentation masks happens mostly in the early sampling steps, we limit particle guidance to the initial steps. Table D.1 shows that limiting the guidance to the initial step performs similarly well as computing it at every step, but saves computation by avoiding the backward steps necessary for computing the guidance term.

When choosing the strength of particle guidance, Table D.1 shows a trade-off between image quality and diversity. The loss in image quality is caused by the guidance pushing samples apart without any regard for the specific dataset or the learned score function. This can easily cause the samples to move into subspaces on which the denoising model has not been trained well, leading to faulty predictions.

Analyzing the results in Table 2 and Table 3, we find that particle guidance always beats naive sampling, but tends to perform worse than SPELL. Most noticeable, on Cityscapes with 16 samples, it only reaches an image quality of 82.5%, while the next worst method is SPELL with a much higher 93.6%. At the same time, particle guidance reaches an astonishing 15.3 out of 16 distinct modes, while SPELL only reaches 7.3, showcasing the trade-off between image quality and sample diversity inherent to such methods that modify the sampling trajectory.

5.4 SPELL

SPELL's main hyperparameter is the shield radius r. It defines an L2 distance within which no other sample is allowed to fall. If a sample violates this shield, it is pushed outside of the radius. In the

Table 4. SPELL: Varying shield radius. Based on diversity statistics on the Cityscapes training set, we perform a coarse search around $r_0 = 12.7$.

Shield radius r	Image quality ↑	Distinct modes ↑	HM IoU* ↑
6.350	0.944	6.6	0.540
9.525	0.934	7.4	**0.564**
12.700	0.924	7.8	0.561
15.875	0.914	8.1	0.551

case of binary segmentation masks, this L2 distance is equivalent to the square root of the number of pixels that must differ between two samples. This correspondence provides a very direct way to specify the desired diversity that does not exist for natural images (which use the full space in [0,1]) or for latent diffusion models (where distances in latent space do not directly correspond to distances in image space). It becomes easy to set a shield radius, even if only single annotations are available in the training set.

We use the fact that we know several targets for each input to determine a starting value r_0 for the shield radius. For each input, we compute the minimum L2 distance among unique targets, and then compute the mean of these minima:

$$r_0 = \frac{1}{N}\sum_{i=1}^{N} \min\left(\{\ ||y_{i,j} - y_{i,k}||_2 \ | y_{i,j} \neq y_{i,k}\}\right) \quad (9)$$

For each dataset, we conduct a coarse hyperparameter search around r_0 to determine the parameter to use. Table 4 shows this for Cityscapes. These results confirm that r_0 as computed above is a a very good initial value. For MMFire and LIDC, we found r_0 to be the best (Table C.1).

The hard L2-limit enforced by SPELL means that samples are pushed apart without regard for how realistic the resulting images are. Especially towards the end of sampling, this is contrary to particle guidance, which slowly fades out with decreasing noise level. SPELL's stronger push seems to be unproblematic in the original SPELL paper [17], which uses latent diffusion models. These models have a downstream decoder model that maps the final latent representation to an image, which can potentially counter-act imperfect sampling outcomes. However, since we are directly applying the repellence in image space, we have to be more careful. To prevent SPELL from having a negative influence towards the end of sampling, we limit SPELL's application to $s_{\min} = 40$, where $s_{\min}$ is the highest noise level at which the guidance is still applied. In our case, that corresponds to the second sampling step. This change allows the score function to still guide the samples that were perturbed by SPELL towards more likely outcomes, leading to an improvement of 1.3% HM IoU* on Cityscapes (see Table 5).

On Cityscapes, SPELL consistently beats all other methods. Similar to particle guidance, repelling from in-batch and previously generated samples performs best, which intuitively makes sense. SPELL stays within 2.4% of the best image quality despite modifying the sampling trajectory, which is an acceptable trade-off for the large HM IoU* gains. On MMFire, particle guidance matches SPELL.

