Appendices

Table of Contents

A Theoretical Rank Analysis: Proofs

Lemma A.1 (Rank Increase)**.** *Let the logits $\boldsymbol{\eta} \in \mathbb{R}^{kd}$ be low-rank Gaussian distributed $\boldsymbol{\eta} \mid \mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}(\mathbf{x}), \boldsymbol{\Sigma}(\mathbf{x})) = p_{H|X}$, where the covariance matrix $\boldsymbol{\Sigma}(\mathbf{x}) \in \mathbb{R}^{kd \times kd}$ has rank $\operatorname{rank}(\boldsymbol{\Sigma}(\mathbf{x})) = r$. If we define $\mathbf{y} = g(\boldsymbol{\eta})$, where $g = \operatorname{softmax}_k$, the rank of the covariance matrix $\operatorname{Cov}(\mathbf{y})$ of the pushforward distribution $g_{\#}p_{H|X}$ increases under the condition:*

$$\operatorname{rank}(\operatorname{Cov}(\mathbf{y})) > r \iff r < d(k-1). \tag{21}$$

Proof. First recall that we use $\operatorname{softmax}_k(\cdot)$ to denote that the softmax function is applied row-wise across the k dimension (i.e. over the number of categories k), after reshaping the input logits $\boldsymbol{\eta} \in \mathbb{R}^{kd}$ to $\boldsymbol{\eta} \in \mathbb{R}^{k \times d}$. More formally:

$$\mathbf{y}_{:,j} = \operatorname{softmax}(\boldsymbol{\eta}_{:,j}), \qquad \text{where} \qquad y_{ij} = \frac{e^{\eta_{ij}}}{\sum_{l=1}^{k} e^{\eta_{lj}}}, \qquad \text{for} \quad i = 1, \ldots, k,\ j = 1, \ldots, d. \tag{22}$$

Now let $\mathbf{f} : \mathbb{R}^{kd} \to (0,1)^{kd}$ be the composition of $\operatorname{softmax}_k(\boldsymbol{\eta})$ then flattening $\mathbf{y} \in (0,1)^{k \times d}$ back to $\mathbf{y} \in (0,1)^{kd}$. The resulting Jacobian $\mathbf{J}_{\mathbf{f}} \in \mathbb{R}^{kd \times kd}$ of $\mathbf{f}$ is a sparse matrix given by:

$$\mathbf{J}_{\mathbf{f}} = \begin{bmatrix} \frac{\partial y_1}{\partial \eta_1} & \cdots & \frac{\partial y_1}{\partial \eta_{kd}} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_{kd}}{\partial \eta_1} & \cdots & \frac{\partial y_{kd}}{\partial \eta_{kd}} \end{bmatrix}, \qquad \text{where} \qquad \frac{\partial y_i}{\partial \eta_j} \neq 0 \quad \text{if} \quad j \equiv i \pmod{k}. \tag{23}$$

In words, $\mathbf{J}_{\mathbf{f}}$ has a total of dk^2 non-zero entries, including the diagonal and a checkerboard-like pattern with column-wise intervals of stride k, caused by the flattening operation.

Intuitively, $\mathbf{f}$ non-linearly couples each element in $\boldsymbol{\eta}$ with $k-1$ number of other elements in $\boldsymbol{\eta}$ to produce $\mathbf{y}$, i.e. each $\boldsymbol{\eta}_{:,j} \in \mathbb{R}^{k\times 1}$ is independently mapped to the $(k-1)$-dimensional simplex Δ^{k-1} via an element-wise softmax, for $j = 1, 2, \ldots, d$. As a result, we can conclude that $\operatorname{rank}(\mathbf{J_f}) = d(k-1)$.

Considering a Taylor expansion of $\operatorname{Cov}(\mathbf{y})$, we know that the first-order term only contributes linearly to the result:

$$\operatorname{Cov}(\mathbf{y}) = \mathbf{J_f}\boldsymbol{\Sigma}(\mathbf{x})\mathbf{J_f}^\top + \cdots, \qquad \text{and} \qquad \operatorname{rank}(\mathbf{J_f}\boldsymbol{\Sigma}(\mathbf{x})\mathbf{J_f}^\top) = \min(r, d(k-1)) = r. \tag{24}$$

Since $\mathbf{f}$ is non-linear, the higher-order terms in the Taylor expansion (and their rank) must be non-zero. Thus, by the rank subadditivity property: $\operatorname{rank}(A+B) \leq \operatorname{rank}(A) + \operatorname{rank}(B)$, we conclude that $\operatorname{rank}(\operatorname{Cov}(\mathbf{y})) > r$ iff $r < d(k-1)$. □

Theorem A.2 (Sublinear Growth of the Effective Rank)**.** *Given a low-rank Gaussian covariance matrix $\boldsymbol{\Sigma}(\mathbf{x}) \in \mathbb{R}^{kd\times kd}$ with initial rank $\operatorname{rank}(\boldsymbol{\Sigma}(\mathbf{x})) = r < d(k-1)$. The increase in the effective rank $\operatorname{erank}(\operatorname{Cov}(\mathbf{y}))$, in the sense of Lemma 4.1, grows sublinearly with respect to the initial rank r.*

Proof. Recall that the rank of $\boldsymbol{\Sigma}(\mathbf{x}) \in \mathbb{R}^{kd\times kd}$ is by definition given by:

$$\operatorname{rank}(\boldsymbol{\Sigma}(\mathbf{x})) = \sum_{i=1}^{kd} \mathbb{I}(\sigma_i > 0), \qquad \text{where} \qquad \sigma_1 \geq \sigma_2 \geq, \ldots, \sigma_{kd} \geq 0 \tag{25}$$

are the singular values of $\boldsymbol{\Sigma}(\mathbf{x})$, and $\mathbb{I}(\cdot)$ is the indicator function. Since $\operatorname{rank}(\boldsymbol{\Sigma}(\mathbf{x})) = r$ we have $\sigma_1 \geq, \ldots, \geq \sigma_r > 0$.

Given that the non-linear pushforward operation defined in Lemma 4.1 is rank-increasing (as long as $r < d(k-1)$), we can assert that the final rank, $\operatorname{rank}(\operatorname{Cov}(\mathbf{y})) = r_1$, grows *at least linearly* w.r.t. the initial rank $\operatorname{rank}(\boldsymbol{\Sigma}(\mathbf{x})) = r$. This is intuitive as each non-zero singular value of $\boldsymbol{\Sigma}(\mathbf{x})$ contributes at least one additional rank increment to $\operatorname{rank}(\operatorname{Cov}(\mathbf{y}))$.

