Outline

1. What are probabilistic circuits?
   tractable deep generative models

2. What are they useful for?
   controlling generative AI

3. What is the underlying theory?
   probability generating polynomials
Outline

1. What are probabilistic circuits?
   - *tractable deep generative models*

2. What are they useful for?
   - *controlling generative AI*

3. What is the underlying theory?
   - *probability generating polynomials*
**Why probabilistic inference?**

$q_1$: What is the probability that today is a Monday and there is a traffic jam on Westwood Blvd.?

$X = \{\text{Day}, \text{Time}, \text{Jam}_{\text{Str1}}, \text{Jam}_{\text{Str2}}, \ldots, \text{Jam}_{\text{StrN}}\}$

$q_1(m) = p_m(\text{Day} = \text{Mon}, \text{Jam}_{\text{Wwood}} = 1)$

$\implies$ **marginals**
Tractable Probabilistic Inference

A class of queries $Q$ is tractable on a family of probabilistic models $M$ iff for any query $q \in Q$ and model $m \in M$ exactly computing $q(m)$ runs in time $O(poly(|m|))$.

$\Rightarrow \text{ often poly will in fact be linear!}$
q3: What is the probability that today is a Monday at 12.00 and there is a traffic jam only on Westwood Blvd.?

\[ X = \{ \text{Day, Time, Jam}_{\text{wood}}, \text{Jam}_{\text{Str2}}, \ldots, \text{Jam}_{\text{StrN}} \} \]

\[ q_3(m) = p_m(X = \{ \text{Mon, 12.00, 1, 0, \ldots, 0} \}) \]

...fundamental in maximum likelihood learning

\[ \theta_m^{\text{MLE}} = \arg\max_\theta \prod_{x \in D} p_m(x; \theta) \]
Variational Autoencoders

\[
\log p_\theta(x) \geq \mathbb{E}_{z \sim q_\phi(z|x)} \left[ \log p_\theta(x \mid z) \right] - \text{KL}(q_\phi(z \mid x) \mid \mid p(z))
\]

- An explicit likelihood model!
- But computing \( \log p_\theta(x) \) is intractable
  \[ \Rightarrow \text{an infinite and uncountable mixture} \]
  \[ \Rightarrow \text{no tractable EVI} \]
- We need to optimize the ELBO...
  \[ \Rightarrow \text{which is “tricky”} \]
Normalized flows

\[ p_x(x) = p_z(f^{-1}(x)) \left| \text{det} \left( \frac{\delta f^{-1}}{\delta x} \right) \right| \]

- an explicit likelihood!
- tractable EVI queries!
- many neural variants
  - RealNVP (Dinh et al. 2016),
  - MAF (Papamakarios et al. 2017)
  - MADE (Germain et al. 2015),
  - PixelRNN (Oord et al. 2016)
Marginal queries (MAR)

$q_1$: What is the probability that today is a Monday at 12:00 and there is a traffic jam only on Westwood Blvd.?

$q_1(m) = p_m(\text{Day} = \text{Mon}, \text{Jam}_{\text{Westwood}} = 1)$

General: $p_m(e) = \int p_m(e, H) \, dH$

where $E \subset X$, $H = X \setminus E$
Normalizing flows

\[ p_X(x) = p_Z(f^{-1}(x)) \left| \det \left( \frac{\delta f^{-1}}{\delta x} \right) \right| \]

- an explicit likelihood! \implies \textit{tractable EVI queries!}

- \textbf{MAR is generally intractable:}
  - we cannot easily integrate over high-dimensional \( f \)
Probabilistic circuits

*computational graphs* that recursively define distributions
Probabilistic circuits

computational graphs that recursively define distributions

\[ p(X_1) = w_1 p_1(X_1) + w_2 p_2(X_1) \]

⇒ mixtures

\[ p(X) = p(Z = 1) \cdot p_1(X|Z = 1) + p(Z = 2) \cdot p_2(X|Z = 2) \]
Probabilistic circuits

*computational graphs* that recursively define distributions

\[
p(X_1) = w_1 p_1(X_1) + w_2 p_2(X_1)
\]

⇒ *mixtures*

\[
p(X_1, X_2) = p(X_1) \cdot p(X_2)
\]

⇒ *factorizations*
a unifying framework for tractable models
Likelihood \[ p(X_1 = -1.85, X_2 = 0.5, X_3 = -1.3, X_4 = 0.2) \]
Likelihood \[ p(X_1 = -1.85, X_2 = 0.5, X_3 = -1.3, X_4 = 0.2) \]
Likelihood \[ p(X_1 = -1.85, X_2 = 0.5, X_3 = -1.3, X_4 = 0.2) \]
A sum node is *smooth* if its children depend on the same set of variables.