Table 5. SPELL: Limit application to high noise. We vary $s_{\min}$, the highest noise level at which SPELL is still applied, to reduce the potentially negative influence during sampling. We begin sampling from pure Gaussian noise with $\sigma_{\max} = 80$, the default for the EDM framework. $s_{\min} = \infty$ represents never using SPELL. $s_{\min} = 0$ represents always using SPELL.

$s_{\min}$	Image quality ↑	Distinct modes ↑	HM IoU* ↑	Runtime ↓
∞	0.956	12.2	0.416	41m
70	0.938	7.0	0.571	41m
40	0.936	7.3	**0.577**	40m
20	0.935	7.4	0.571	41m
10	0.935	7.4	0.571	41m
0	0.934	7.4	0.564	0h40m

On LIDC, both particle guidance and SPELL clearly underperform the probabilistic UNet. Further work may be needed to tune our diffusion model on LIDC, though this performance gap is reversed on the other datasets. However, the diversity-encouraging methods always outperforms naive sampling from the diffusion model. Thus, it always seems advisable to use SPELL, even on datasets with fairly symmetric distributions, like LIDC.

6 Future work

While MMFire is very useful for benchmarking, it lacks *realistic* diversity. Such diversity can be added to real-world datasets by simulating alternative futures at each step. For our clustering-based approach, density-based clustering algorithms could make it easier to detect low-probability outliers as separate clusters. Furthermore, we only investigated training-free methods. Training-based approaches might improve upon these. Lastly, consistency models [32] might provide better one-step approximations, which all investigated methods rely upon.

7 Conclusion

We introduced MMFire, a simulated multi-modal wildfire spread dataset. We demonstrated that particle guidance and SPELL substantially improve prediction diversity with minimal quality loss, with SPELL achieving 7.5% HM IoU* gains on MMFire and 16.1% on Cityscapes. Our clustering-based pruning provides an alternative that is computationally intense, but preserves image quality better. These advances enable more sample-efficient multi-modal modeling of wildfire spread.

References

[1] F. Huot, R. L. Hu, N. Goyal, T. Sankar, M. Ihme, and Y.-F. Chen. "Next Day Wildfire Spread: A Machine Learning Dataset to Predict Wildfire Spreading From Remote-Sensing Data". In: *IEEE Transactions on Geoscience and Remote Sensing* 60 (2022). Conference Name: IEEE Transactions on Geoscience and Remote Sensing, pp. 1–13. ISSN: 1558-0644. DOI: 10.1109/TGRS.2022.3192974.

[2] S. Gerard, Y. Zhao, and J. Sullivan. "WildfireSpreadTS: A dataset of multi-modal time series for wildfire spread prediction". In: *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. 2023. URL: https://openreview.net/forum?id=RgdGkPRQ03.

[3] M. Rösch, M. Nolde, T. Ullmann, and T. Riedlinger. "Data-Driven Wildfire Spread Modeling of European Wildfires Using a Spatiotemporal Graph Neural Network". In: *Fire* 7.6 (June 2024). Number: 6 Publisher: Multidisciplinary Digital Publishing Institute, p. 207. ISSN: 2571-6255. DOI: 10.3390/fire7060207. URL: https://www.mdpi.com/2571-6255/7/6/207 (visited on 07/16/2025).

[4] T. Amit, T. Shaharbany, E. Nachmani, and L. Wolf. *SegDiff: Image Segmentation with Diffusion Probabilistic Models*. Sept. 7, 2022. DOI: 10.48550/arXiv.2112.00390. arXiv: 2112.00390[cs]. URL: http://arxiv.org/abs/2112.00390 (visited on 08/30/2023).

[5] J. Wolleb, R. Sandkühler, F. Bieder, P. Valmaggia, and P. C. Cattin. "Diffusion Models for Implicit Image Segmentation Ensembles". In: *Proceedings of The 5th International Conference on Medical Imaging with Deep Learning*. Ed. by E. Konukoglu, B. Menze, A. Venkataraman, C. Baumgartner, Q. Dou, and S. Albarqouni. Vol. 172. Proceedings of Machine Learning Research. PMLR, July 6, 2022, pp. 1336–1348. URL: https://proceedings.mlr.press/v172/wolleb22a.html.