This holds for the *effective rank* $\operatorname{erank}(\operatorname{Cov}(\mathbf{y}))$ if and only if the distribution of singular values of $\operatorname{Cov}(\mathbf{y})$, p, is uniform:

$$\operatorname{erank}(\operatorname{Cov}(\mathbf{y})) = \operatorname{rank}(\operatorname{Cov}(\mathbf{y})) \iff p_i = \frac{1}{r_1}, \quad \text{for} \quad i = 1, 2, \ldots, r_1, \tag{26}$$

which is straightforward to show by substituting p into the definition of effective rank (c.f. Definition 4.2):

$$H(p) = -\sum_{i=1}^{r_1} \frac{1}{r_1} \log\left(\frac{1}{r_1}\right) = \log r_1 \implies \operatorname{erank}(\operatorname{Cov}(\mathbf{y})) = e^{H(p)} = r_1 = \operatorname{rank}(\operatorname{Cov}(\mathbf{y})). \tag{27}$$

Alternatively, when the singular value distribution p is non-uniform, e.g. skewed, we have that:

$$\sigma_1 \gg \sigma_2 \geq, \ldots, \geq \sigma_{r_1} > 0 \implies H(p) < \log r_1 \implies \operatorname{erank}(\operatorname{Cov}(\mathbf{y})) < \operatorname{rank}(\operatorname{Cov}(\mathbf{y})), \tag{28}$$

since the uniform is the maximum entropy distribution and $H(p)$ is bounded above by $\log r_1$. For a random matrix with continuous entries and $kd > 1$, the probability that all its singular values are equal is zero; so the singular value distribution p is almost surely non-uniform. Thus, since linear growth holds iff p is uniformly distributed (c.f. Eq. (26)), the final effective rank $\operatorname{erank}(\operatorname{Cov}(\mathbf{y})) = r_1$ must grow *sublinearly* w.r.t. the initial rank $\operatorname{rank}(\boldsymbol{\Sigma}(\mathbf{x})) = r$, concluding the proof. □

Remark A.3. One way to intuit this result is by recalling that entropy grows logarithmically with the number of dimensions in uniform distributions. For non-uniform distributions, the growth of entropy can be slower, depending on how the additional event (represented by, say, $\sigma_{r+1} > 0$ in our case) relates to the existing probabilities $p_1, p_2, \ldots, p_r$.

B Flow Stochastic Segmentation Networks: Proofs

We begin by reviewing autoregressive image models, then proceed to theoretical results and design options for Flow-SSNs.

B.1 Autoregressive Image Models

Autoregressive models factorise joint probability distributions into a product of conditional distributions using the chain rule of probability. For example, by representing an image $\mathbf{y}$ as a sequence of pixels $y_1, y_2, \ldots, y_T$, we can estimate the joint pixel distribution by maximum likelihood estimation on the parameters of a model:

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log p(\mathbf{y}_i; \boldsymbol{\theta}), \qquad p(\mathbf{y}_i; \boldsymbol{\theta}) = \prod_{t=1}^{T} p(y_{i,t} \mid y_{i,1}, \ldots, y_{i,t-1}; \boldsymbol{\theta}_t). \tag{29}$$

Autoregressive image models [74, 87, 88] can estimate arbitrary joint pixel distributions but are computationally costly to sample from as each pixel must be generated sequentially. Discrete-time autoregressive Flow-SSNs avoid sequential sampling.

Proposition B.1 (Full Covariance Flow Transformation)**.** *Let* $\mathbf{u} = (u_1, u_2, \ldots, u_d)^\top$ *be a Gaussian random vector with diagonal covariance* $\mathcal{N}(\boldsymbol{\mu}, \operatorname{diag}(\boldsymbol{\sigma}^2))$*. A linear autoregressive model is sufficient to transform* $\mathbf{u}$ *into a new variable* $\boldsymbol{\eta} \in \mathbb{R}^d$ *with full covariance* $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$*.*

Proof. First recall that autoregressive models can represent any joint distribution as a product of conditionals: $p(\boldsymbol{\eta}; \theta) = \prod_{i=1}^{d} p(\eta_i \mid \eta_1, \ldots, \eta_{i-1}; \theta)$. Define the autoregressive transformation of $\mathbf{u}$ into $\boldsymbol{\eta}$ as follows:

$$\eta_i = \mu_i(\boldsymbol{\eta}_{1:i-1}) + \sigma_i(\boldsymbol{\eta}_{1:i-1}) u_i, \qquad \text{for} \quad i = 1, 2, \ldots, d, \tag{30}$$

where $\mu_i(\cdot)$ and $\sigma_i(\cdot)$ are functions of preceding elements $\boldsymbol{\eta}_{1:i-1}$, thereby inducing a lower triangular dependency structure in $\boldsymbol{\eta}$. Thus, there exists a lower triangular matrix $\mathbf{L} \in \mathbb{R}^{d \times d}$ such that $\mathbf{L}\mathbf{L}^\top = \boldsymbol{\Sigma}$ (via Cholesky decomposition), and the autoregressive transformation can be equivalently expressed as a set of linear equations of the form:

$$\eta_i = \sum_{j=1}^{i-1} L_{ij} u_j + L_{ii} u_i, \qquad \text{for} \quad i = 1, 2, \ldots, d, \tag{31}$$

where L_{ij} determine the linear dependencies between elements, thereby inducing a full covariance structure for $\boldsymbol{\eta}$. □

Remark B.2. Kingma et al. [40] undoubtedly recognised this fact, as it was informally mentioned in their exposition. We provide a simple proof here to make the argument in favour of our proposed approach more rigorous.

B.2 Choosing a Flow & Optimization Objective

The following provides supplementary derivations of the different optimization objectives for training discrete-time Flow-SSNs outlined in the main paper. Recall the discrete-time Flow-SSN is defined as follows:

$$p(\mathbf{y} \mid \mathbf{x}) = \int p(\mathbf{y} \mid \boldsymbol{\eta}) p(\boldsymbol{\eta} \mid \mathbf{x}; \lambda, \theta) \, \mathrm{d}\boldsymbol{\eta} \tag{32}$$

$$\text{where} \quad p(\boldsymbol{\eta} \mid \mathbf{x}; \lambda, \theta) = p_{U|X}(\mathbf{u} \mid \mathbf{x}; \lambda) \left| \det \mathbf{J}_\phi(\mathbf{u}) \right|^{-1}, \quad \text{and} \quad p(\mathbf{y} \mid \boldsymbol{\eta}) = \operatorname{Cat}(\mathbf{y}; \operatorname{softmax}_k(\boldsymbol{\eta})). \tag{33}$$

We then have to choose a flow to model $p(\boldsymbol{\eta} \mid \mathbf{x}; \lambda, \theta)$, and for this we revisit affine autoregressive flows, specifically IAFs [40] and MAFs [61] as they are both simple and flexible enough for our needs. They remain relatively underexplored to date, and in this work, we combine them with modern autoregressive Transformers to build Flow-SSNs. With that in mind, in the following subsections, we detail the various design options available for Flow-SSNs.