A product node is *decomposable* if its children depend on disjoint sets of variables.

*smooth circuit*  

*decomposable circuit*
Smoothness + decomposability = tractable MAR

If \( p(x) = \sum_i w_i p_i(x) \), (smoothness):

\[
\int p(x) dx = \int \sum_i w_i p_i(x) dx = \\
\sum_i w_i \int p_i(x) dx
\]

\[\Rightarrow \text{integrals are “pushed down” to children}\]

[Darwiche & Marquis JAIR 2001, Poon & Domingos UAI11]
Smoothness + decomposability = tractable MAR

If \( p(x, y, z) = p(x)p(y)p(z) \), (decomposability):

\[
\int \int \int p(x, y, z) \, dx \, dy \, dz = \\
= \int \int \int p(x)p(y)p(z) \, dx \, dy \, dz = \\
= \int p(x) \, dx \int p(y) \, dy \int p(z) \, dz
\]

\( \Rightarrow \) integrals decompose into easier ones
**Smoothness** + **decomposability** = **tractable MAR**

Forward pass evaluation for MAR

\[ \Rightarrow \text{linear in circuit size!} \]

E.g. to compute \( p(x_2, x_4) \):

- leafs over \( X_1 \) and \( X_3 \) output \( Z_i = \int p(x_i) \, dx_i \)
  \[ \Rightarrow \text{for normalized leaf distributions: 1.0} \]
- leafs over \( X_2 \) and \( X_4 \) output **EVI**
- feedforward evaluation (bottom-up)
Outline

1. What are probabilistic circuits?
   *tractable deep generative models*

2. What are they useful for?
   *controlling generative AI*

3. What is the underlying theory?
   *probability generating polynomials*
Cute, but these models cannot compete?

<table>
<thead>
<tr>
<th>bpd</th>
<th>2008-2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tabular</td>
<td>😞</td>
</tr>
<tr>
<td>MNIST</td>
<td>😳</td>
</tr>
<tr>
<td>F-MNIST</td>
<td>😳</td>
</tr>
<tr>
<td>EMNIST-L</td>
<td>😳</td>
</tr>
<tr>
<td>CIFAR</td>
<td>😳</td>
</tr>
<tr>
<td>Imagenet32</td>
<td>😳</td>
</tr>
<tr>
<td>Imagenet64</td>
<td>😳</td>
</tr>
</tbody>
</table>
Cute, but these models cannot compete?

<table>
<thead>
<tr>
<th>bpd</th>
<th>2008-2020</th>
<th>2020-2021</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tabular</td>
<td>😞</td>
<td>😞</td>
</tr>
<tr>
<td>MNIST</td>
<td>😨</td>
<td>😨 &gt; 1.67</td>
</tr>
<tr>
<td>F-MNIST</td>
<td>😨</td>
<td>😨 &gt; 4.29</td>
</tr>
<tr>
<td>EMNIST-L</td>
<td>😨</td>
<td>😨 &gt; 2.73</td>
</tr>
<tr>
<td>CIFAR</td>
<td>😨</td>
<td>😨</td>
</tr>
<tr>
<td>Imagenet32</td>
<td>😨</td>
<td>😨</td>
</tr>
<tr>
<td>Imagenet64</td>
<td>😨</td>
<td>😨</td>
</tr>
</tbody>
</table>

General-purpose architecture
Cute, but these models cannot compete?