[6] T. Chen, C. Wang, and H. Shan. "Berdiff: Conditional bernoulli diffusion model for medical image segmentation". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2023, pp. 491–501.

[7] T. Chen, L. Li, S. Saxena, G. Hinton, and D. J. Fleed. "A Generalist Framework for Panoptic Segmentation of Images and Videos". In: *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2023 IEEE/CVF International Conference on Computer Vision (ICCV). Paris, France: IEEE, Oct. 1, 2023, pp. 909–919. ISBN: 979-8-3503-0718-4. DOI: 10.1109/ICCV51070.2023.00090. URL: https://ieeexplore.ieee.org/document/10377333/ (visited on 09/01/2025).

[8] T. Chen, R. ZHANG, and G. Hinton. "Analog bits: Generating discrete data using diffusion models with self-conditioning". In: *The eleventh international conference on learning representations*. 2023. URL: https://openreview.net/forum?id=3itjR9QxFw.

[9] Z. Lai, Y. Duan, J. Dai, Z. Li, Y. Fu, H. Li, Y. Qiao, and W. Wang. *Denoising Diffusion Semantic Segmentation with Mask Prior Modeling*. June 22, 2023. arXiv: 2306.01721[cs]. URL: http://arxiv.org/abs/2306.01721 (visited on 10/20/2023).

[10] A. Rahman, J. M. J. Valanarasu, I. Hacihaliloglu, and V. M. Patel. "Ambiguous Medical Image Segmentation Using Diffusion Models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2023, pp. 11536–11546.

[11] L. Zbinden, L. Doorenbos, T. Pissas, A. T. Huber, R. Sznitman, and P. Márquez-Neila. "Stochastic Segmentation with Conditional Categorical Diffusion Models". In: *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2023 IEEE/CVF International Conference on Computer Vision (ICCV). Paris, France: IEEE, Oct. 1, 2023, pp. 1119–1129. ISBN: 979-8-3503-0718-4. DOI: 10.1109/ICCV51070.2023.00109. URL: https://ieeexplore.ieee.org/document/10376866/ (visited on 01/27/2025).

[12] J. Wu, R. Fu, H. Fang, Y. Zhang, Y. Yang, H. Xiong, H. Liu, and Y. Xu. "Medsegdiff: Medical image segmentation with diffusion probabilistic model". In: *Medical imaging with deep learning*. PMLR, 2024, pp. 1623–1639.

[13] W. Yu, A. Ghosh, T. S. Finn, R. Arcucci, M. Bocquet, and S. Cheng. *A Probabilistic Approach to Wildfire Spread Prediction Using a Denoising Diffusion Surrogate Model*. July 1, 2025. DOI: 10.48550/arXiv.2507.00761. arXiv: 2507.00761[cs]. URL: http://arxiv.org/abs/2507.00761 (visited on 09/11/2025).

[14] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016, pp. 3213–3223. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Cordts_The_Cityscapes_

Dataset_CVPR_2016_paper.html (visited on 03/02/2025).