B.3 Expected Categorical Likelihood: Monte Carlo Estimator

The simplest approach is to use an IAF $p^{\text{IAF}}(\boldsymbol{\eta} \mid \mathbf{x}; \lambda, \theta)$, which is fast to sample form, and use a simple Monte Carlo estimator of the likelihood in Eq. (32) analogous to a standard SSN:

$$p(\mathbf{y} \mid \mathbf{x}) = \mathbb{E}_{\boldsymbol{\eta} \sim p^{\text{IAF}}(\boldsymbol{\eta} \mid \mathbf{x}; \lambda, \theta)} \left[p(\mathbf{y} \mid \boldsymbol{\eta}) \right] \tag{34}$$

$$= \mathbb{E}_{\mathbf{u} \sim p_{U|X}(\mathbf{u} \mid \mathbf{x}; \lambda)} \left[p(\mathbf{y} \mid \boldsymbol{\eta} = \phi(\mathbf{u}; \theta)) \right] \tag{35}$$

$$\approx \frac{1}{M} \sum_{i=1}^{M} p(\mathbf{y} \mid \phi(\mathbf{u}^{(i)}; \theta)), \qquad \mathbf{u}^{(i)} | \mathbf{x} \sim p_{U|X}, \tag{36}$$

where $(\mathbf{x}, \mathbf{y}) \sim p_{\text{data}}(\mathbf{x}, \mathbf{y})$ and the flow is parameterised by an autoregressive model with parameters θ. Taking logs, we get:

$$\log p(\mathbf{y} \mid \mathbf{x}) \approx \log \frac{1}{M} \sum_{i=1}^{M} p(\mathbf{y} \mid \phi(\mathbf{u}^{(i)}; \theta)) \tag{37}$$

$$= \operatorname{LSE}\left(\log p(\mathbf{y} \mid \phi(\mathbf{u}^{(1)}; \theta)) + \cdots + \log p(\mathbf{y} \mid \phi(\mathbf{u}^{(M)}; \theta)) \right) - \log M, \tag{38}$$

where $\operatorname{LSE}(\cdot)$ is the log-sum-exp function, and the individual categorical likelihood terms are given by:

$$\log p(\mathbf{y} \mid \boldsymbol{\eta} = \phi(\mathbf{u}; \theta)) = \log \operatorname{Cat}(\mathbf{y}; \operatorname{softmax}_k(\boldsymbol{\eta})) = \sum_{i=1}^{k} \sum_{j=1}^{d} y_{i,j} \log \operatorname{softmax}(\boldsymbol{\eta}_{:,j})_i. \tag{39}$$

Remark B.3. This approach is the most similar to standard SSNs as it uses the same Monte Carlo setup to integrate out the logits and compute $p(\mathbf{y} \mid \mathbf{x})$. We simply replace the low-rank Gaussian parameterisation with an autoregressive flow that is still cheap to sample from: an IAF. The approach is attractive in that the majority of the model capacity is allocated to learning the base distribution $p_{U|X}$, which only requires a single forward pass to compute. Given the parameters of $p_{U|X}$, it is cheap to sample from it and compute the logits $\boldsymbol{\eta} = \phi(\mathbf{u}; \theta)$, as the flow ϕ is lightweight, e.g. a single linear autoregressive transformation is sufficient (c.f. Proposition 5.1). In practice, it can be beneficial to make ϕ more expressive.

B.4 Dual-Flow: Evidence Lower Bound

An important fact about IAFs is that, although scoring observations is slow and unparallelizable, they can still score their *own* samples efficiently since intermediate outputs can be cached when sampling $\boldsymbol{\eta} \sim p^{\text{IAF}}$, then reused for scoring the sample. This opens up various design options for Flow-SSNs, which we explain in greater detail next.

B.4.1 Dual-Flow

We introduce a *dual-flow* setup comprised of an IAF $p^{\text{IAF}}(\boldsymbol{\eta} \mid \mathbf{x}; \lambda, \theta)$ and an MAF $p^{\text{MAF}}(\boldsymbol{\eta} \mid \mathbf{x}; \hat{\lambda}, \hat{\theta})$, both defined in logit space and trained concurrently, such that we maximize a lower bound on $\log p(\mathbf{y} \mid \mathbf{x})$:

$$\log p(\mathbf{y} \mid \mathbf{x}) = \log \int p(\mathbf{y} \mid \boldsymbol{\eta}) p^{\text{MAF}}(\boldsymbol{\eta} \mid \mathbf{x}; \hat{\lambda}, \hat{\theta}) \, \mathrm{d}\boldsymbol{\eta} \tag{40}$$

$$= \int p(\mathbf{y} \mid \boldsymbol{\eta}) p^{\text{MAF}}(\boldsymbol{\eta} \mid \mathbf{x}; \hat{\lambda}, \hat{\theta}) \frac{p^{\text{IAF}}(\boldsymbol{\eta} \mid \mathbf{x}; \lambda, \theta)}{p^{\text{IAF}}(\boldsymbol{\eta} \mid \mathbf{x}; \lambda, \theta)} \, \mathrm{d}\boldsymbol{\eta} \tag{41}$$

$$\geq \mathbb{E}_{\boldsymbol{\eta} \sim p^{\text{IAF}}(\boldsymbol{\eta} | \mathbf{x}; \lambda, \theta)} \left[\log \frac{p(\mathbf{y} \mid \boldsymbol{\eta}) p^{\text{MAF}}(\boldsymbol{\eta} \mid \mathbf{x}; \hat{\lambda}, \hat{\theta})}{p^{\text{IAF}}(\boldsymbol{\eta} \mid \mathbf{x}; \lambda, \theta)} \right] \tag{42}$$

$$= \mathbb{E}_{\boldsymbol{\eta} \sim p^{\text{IAF}}(\boldsymbol{\eta} | \mathbf{x}; \lambda, \theta)} \left[\log p(\mathbf{y} \mid \boldsymbol{\eta}) \right] - D_{\text{KL}} \left(p^{\text{IAF}} \parallel p^{\text{MAF}} \right). \tag{43}$$

where $(\mathbf{x}, \mathbf{y}) \sim p_{\text{data}}(\mathbf{x}, \mathbf{y})$. There is no general closed-form solution to this KL term, so we approximate it using samples:

$$D_{\text{KL}} \left(p^{\text{IAF}} \parallel p^{\text{MAF}} \right) \approx \frac{1}{M} \sum_{i=1}^{M} \log \frac{p^{\text{IAF}}(\boldsymbol{\eta}^{(i)} \mid \mathbf{x}; \lambda, \theta)}{p^{\text{MAF}}(\boldsymbol{\eta}^{(i)} \mid \mathbf{x}; \hat{\lambda}, \hat{\theta})}, \qquad \boldsymbol{\eta}^{(i)} | \mathbf{x} \sim p^{\text{IAF}}, \tag{44}$$

which is cheap since sampling from p^{IAF} is parallelizable and computing the base distribution $p_{U|X}$ only requires a single forward pass of $\mathbf{x}$. Furthermore, scoring p^{IAF}'s samples under p^{MAF} is also cheap since scoring in MAFs is parallelizable.

In practice, we recommend using a different (unbiased) estimator which has lower variance, proposed by Schulman [75]:

$$D_{\text{KL}} \left(p^{\text{IAF}} \parallel p^{\text{MAF}} \right) \approx \frac{1}{M} \sum_{i=1}^{M} \operatorname{expm1}(r) - r, \qquad r := \log \frac{p^{\text{IAF}}(\boldsymbol{\eta}^{(i)} \mid \mathbf{x}; \lambda, \theta)}{p^{\text{MAF}}(\boldsymbol{\eta}^{(i)} \mid \mathbf{x}; \hat{\lambda}, \hat{\theta})}, \qquad \boldsymbol{\eta}^{(i)} | \mathbf{x} \sim p^{\text{IAF}}. \tag{45}$$

Remark B.4. This combination of IAFs and MAFs is reminiscent of *probability density distillation* [60], a student-teacher distillation technique used in audio synthesis models, wherein a pre-trained MAF is fixed as a teacher, and an IAF student learns to match it. In our setup, both flow models are trained concurrently to maximise a bespoke lower bound for stochastic segmentation tasks. Furthermore, it is possible for p^{IAF} and p^{MAF} to share the base distribution parameters λ.

B.4.2 Improper Uniform Prior

Alternatively to the above, we can replace p^{MAF} with an improper uniform prior p such that:

$$p(\boldsymbol{\eta}) = \text{const}, \ \forall \boldsymbol{\eta} \implies D_{\text{KL}} \left(p^{\text{IAF}} \parallel p \right) = \mathbb{E}_{p^{\text{IAF}}(\boldsymbol{\eta} | \mathbf{x}; \lambda, \theta)} \left[\log \frac{p^{\text{IAF}}(\boldsymbol{\eta} \mid \mathbf{x}; \lambda, \theta)}{\text{const}} \right] \tag{46}$$

$$= -H(p^{\text{IAF}}) - \log \text{const}, \tag{47}$$

which, after dropping the constant term w.r.t the model parameters, yields the objective:

$$\log p(\mathbf{y} \mid \mathbf{x}) \geq \mathbb{E}_{\boldsymbol{\eta} \sim p^{\text{IAF}}(\boldsymbol{\eta} | \mathbf{x}; \lambda, \theta)} \left[\log p(\mathbf{y} \mid \boldsymbol{\eta}) \right] + \beta H(p^{\text{IAF}}), \tag{48}$$

with a weighting hyperparameter $\beta > 0$. Maximising $H(p^{\text{IAF}})$ prevents the model from collapsing to a deterministic one. The special case where β is set to 0 makes this objective equivalent to the expected categorical likelihood above.

Proposition B.5 (IAF Entropy Estimator)**.** *The entropy $H(p^{\mathrm{IAF}})$ can be efficiently estimated in parallel via Monte Carlo sampling $\mathbf{u}^{(i)}|\mathbf{x} \sim p_{U|X}$ using the following formula:*

$$H(p^{\mathrm{IAF}}) \approx H(p_{U|X}) - \frac{1}{M}\sum_{i=1}^{M} \log \left|\det \mathbf{J}_{\phi^{-1}}(\phi(\mathbf{u}^{(i)};\theta))\right|, \qquad \mathbf{u}^{(i)}|\mathbf{x} \sim p_{U|X}. \tag{49}$$

Proof. We simply start with the definition of differential entropy, then use a change-of-variables and simplify:

$$\begin{aligned}
H(p^{\mathrm{IAF}}) &= -\mathbb{E}_{\boldsymbol{\eta}\sim p^{\mathrm{IAF}}(\boldsymbol{\eta}|\mathbf{x};\lambda,\theta)}\left[\log p^{\mathrm{IAF}}(\boldsymbol{\eta} \mid \mathbf{x};\lambda,\theta)\right] && (50)\\
&= -\int p^{\mathrm{IAF}}(\boldsymbol{\eta} \mid \mathbf{x};\lambda,\theta)\Big[\log p_{U|X}(\phi^{-1}(\boldsymbol{\eta};\theta) \mid \mathbf{x};\lambda) + \log\left|\det \mathbf{J}_{\phi^{-1}}(\boldsymbol{\eta})\right|\Big]\,\mathrm{d}\boldsymbol{\eta} && (51)\\
&= -\int p_{U|X}(\mathbf{u} \mid \mathbf{x};\lambda)\Big[\log p_{U|X}(\mathbf{u} \mid \mathbf{x};\lambda) + \log\left|\det \mathbf{J}_{\phi^{-1}}(\phi(\mathbf{u};\theta))\right|\Big]\,\mathrm{d}\mathbf{u} && (52)\\
&= H(p_{U|X}) - \mathbb{E}_{\mathbf{u}\sim p_{U|X}(\mathbf{u}|\mathbf{x};\lambda)}\left[\log\left|\det \mathbf{J}_{\phi^{-1}}(\phi(\mathbf{u};\theta))\right|\right], && (53)\\
&\approx H(p_{U|X}) - \frac{1}{M}\sum_{i=1}^{M} \log\left|\det \mathbf{J}_{\phi^{-1}}(\phi(\mathbf{u}^{(i)};\theta))\right|, \qquad \mathbf{u}^{(i)}|\mathbf{x} \sim p_{U|X}, && (54)
\end{aligned}$$

which is what we wanted to show. □

Remark B.6. The entropy $H(p_{U|X})$ is available in closed-form for typical base distributions (e.g. Gaussian). Computing $p_{U|X}(\mathbf{u} \mid \mathbf{x};\lambda)$ only requires a single forward pass, which is important as it comprises the majority of the model parameters. The flow component $\phi(\cdot)$ is lightweight, since a single (linear) layer is sufficient (c.f. Proposition 5.1). Lastly, recall that autoregressive flows admit a lower triangular Jacobian by design, and as such, their log absolute determinant simplifies to a sum of the diagonal elements, which is easy to compute.

B.5 Logit Bijections

For bijective logit mappings $\mathbf{y} = g(\boldsymbol{\eta})$ with tractable Jacobians determinants, the likelihood term $p(\mathbf{y} \mid \mathbf{x})$ in Flow-SSN models can be evaluated directly using the following change-of-variables formula:

$$p(\mathbf{y} \mid \mathbf{x}) = p_{U|X}((\phi \circ g)^{-1}(\mathbf{y}) \mid \mathbf{x};\lambda)\left|\det \mathbf{J}_{\phi^{-1}}(g^{-1}(\mathbf{y}))\right|\left|\det \mathbf{J}_{g^{-1}}(\mathbf{y})\right|.$$

However, the above does not hold when $g(\cdot)$ is the softmax function, as it is not bijective. TensorFlow Probability [1] provides the somewhat underused function `tfp.bijectors.SoftmaxCentered`, which is an alternative bijective softmax transformation that we can use here, albeit at the cost of having to dequantise $\mathbf{y}$. We must also train with $k+1$ classes, where a newly introduced dummy class acts as a pivot and facilitates the bijective property of the function. Relatedly, De Sousa Ribeiro et al. [17], Monteiro et al. [56] have also recently used this transformation to train discrete causal mechanisms.

C Evaluation Metrics

Throughout this work, we use the following evaluation metrics standard for stochastic segmentation tasks.