[15] S. G. Armato III, G. McLennan, L. Bidaut, M. F. McNitt-Gray, C. R. Meyer, A. P. Reeves, B. Zhao, D. R. Aberle, C. I. Henschke, E. A. Hoffman, E. A. Kazerooni, H. MacMahon, E. J. R. van Beek, D. Yankelevitz, A. M. Biancardi, P. H. Bland, M. S. Brown, R. M. Engelmann, G. E. Laderach, D. Max, R. C. Pais, D. P.-Y. Qing, R. Y. Roberts, A. R. Smith, A. Starkey, P. Batra, P. Caligiuri, A. Farooqi, G. W. Gladish, C. M. Jude, R. F. Munden, I. Petkovska, L. E. Quint, L. H. Schwartz, B. Sundaram, L. E. Dodd, C. Fenimore, D. Gur, N. Petrick, J. Freymann, J. Kirby, B. Hughes, A. Vande Casteele, S. Gupte, M. Sallam, M. D. Heath, M. H. Kuhn, E. Dharaiya, R. Burns, D. S. Fryd, M. Salganicoff, V. Anand, U. Shreter, S. Vastagh, B. Y. Croft, and L. P. Clarke. "The Lung Image Database Consortium (LIDC) and Image Database Resource Initiative (IDRI): A Completed Reference Database of Lung Nodules on CT Scans". In: *Medical Physics* 38.2 (2011), pp. 915–931. ISSN: 2473-4209. DOI: 10.1118/1.3528204. URL: https://onlinelibrary.wiley.com/doi/abs/10.1118/1.3528204 (visited on 02/26/2025).

[16] G. Corso, Y. Xu, V. D. Bortoli, R. Barzilay, and T. Jaakkola. "Particle Guidance: non-I.I.D. Diverse Sampling with Diffusion Models". In: NeurIPS 2023 Workshop on Deep Learning and Inverse Problems. Nov. 3, 2023. URL: https://openreview.net/forum?id=hEyIHsyZ9F (visited on 11/24/2024).

[17] M. Kirchhof, J. Thornton, L. Béthune, P. Ablin, E. Ndiaye, and M. Cuturi. "Shielded Diffusion: Generating Novel and Diverse Images using Sparse Repellency". In: Forty-second International Conference on Machine Learning. June 18, 2025. URL: https://openreview.net/forum?id=XAckVo0iNj (visited on 08/28/2025).

[18] J. Wu, W. Ji, H. Fu, M. Xu, Y. Jin, and Y. Xu. "MedSegDiff-V2: Diffusion-Based Medical Image Segmentation with Transformer". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 38.6 (Mar. 24, 2024). Number: 6, pp. 6030–6038. ISSN: 2374-3468. DOI: 10.1609/aaai.v38i6.28418. URL: https://ojs.aaai.org/index.php/AAAI/article/view/28418 (visited on 07/18/2024).

[19] S. Sadat, J. Buhmann, D. Bradley, O. Hilliges, and R. M. Weber. "CADS: Unleashing the Diversity of Diffusion Models through Condition-Annealed Sampling". In: The Twelfth International Conference on Learning Representations. Oct. 13, 2023. URL: https://openreview.net/forum?id=zMoNrajk2X (visited on 02/19/2024).

[20] K. Pandey, Y. Hold-Geoffroy, M. Gadelha, N. J. Mitra, K. Singh, and P. Guerrero. "Motion modes: What could happen next?" In: *Proceedings of the computer vision and pattern recognition conference*. 2025, pp. 2030–2039.

[21] J. Lu, R. Teehan, and M. Ren. "Procreate, don't reproduce! propulsive energy diffusion for creative generation". In: *European conference on computer vision*. Springer, 2024, pp. 397–414.

[22] T. Karras, M. Aittala, T. Aila, and S. Laine. "Elucidating the Design Space of Diffusion-Based Generative Models". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., 2022, pp. 26565–26577. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/a98846e9d9cc01cfb87eb694d946ce6b-Paper-Conference.pdf.

[23] P. Dhariwal and A. Nichol. *Diffusion Models Beat GANs on Image Synthesis*. June 1, 2021. DOI: 10.48550/arXiv.2105.05233. arXiv: 2105.05233. URL: http://arxiv.org/abs/2105.05233 (visited on 11/18/2024).

[24] J. Ho and T. Salimans. "Classifier-Free Diffusion Guidance". In: *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*. 2021.

[25] T. Karras, M. Aittala, T. Kynkäänniemi, J. Lehtinen, T. Aila, and S. Laine. "Guiding a Diffusion Model with a Bad Version of Itself". In: The Thirty-eighth Annual Conference on Neural Information Processing Systems. Nov. 6, 2024. URL: https://openreview.net/forum?id=bg6fVPVs3s&referrer=%5Bthe%20profile%20of%20Tero%20Karras%5D(%2Fprofile%3Fid%3D~Tero_Karras1) (visited on 11/18/2024).