Dice Similarity Coefficient. For two label maps $\mathbf{y}, \hat{\mathbf{y}}$ of dimension $h \times w$, the Dice Similarity Coefficient (DSC) is defined as follows:

$$\mathrm{DSC}(\mathbf{y},\hat{\mathbf{y}}) = \frac{2|\mathbf{y} \cap \hat{\mathbf{y}}|}{|\mathbf{y}| + |\hat{\mathbf{y}}|}, \quad \text{where} \quad |\mathbf{y} \cap \hat{\mathbf{y}}| = \sum_{i=1}^{h}\sum_{j=1}^{w} y_{i,j} \times \hat{y}_{i,j}, \quad \text{and} \quad |\mathbf{y}| = \sum_{i=1}^{h}\sum_{j=1}^{w} y_{i,j}. \tag{55}$$

Intersection over Union. Intersection over Union (IoU) is a similar widely used segmentation metric defined as follows:

$$\mathrm{IoU}(\mathbf{y},\hat{\mathbf{y}}) = \frac{|\mathbf{y} \cap \hat{\mathbf{y}}|}{|\mathbf{y} \cup \hat{\mathbf{y}}|}, \quad \text{where} \quad |\mathbf{y} \cup \hat{\mathbf{y}}| = |\mathbf{y}| + |\hat{\mathbf{y}}| - |\mathbf{y} \cap \hat{\mathbf{y}}|. \tag{56}$$

Compared with DSC, IoU is generally a less forgiving metric since it does not weight the overlap as strongly, thereby penalising mismatches more strongly.

Generalised Energy Distance. We use Generalised Energy Distance (GED) to compare the quality of samples from the model with the ground truth labels. Independent samples $\hat{\mathbf{y}}, \hat{\mathbf{y}}' \overset{\text{iid}}{\sim} p_{\text{model}}$ are drawn from the predictive model distribution p_{model} whereas the multiple ground truth annotations $\mathbf{y}, \mathbf{y}' \sim p_{\text{data}}$ are from the data distribution p_{data}, then:

$$D^2_{\text{GED}}(p_{\text{data}}, p_{\text{model}}) = 2\,\mathbb{E}_{\mathbf{y}\sim p_{\text{data}}, \hat{\mathbf{y}}\sim p_{\text{model}}}[d(\mathbf{y}, \hat{\mathbf{y}})] - \mathbb{E}_{\hat{\mathbf{y}}, \hat{\mathbf{y}}'\sim p_{\text{model}}}[d(\hat{\mathbf{y}}, \hat{\mathbf{y}}')] - \mathbb{E}_{\mathbf{y}, \mathbf{y}'\sim p_{\text{data}}}[d(\mathbf{y}, \mathbf{y}')]. \tag{57}$$

We follow Kohl et al. [43], Monteiro et al. [55] by using the distance function $d(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \text{IoU}(\mathbf{y}, \hat{\mathbf{y}})$, which is a metric, and implies D^2_{GED} is also a metric. Lower GED indicates better alignment between the predictive and ground truth distributions. The sample diversity $\mathbb{E}_{\hat{\mathbf{y}}, \hat{\mathbf{y}}'\sim p_{\text{model}}}[d(\hat{\mathbf{y}}, \hat{\mathbf{y}}')]$ is also a quantity of interest, as it measures the average distance between pairs of samples from the predictive distribution, and provides a measure of variability in our samples. We note that diversity is only contextually relevant, as high diversity alone can be trivially achieved with random noise as a model.

Hungarian-matched IoU. We also use Hungarian-matched IoU (HM-IoU) to compare samples between the ground truth and predictive distributions. HM-IoU uses the Hungarian algorithm to find an optimal assignment between the two sets of samples: $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\} \sim p_{\text{data}}$ and $\{\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots, \hat{\mathbf{y}}_M\} \overset{\text{iid}}{\sim} p_{\text{model}}$, then using $1 - \text{IoU}$ as the cost matrix for solving a linear sum assignment problem [44]. HM-IoU can be viewed as more robust across sets of ground truth and predictive samples compared to the GED, which can suffer from inflated scores if the predictive samples are very diverse.

D Implementation Details

D.1 Training Setup & Hyperparameters

In this section, we provide the setup and hyperparameters we used to train our Flow-SSN models on the medical datasets. Our implementation is based in `PyTorch` [64]. As shown in Table 3, we use a UNet [19, 72] to parameterise the base distribution of our Flow-SSN (i.e. Prior Network). To parameterise the flow transformation itself (i.e. Flow Network), we use an autoregressive Transformer for the discrete-time Flow-SSN variant and a UNet for the continuous-time variant. For data augmentation on LIDC-IDRI, we used random 90 degree rotations and vertical/horizontal flips with 0.5 probability. For data augmentation on REFUGE-MultiRater, we used random vertical flips with 0.5 probability; random rotations in the range of [-20, 20] degrees, random resized cropping to 256x256 with scale [0.9, 1.1], and applied color jitter of 0.3 to the images.

The remaining hyperparameters we used are largely equal for both datasets, as reasonable initial values performed well enough; we did not perform extensive hyperparameter tuning for each dataset. For the continuous-time Flow-SSN, 8 ODE solving steps (Euler method in `torchdiffeq` [13]) were used on the validation set throughout training for model selection. In all cases, the best checkpoint was selected based on the lowest GED achieved on the validation set during training. The final model artefact we use for evaluation is an exponential moving average (EMA) of the model parameters.

Table 3. Training hyperparameters used across all experiments.

	LIDC-IDRI		**REFUGE-MultiRater**	
CONFIG	Flow-SSN$_\Delta$	Flow-SSN$_\infty$	Flow-SSN$_\Delta$	Flow-SSN$_\infty$
Prior Network	UNet	UNet	UNet	UNet
Flow Network	Transformer	UNet	Transformer	UNet
Optimiser	AdamW	AdamW	AdamW	AdamW
Batch Size	16	16	16	16
Learning Rate	10^{-4}	10^{-4}	10^{-4}	10^{-4}
LR Warmup	Linear 2K	Linear 2K	Linear 1K	Linear 1K
Weight Decay	10^{-4}	10^{-4}	10^{-4}	10^{-4}
EMA Rate	0.9999	0.9999	0.999	0.999
Max Epochs	1001	1001	1001	1001
Eval Freq.	16	16	50	50
MC Samples (train)	16	1	32	1
MC Samples (eval)	16	16	16	16
Prior Dist.	Gaussian	Gaussian	Gaussian	Gaussian

D.2 UNet Architecture

As mentioned in the main text, we reimplemented a streamlined version of Dhariwal and Nichol [19]'s UNet, which uses fewer attention layers. Recall that LIDC-IDRI images are grayscale, whereas REFUGE images are RGB. To parameterise a Flow-SSN prior, the UNet outputs a mean and variance per output pixel, representing the 'initial guess' distribution as a diagonal Gaussian, to be later refined by the choice of flow. The prior's mean can be optionally initialised with a pre-trained network and then fine-tuned alongside the flow network. In our case, we train everything end-to-end for simplicity and find that fixing the prior variance to, e.g. 1, can also help stabilise training in some instances. Table 4 shows the remaining details, including input and output shapes for both the prior and flow networks.

Table 4. UNet hyperparameters used for parameterising both the base distribution (Prior) and the flows in our Flow-SSN models. We use (Δ) and (∞) to denote relation to the discrete and continuous-time version of the associated Flow-SSN.

	LIDC-IDRI		**REFUGE-MultiRater**	
CONFIG	Prior (Δ, ∞)	Flow (∞)	Prior (Δ, ∞)	Flow (∞)
Input Shape	(1, 128, 128)	(2, 128, 128)	(3, 256, 256)	(2, 256, 256)
Model Channels	32	16	32	32
Output Channels	4	2	4	2
Residual Blocks	1	1	2	1
Dropout	0.1	0.1	0.1	0.1
Channel Multipliers	[1, 2, 4, 8]	[1, 1, 1]	[1, 2, 2, 4, 6]	[1, 1, 1, 1]
Attention Resolution	[16]	[16]	[16]	[16]
Num. Heads	1	1	1	1
Head Channels	64	16	64	32
#Parameters	14.4M	150K	14.6M	787K

D.3 Autoregressive Transformer Architecture

To parameterise the discrete-time autoregressive Flow-SSN, we require a lightweight autoregressive model. The challenge is that our flow lives in pixel-space, so autoregressively predicting each pixel can become computationally expensive for large datasets. To overcome this obstacle, we propose two things. The first is that we use an IAF [40] defined in *in pixel-space* and train under the expected likelihood objective in Eq. (14), which avoids pixel-wise sequential likelihood evaluation and enables fast, one-pass sampling at inference time. Recall that MAFs [61] are equally fast to train as likelihood evaluation can be parallelised, but they still require sequential autoregressive sampling at inference time. The second is that we use an autoregressive Transformer to parameterise the flow, with image strips/patches of, e.g., size (1,8) or (8,8) pixels to reduce memory requirements. To reconstruct the patches back to the output size, we simply use a transposed convolution. Grouping pixels this way induces a block covariance structure in pixel space, but as long as the strips/patches are small relative to the full image size, the maximum attainable covariance rank remains high (i.e. in the hundreds/thousands for real images). Furthermore, since Flow-SSNs model pixel covariance in logit space and transform the learned distribution by a non-linear transfer function (softmax), there is a sublinear increase in rank we can benefit from (c.f. theoretical results in Appendix A).

In summary, and perhaps surprisingly, we find that the above combination of design choices works well provided we use enough MC samples during training, e.g. ≥ 8, otherwise, the model can collapse to a near-deterministic state (c.f. Fig. 11). It is worth noting that we briefly experimented with a shallow PixelCNN [74, 88, 88] to parameterise the flow instead of an autoregressive Transformer, but did not have as much success. As shown in Table 5, the final architecture setup for a discrete-time autoregressive Flow-SSN uses just 1 autoregressive flow transformation parameterised by a single Transformer layer! As proven in Appendix A, a single autoregressive transformation is sufficient to transform the diagonal Gaussian prior into a highly expressive full covariance. We confirm this works well in practice on both toy and real medical imaging datasets.

Table 5. Autoregressive Transformer hyperparameters used for our discrete-time autoregressive Flow-SSNs.

CONFIG	**LIDC-IDRI** Flow (Δ)	**REFUGE-MultiRater** Flow (Δ)
Input Shape	(2, 128, 128)	(2, 256, 256)
Num. Flows	1	1
Flow Type	IAF	IAF
Output Channels	4	4
Embed Dim	64	128
MLP Width	256	512
Patch Size	(1, 8)	(8, 8)
Patchify	Conv	Conv
Unpatchify	ConvTranspose	ConvTranspose
Num. Blocks	1	2
Num. Heads	1	1
Pos Embed Init	$\mathcal{N}(0, 0.02)$	$\mathcal{N}(0, 0.02)$
Dropout	0.1	0.1
Activation	GELU	GELU
#Parameters	345K	0.92M

E Extra Results

E.1 Sampling Efficiency & Additional Baselines

As reported in Table 6, our method is $\approx 10\times$ more efficient than CCDM [92], thanks to most of the model parameters being in the flow's *prior* (i.e. base/source distribution), which only requires a single forward pass to start the sampling chain. Afterwards, only the *flow* network is needed to solve the ODE, and it has only 150K parameters[5]. The advantage of our model translates to any other diffusion-based segmentation model that dedicates all its model capacity to learning the score/velocity field [3, 66, 89, 92]. That said, we expect that using large models for both the prior *and* the flow would improve performance even further; though we argue that the latter may not be necessary if the prior is expressive enough, e.g. a foundation model. Table 7 reports further baseline comparisons against recent SOTA methods; Flow-SSN outperforms all previous methods.

Table 6. Comparing Flow-SSN sampling efficiency against CCDM (see Figure 5 in [92]).

	CCDM [92]			Flow-SSN		
Steps	$D^2_{\text{GED}}(16) \downarrow$	HM-IoU $\uparrow$	FLOPS $\downarrow$	$D^2_{\text{GED}}(16) \downarrow$	HM-IoU $\uparrow$	FLOPS $\downarrow$
1	-	-	-	$0.240_{\pm.002}$	$0.879_{\pm.000}$	12G
50	$\simeq 0.34_{\pm\text{N/A}}$	$\simeq 0.55_{\pm\text{N/A}}$	365G	$0.210_{\pm.002}$	$0.873_{\pm.000}$	**51G**
250	$0.212_{\pm.002}$	$0.623_{\pm.002}$	1.83T	$0.207_{\pm.000}$	$0.873_{\pm.001}$	**207G**

Table 7. Additional comparisons of Flow-SSN against recent SOTA methods on LIDC-IDRI.

METHOD	Pub. Venue	$D^2_{\text{GED}}(16) \downarrow$	HM-IoU $\uparrow$	#Param
Tyche [67]	CVPR'24	$0.400_{\pm.010}$	-	1.7M
CCDM [92]	ICCV'23	$0.212_{\pm.002}$	$0.623_{\pm.002}$	9M
MoSE [25]	ICLR'23	$0.218_{\pm.003}$	$0.624_{\pm.004}$	42M
CIMD [66]	CVPR'23	$0.234_{\pm.005}$	$0.587_{\pm.001}$	24M
Flow-SSN$_\infty$	ICCV'25	$\mathbf{0.207}_{\pm.000}$	$\mathbf{0.873}_{\pm.001}$	14M

[5] For 1-step we use a discrete-time inverse autoregressive Flow-SSN

E.2 Ablation Study: Increasing the Assumed Rank

We first assess the scalability of standard SSNs by ablating an increase in the assumed rank using a replication of Monteiro et al. [55]'s setup. We observe (Figure 10) that SSNs tend to collapse to deterministic models even under mild increases in the assumed rank, which shows that a different approach is needed to capture higher-order interactions between pixels.