[26] M. Doyle, M. Threet, M. Dotter, C. Kempis, A. Tapley, and T. Welsh. *SimFire*. Version 2.0.1. July 2024. URL: https://github.com/mitrefireline/simfire.

[27] LANDFIRE. *Anderson Fire Behavior Fuel Model (FBFM13) Layer*. LANDFIRE 2.0.0. U.S. Department of the Interior, Geological Survey, and U.S. Department of Agriculture, 2016. URL: https://landfire.gov/ (visited on 09/04/2025).

[28] LANDFIRE. *Elevation Layer*. LANDFIRE 2.2.0. U.S. Department of the Interior, Geological Survey, and U.S. Department of Agriculture, 2020. URL: https://landfire.gov/ (visited on 09/04/2025).

[29] R. C. Rothermel. "A mathematical model for predicting fire spread in wildland fuels". In: *Res. Pap. INT-115. Ogden, UT: U.S. Department of Agriculture, Intermountain Forest and Range Experiment Station. 40 p.* 115 (1972). URL: https://www.fs.usda.gov/research/treesearch/32533 (visited on 11/14/2022).

[30] S. Kohl, B. Romera-Paredes, C. Meyer, J. De Fauw, J. R. Ledsam, K. Maier-Hein, S. M. A. Eslami, D. Jimenez Rezende, and O. Ronneberger. "A Probabilistic U-Net for Segmentation of Ambiguous Images". In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/hash/473447ac58e1cd7e96172575f48dca3b-Abstract.html (visited on 11/12/2024).

[31] Z. Gao, Y. Chen, C. Zhang, and X. He. *Modeling Multimodal Aleatoric Uncertainty in Segmentation with Mixture of Stochastic Experts*. Feb. 26, 2023. DOI: 10.48550/arXiv.2212.07328. arXiv: 2212.07328[cs]. URL: http://arxiv.org/abs/2212.07328 (visited on 08/23/2024).

[32] Y. Song, P. Dhariwal, M. Chen, and I. Sutskever. "Consistency models". In: *Proceedings of the 40th international conference on machine learning*. Ed. by A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett. Vol. 202. Proceedings of machine learning research. PMLR, July 23, 2023, pp. 32211–32252. URL: https://proceedings.mlr.press/v202/song23a.html.

[33] J. Borovec, W. Falcon, A. Nitta, A. H. Jha, otaj, A. Brundyn, D. Byrne, N. Raw, S. Matsumoto, T. Koker, B. Ko, A. Oke, S. Sundrani, Baruch, C. Clement, C. POIRET, R. Gupta, H. Aekula, A. Wälchli, A. Phatak, I. Kessler, J. Wang, J. Lee, S. Mehta, Z. Yang, G. O'Donnell, and zlapp. *Lightning-AI/lightning-bolts: Minor patch release*. Version 0.6.0.post1. Dec. 2022. DOI: 10.5281/zenodo.7447212. URL: https://doi.org/10.5281/zenodo.7447212.

[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[35] W. Falcon and The PyTorch Lightning team. *PyTorch Lightning*. Version 1.4. doi: 10.5281/zenodo.3828935, https://www.pytorchlightning.ai, Last visited 2021-09-17. Mar. 30, 2019. URL: https://www.pytorchlightning.ai (visited on 09/17/2021).

[36] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. *Score-Based Generative Modeling through Stochastic Differential Equations*. Feb. 10, 2021. arXiv: 2011.13456[cs,stat]. URL: http://arxiv.org/abs/2011.13456 (visited on 02/16/2024).

[37] L. N. Smith and N. Topin. "Super-convergence: very fast training of neural networks using large learning rates". In: *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*. Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications. Vol. 11006. SPIE, May 10, 2019, pp. 369–386. DOI: 10.1117/12.2520589. URL: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/11006/1100612/Super-convergence--very-fast-training-of-neural-networks-using/10.1117/12.2520589.full (visited on 08/30/2025).

A Datasets

A.1 MMFire

The dataset consists of 9608 input samples of size $7 \times 64 \times 64$, each associated with eight simulated future fire spread segmentation masks of size 64×64. We use a split of 5000 training samples, 2500 validation samples, and 2108 test samples. We do not apply any augmentations. Any augmentations applied will need to make sure that the wind direction is correctly transformed, e.g. in rotations or flips.

A.2 Cityscapes

Our implementation of the Cityscapes dataset is based on the existing Lightning Bolts [33] Cityscapes data module. We only use the 5000 images with fine annotations, resized to 64×128 for faster experiments. We keep the split of 2975 training images,

500 validation images, and 1525 test images. We use the semantic segmentation labels as a starting point, before filtering down to the four classes which we randomly flip between the positive and negative class. We only use color jittering as augmentations. We train with batch size 32 and parameterize the log-normal training noise distribution with a mean of $\mu_{\text{train}} = 1.5$.

A.3 LIDC

To download and preprocess the LIDC dataset, we followed the steps indicated by the authors of [31] at https://github.com/gaozhitong/MoSE-AUSeg/, and use their dataset class to load the data. The dataset contains 9794 training images, 2314 validation images, and 2988 test images. It includes random horizontal and vertical flipping and rotation by up to 10°, all applied to both image and label simultaneously. The images and labels have a size of 128×128 pixels.

B Implementation details

Our code base is implemented in PyTorch [34] and PyTorch Lightning [35]. For the diffusion model, we use the official EDM [22] repository at https://github.com/NVlabs/edm.

All experiments were run on NVIDIA A40 GPUs, provided as part of a scientific computational cluster that is credited in the acknowledgments.

During sampling, the ODE is solved by starting from $\boldsymbol{x} \sim \mathcal{N}$ at $\sigma_{\max} = 80$ and numerically integrating to $\sigma_{\min} = 0.002$, using the 2nd order Heun solver from EDM. Unless specifically noted, we only use deterministic sampling, i.e. we set $S_{\text{churn}} = 0$.

We use the NCSN++ architecture [36], as implemented in the EDM code base and EDM preconditioning, modified to take an image as conditional information by concatenating it to the noisy mask $\boldsymbol{x}$ that is being generated. Only the base multiplier for the number of features is modified for the different datasets: For Cityscapes, we use the default value of 128, since the conditioning image and output distribution are rather complex. For MMFire and LIDC, we use 64.

C Experimental details

C.1 Training the diffusion models

Unless mentioned otherwise, we train the base models with AdamW with learning rate 1e-4, and $\beta_1 = 0.9, \beta_2 = 0.99$. For Cityscapes, we train for 400 epochs, for LIDC, we train for 200 epochs, for MMFire, we train for 1000 epochs. We compute the validation loss after each training epoch and keep the model checkpoint with the lowest validation loss. During training, this validation loss is computed by randomly sampling noise levels according to the training noise distribution for each conditioning. The training noise distribution is always a log-normal distribution, with standard deviation $\sigma_{\text{train}} = 1.2$, which is the default value in EDM, and mean μ_{train}, which we vary between different runs. Note that these parameters refer to the normal distribution, the samples of which are then exponentiated. The mean, mode, and standard deviation of samples drawn from the log-normal distribution are different.