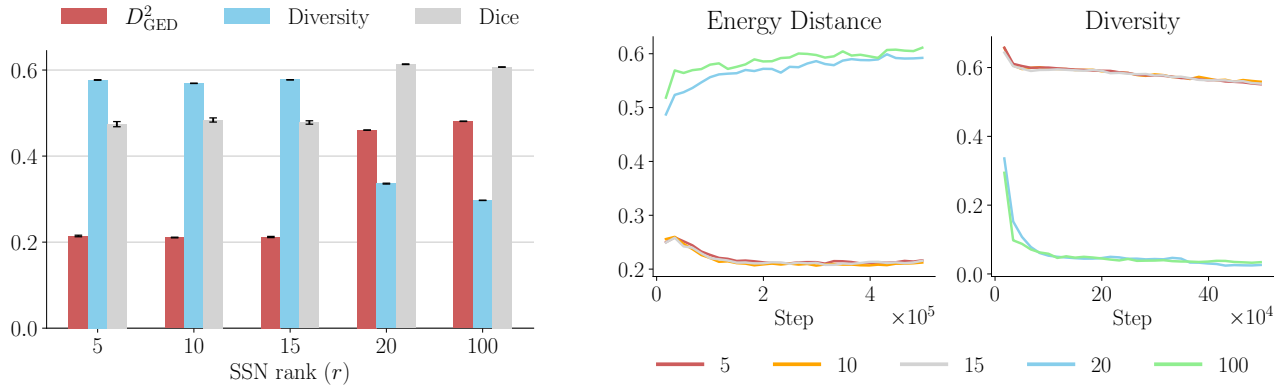

Figure 10. **Scalability ablation study of SSNs on LIDC in terms of the assumed rank**. The results show that mild increases in the assumed rank can cause SSNs to collapse into a near-deterministic state. This warrants a new approach for estimating high-rank covariances.

E.3 Ablation Study: Number of Monte Carlo Samples

As shown in Figure 11, we find that multiple MC samples are needed to properly learn the underlying distribution over outputs when using a discrete-time inverse autoregressive Flow-SSN. With only 1 MC sample, the model enters the near-deterministic regime, and increasing the number of MC samples provides diminishing returns in terms of improved performance. We consider further improving the training stability of IAFs at scale, possibly borrowing tricks from modern MAFs and coupling flows [28, 83, 94], to be fertile ground for future work. The payoff in efficiency is significant as sampling becomes parallelizable.

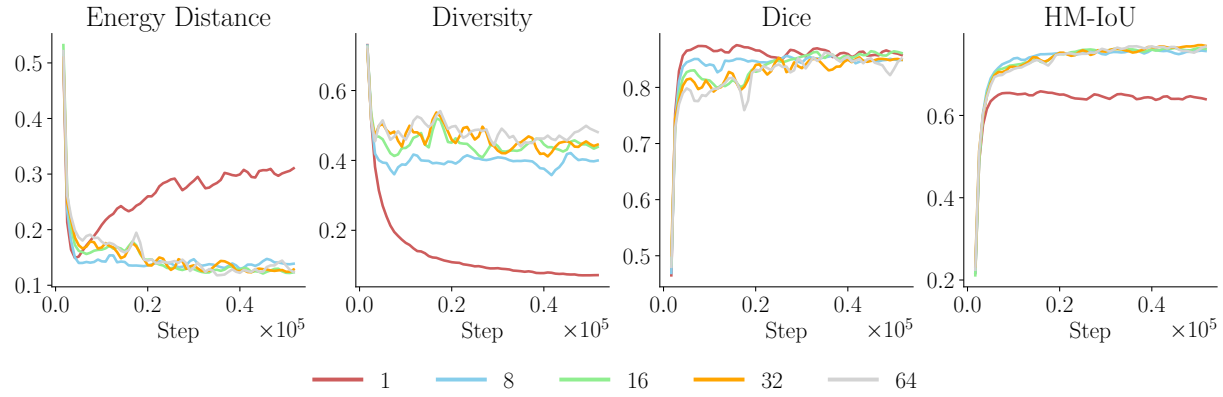

Figure 11. **Ablation analysis of the number of Monte Carlo (MC) samples needed for training**. We look at $\{1, 8, 16, 32, 64\}$ used for training a discrete-time autoregressive Flow-SSN with the objective in Equation (14). The above results were obtained using the REFUGE-MultiRater dataset, and performance on the validation set is shown throughout training.

E.4 Qualitative Results: LIDC-IDRI

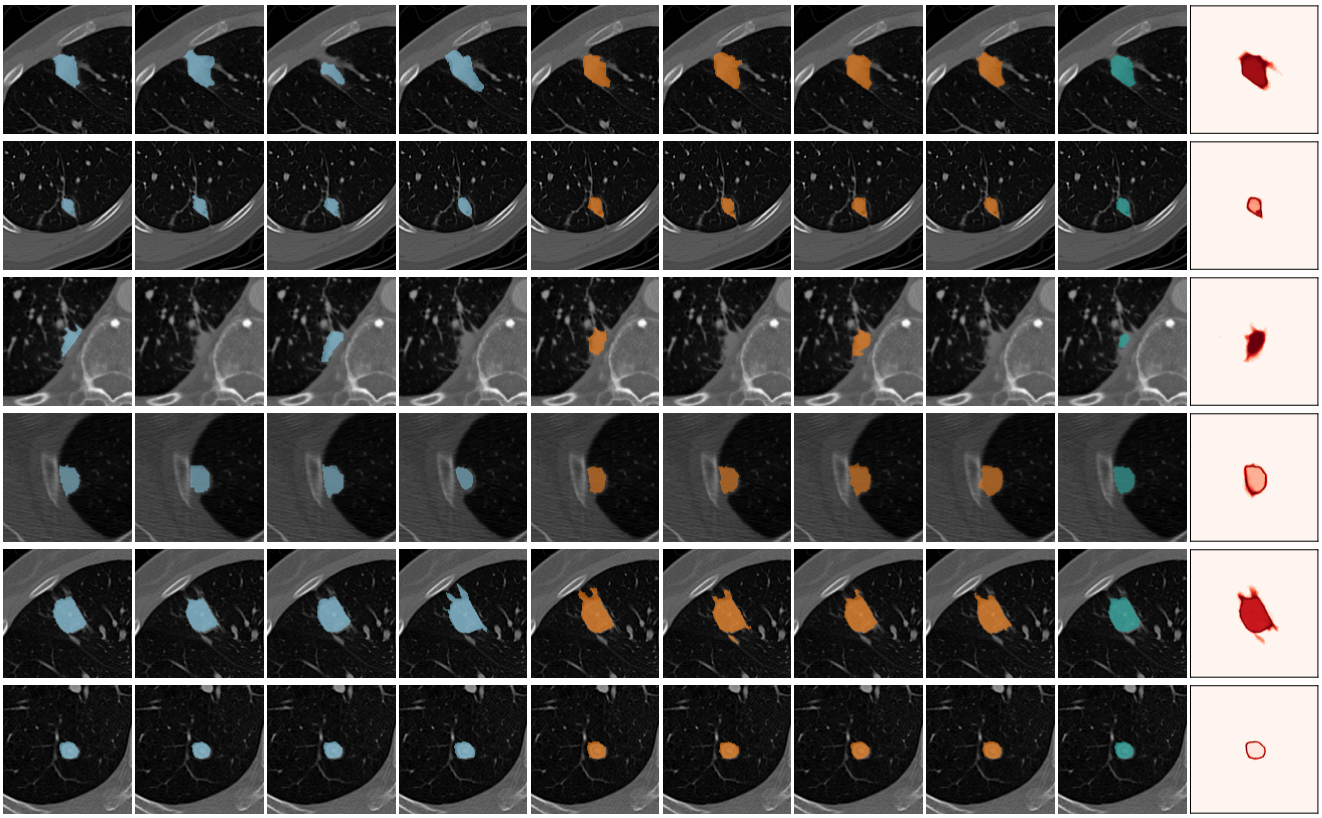

Figure 12. **Extra qualitative results on LIDC-IDRI using our continuous-time Flow-SSN model**. (*Cols. 1-4*) Multiple ground truth segmentations from experts; (*Cols. 5-8*) Non-cherry-picked random samples from our model; (*Cols. 9, 10*) The mean prediction and per-pixel uncertainty map. In all cases, 100 MC samples and 50 ODE solving steps (Euler method) were used for evaluation.

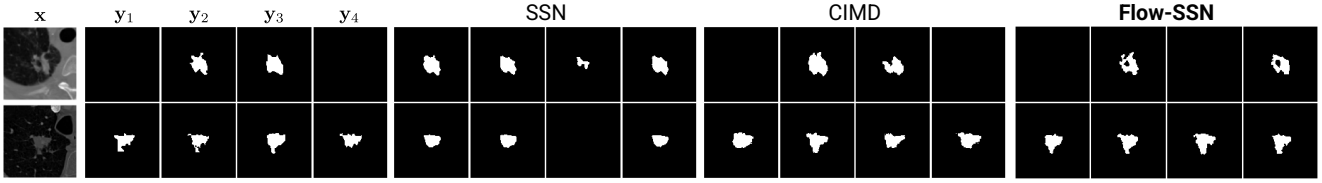

Figure 13. **Qualitative comparison of Flow-SSN against previous methods on LIDC-IDRI**. We used the same images as CIMD [66] for fair comparisons. (*Col. 1*) Input images for segmentation; (*Cols. 2-5*) Ground truth annotations from multiple experts (i.e. four total in this case $\{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4\}$); (*Cols. 6-9*) Random samples from our reproduced SSN [55] model; (*Cols. 10-13*) Random samples taken from CIMD [66], a diffusion-based segmentation model; (*Cols. 14-17*) Random samples from our (continuous-time) Flow-SSN model.

E.5 Qualitative Results: REFUGE-MultiRater

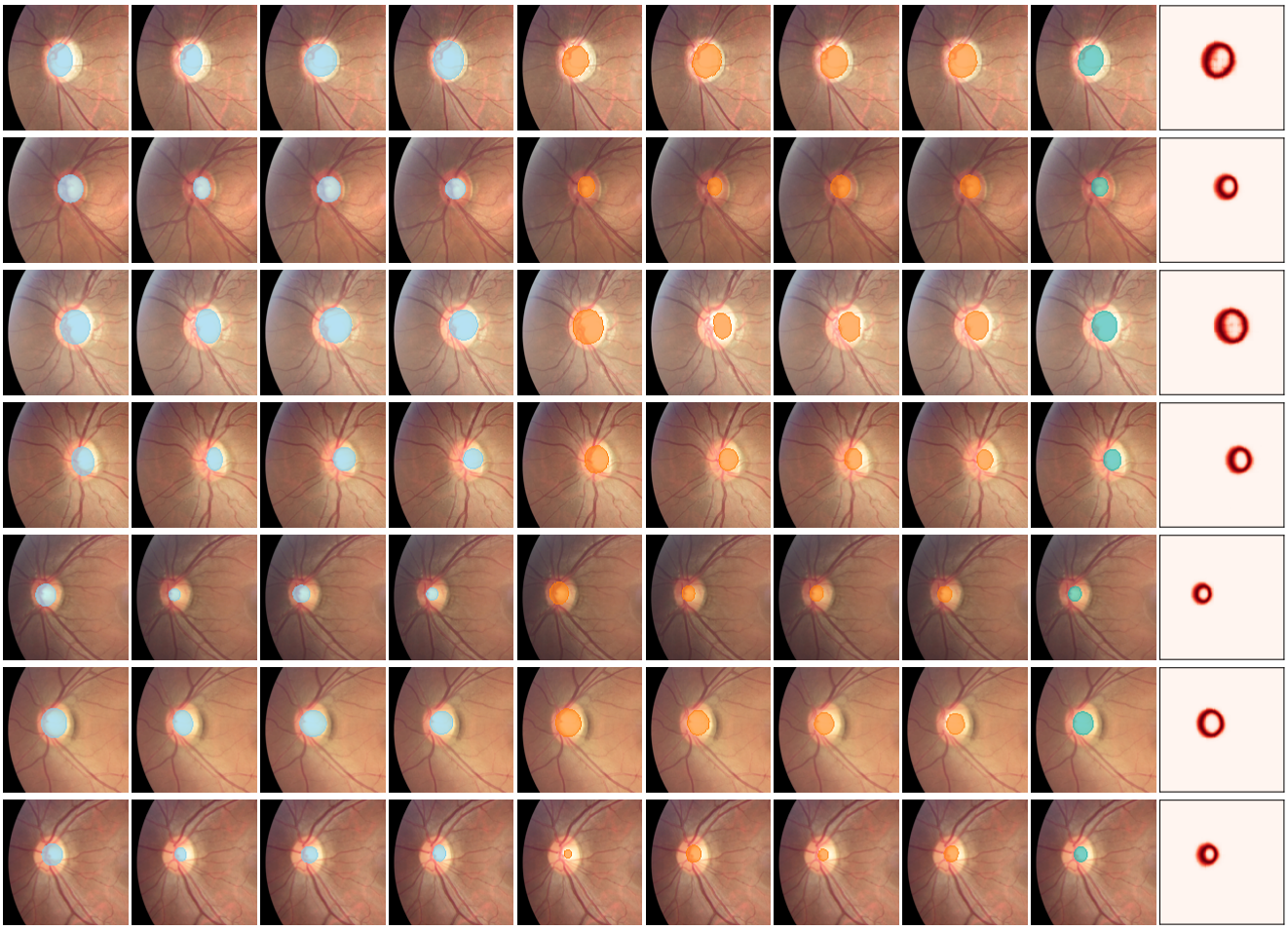

Figure 14. **Extra qualitative results on REFUGE-MultiRater using our discrete-time autoregressive Flow-SSN model**. (*Cols. 1-4*) Multiple ground truth segmentations from experts; in each case, four segmentations were randomly chosen out of the seven available in total; (*Cols. 5-8*) Non-cherry-picked random samples from our discrete-time autoregressive Flow-SSN model; (*Cols. 9, 10*) The mean prediction and per-pixel uncertainty map. In all cases, 512 Monte Carlo samples were used for performing the evaluation.