For each dataset, we train several models, varying the mean (μ_{train}) of the log-normal distribution used for sampling the noise levels during training. In preliminary experiments, we observed that performing model selection simply via lowest validation loss did not lead to a good calibration of the distribution over modes. We therefore perform the model selection with regards to the highest alignment of the sample distribution over modes with the training distribution. For this, we sample 64 segmentation masks per conditioning (e.g. per RGB image in Cityscapes), and compare the distribution over modes with the known ground truth distribution via total variation difference metric (see next paragraph). We also do not perform model selection with regards to HM IoU*. Choosing a model with the highest HM IoU* for a highly skewed distribution would mean that the skewedness is likely not properly represented by the model, even though we of course want to achieve a high HM IoU* in the end. In practice, the models we choose tend to still be among the best in terms of HM IoU* computed over the 64 samples. For datasets for which we only know a set of segmentation masks per image, but not the actual probabilities per mask, we can take the pixel-wise mean across generated masks and compare to the pixel-wise mean across ground truth labels, as a measure of calibration. This is the case for LIDC, where we use the Brier score to select a model. We end up choosing the models trained with the following parameters: MMFire: $\mu_{\text{train}} = 0.5$, Cityscapes: $\mu_{\text{train}} = 1.5$; LIDC: $\mu_{\text{train}} = 1.0$.

To perform **model selection**, we estimate how well a large batch of generated segmentation masks $\{x_i' | 0 \leq i \leq B \; B \in \mathbb{N}\}$ follows the training distribution over modes. For **Cityscapes**, we have flipped all classes separately, thus we want to estimate how well the model follows the per-class Bernoulli flip probabilities. We estimate the flip probabilities from the B generated masks for each class separately. For this, we compute the per-class IoU between generated mask and indicator mask of the respective class, e.g. a mask that is 1 for the road class, and 0 otherwise. Then, we threshold the IoU at 0.5 to decide whether the generated mask represents a choice of the positive or negative mode for the given class.

From these per-mask modes for each class, we estimate the per-batch distribution of modes and compare it to the true Bernoulli distributions via mean total variation distance (TVD). To compute this distance, we compute the mean distance between the C paired distributions: $\text{TVD}_{\text{mean}}(\{(p_c, q_c)|1 \leq c \leq C)\}) = \frac{1}{C}\sum_{c=1}^{C}|p_c - q_c|$, where p_c, q_c are the true and estimated Bernoulli probability for flipping class c to the positive class. For MMFire, we do conceptually the same, except that we assign a single mode to each image, and we compare a single empirical and a single ground truth *categorical* distribution, instead of several Bernoulli distributions.

C.2 Sampling from the diffusion models

For all models, we use the EDM sampling schedule, parameterized by $\rho = 7$ and $n = 10$ timesteps. More steps did not lead to better results. This is likely due to the very conditioning signal that is much stronger than in the case of natural images, conditioned on a text prompt, for example. There are many different images that are consistent with 'dog on the beach wearing sunglasses', but very few segmentation masks that are pixel-perfectly aligned with the input and correctly distinguish between the positive and negative classes.

C.3 Particle guidance and SPELL

For particle guidance, we use the a guidance strength $\alpha = 10$ for all experiments. We ran a coarse grid search across $\alpha \in \{10, 100, 1000\}$ for all datasets separately, but found that $\alpha = 10$ performed best for each of them.

For SPELL, we compute r_0 from the respective training dataset, according to Equation 9. Starting from this, we run a coarse grid search across $\{0.5r_0, 0.75r_0, r_0, 1.25r_0\}$ and choose the best one as r for all experiments. Table C.1 shows the corresponding values. For both MMFire and LIDC, using r_0 proved best among the investigated values. For Cityscapes, we used $0.75r_0$. Which exact value proves best depends on the exact distribution of distances. An option would be to replace the mean with the minimum in Equation 9. However, for many cases, SPELL would then not ensure enough diversity (see Table 4). Thus, r_0 can be taken as a strong starting point, but, depending on the concrete distance distribution of the dataset, a better value might exist near r_0.

Table C.1. SPELL: Shield radii used in experiments. r_0 is an initial estimate for a good shield radius determined from dataset diversity statistics in Equation 9. r is the best value we found in a coarse grid search around r_0. For MMFire and LIDC, r_0 was the best value we found. For Cityscapes, we found in Table 4 that $0.75r_0$ performs slightly better.

Dataset	r_0	r
MMFire	9.525	9.525
Cityscapes	12.700	9.525
LIDC	6.000	6.000

D Evaluation

Metrics for the main results in Table 2 and Table 3 are computed on the test sets. All other tables and figures are treated as part of hyperparameter search or optimization, therefore they are computed on the validation sets. For Cityscapes, the test set segmentation masks are not public, therefore we can not compute any of our metrics on the Cityscapes test set. Therefore, we use the Cityscapes validation set everywhere, instead. Since we mostly care about the relative performance of the methods on the same dataset, this still seems serviceable.

While we mainly focus our evaluations on HM IoU*, which is a combined measure of image quality and diversity, we additionally use explicit measures of these two qualities on MMFire and Cityscapes. For each generated sample, we compute which ground truth mode is closest to the sample. Within a batch of samples, we then count the number of **distinct modes** (or unique modes) that were generated. Since this is a pure argmax computation, a sample that is closest to one particular mode might still have very low quality. Thus, this metric can be very noisy and should always be interpreted with regards to an image quality metric.

For **image quality**, we compute a pixel-wise union of all modes for the current input, to determine which pixels are allowed to be part of the positive class, and which ones should always be assigned the negative class. We want to penalize samples which set pixels to positive that should never be positive. Let this union image be y_{union}. We take the complement to receive $\bar{y}_{union}$, which is 0 in all pixels that are positive in at least one mode, and 1 otherwise. Let x_i be a sample to evaluate, then we compute the image quality metric as $1 - \text{IoU}(x_i, y_{\text{union}})$.

The runtimes we measure include all of the time it takes to run the respective job on a shared scientific computation cluster, including loading the respective model and computing the metrics. The runtimes can vary, depending on the load imposed by other jobs using the shared resources. We assume that they are still useful as broad indications of how much longer certain methods take than others.

D.1 Probabilistic UNet

Following previous work [10, 11], we use the probabilistic UNet [30] as a baseline for ambiguous

Table D.1. Particle guidance: Varying guidance strength α. We investigate the trade-off between image quality and diversity on Cityscapes. Applying guidance on the first step only saves several backward passes, but performs almost exactly the same.

Guidance steps	α	Image quality ↑	Distinct modes ↑	HM IoU* ↑
None	0	0.956	12.2	0.416
First	1	0.954	13.3	0.460
First	10	0.939	14.4	**0.558**
First	100	0.824	15.3	0.553
First	1000	0.691	15.4	0.445
all	1	0.954	13.3	0.460
all	10	0.938	14.4	**0.558**
all	100	0.818	15.3	0.551
all	1000	0.685	15.4	0.445

segmentation. We use an unofficial PyTorch re-implementation[1] to train one model per dataset. In our experiments, we try to stay as close as possible to the original hyperparameter choices that were made for training on LIDC in the original probabilistic UNet paper. Unfortunately, we found that training was rather unstable in our case. To remedy this, we removed the custom weight initialization and instead relied on PyTorch's default initialization schemes. Furthermore, for Cityscapes and MMFire, we switched to the OneCycleLR[37] learning rate scheduler and increased the learning rate from 1e-4 to 1e-3 since convergence was extremely slow otherwise.

We generally tried to see how well the probabilistic UNet works 'out of the box'. Since the results for Cityscapes were much worse than the diffusion model, we also attempted to adjust the hyperparameter that was most likely to be responsible for the low performance, which is the model capacity, given that Cityscapes has 16 modes and the differences between them are relatively large. We therefore doubled the latent dimension from 6 to 12 and doubled the hidden dimensions throughout the model. However, this did not improve the results. The main problem we see in the generated samples is that the road is never predicted as positive. In training, this class has a probability of 5% to be observed as the positive class. Given that it also takes the largest amount of pixels among the classes, never predicting it as positive means a large loss in HM IoU* in half of the modes, explaining the low performance.

[1] https://github.com/stefanknegt/Probabilistic-Unet-Pytorch