<table>
<thead>
<tr>
<th>bpd</th>
<th>2008-2020</th>
<th>2020-2021</th>
<th>ICLR 22</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tabular</td>
<td>😞</td>
<td>😊</td>
<td>🍰</td>
</tr>
<tr>
<td>MNIST</td>
<td>😱 &gt; 1.67</td>
<td>1.20</td>
<td></td>
</tr>
<tr>
<td>F-MNIST</td>
<td>😱 &gt; 4.29</td>
<td>3.34</td>
<td></td>
</tr>
<tr>
<td>EMNIST-L</td>
<td>😱 &gt; 2.73</td>
<td>1.80</td>
<td></td>
</tr>
<tr>
<td>CIFAR</td>
<td>😱</td>
<td>🍰 &gt; 5.50</td>
<td></td>
</tr>
<tr>
<td>Imagenet32</td>
<td>😱</td>
<td>🍰</td>
<td></td>
</tr>
<tr>
<td>Imagenet64</td>
<td>😱</td>
<td>🍰</td>
<td></td>
</tr>
</tbody>
</table>

General-purpose architecture

Custom GPU kernels
**Cute, but these models cannot compete?**

<table>
<thead>
<tr>
<th>bpd</th>
<th>2008-2020</th>
<th>2020-2021</th>
<th>ICLR 22</th>
<th>NeurIPS 22</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tabular</td>
<td>😞</td>
<td>😞</td>
<td>🍰</td>
<td>🍰</td>
</tr>
<tr>
<td>MNIST</td>
<td>😞</td>
<td>😞</td>
<td>&gt; 1.67</td>
<td>1.20</td>
</tr>
<tr>
<td>F-MNIST</td>
<td>😞</td>
<td>😞</td>
<td>&gt; 4.29</td>
<td>3.34</td>
</tr>
<tr>
<td>EMNIST-L</td>
<td>😞</td>
<td>😞</td>
<td>&gt; 2.73</td>
<td>1.80</td>
</tr>
<tr>
<td>CIFAR</td>
<td>😞</td>
<td>😞</td>
<td>🍰</td>
<td>🍰</td>
</tr>
<tr>
<td>Imagenet32</td>
<td>😞</td>
<td>😞</td>
<td>🍰</td>
<td>🍰</td>
</tr>
<tr>
<td>Imagenet64</td>
<td>😞</td>
<td>😞</td>
<td>🍰</td>
<td>🍰</td>
</tr>
</tbody>
</table>

General-purpose architecture

Custom GPU kernels

Pruning without losing likelihood
Cute, but these models cannot compete?

<table>
<thead>
<tr>
<th></th>
<th>bpd 2008-2020</th>
<th>bpd 2020-2021</th>
<th>ICLR 22</th>
<th>NeurIPS 22</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tabular</td>
<td>😞</td>
<td>😞</td>
<td>🍪</td>
<td>🍪</td>
</tr>
<tr>
<td>MNIST</td>
<td>😱 &gt; 1.67</td>
<td>1.20</td>
<td>1.14</td>
<td></td>
</tr>
<tr>
<td>F-MNIST</td>
<td>😱 &gt; 4.29</td>
<td>3.34</td>
<td>3.27</td>
<td></td>
</tr>
<tr>
<td>EMNIST-L</td>
<td>😱 &gt; 2.73</td>
<td>1.80</td>
<td>1.58</td>
<td></td>
</tr>
<tr>
<td>CIFAR</td>
<td>😱 &gt; 5.50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imagenet32</td>
<td>😱</td>
<td>😱</td>
<td>🍪</td>
<td></td>
</tr>
<tr>
<td>Imagenet64</td>
<td>😱</td>
<td>😱</td>
<td>🍪</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Discrete Flow</th>
<th>Hierarchical VAE</th>
<th>PixelVAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>1.90</td>
<td>1.27</td>
<td>1.39</td>
</tr>
<tr>
<td>F-MNIST</td>
<td>3.47</td>
<td>3.28</td>
<td>3.66</td>
</tr>
<tr>
<td>EMNIST-L</td>
<td>1.95</td>
<td>1.84</td>
<td>2.26</td>
</tr>
</tbody>
</table>
### Cute, but these models cannot compete?

<table>
<thead>
<tr>
<th>bpd</th>
<th>2008-2020</th>
<th>2020-2021</th>
<th>ICLR 22</th>
<th>NeurIPS 22</th>
<th>ICLR 23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tabular</td>
<td>🙁</td>
<td>😊</td>
<td>🍰</td>
<td>🍰</td>
<td>🍰</td>
</tr>
<tr>
<td>MNIST</td>
<td>😱</td>
<td>😱</td>
<td>1.20</td>
<td>1.14</td>
<td></td>
</tr>
<tr>
<td>F-MNIST</td>
<td>😱</td>
<td>😱</td>
<td>3.34</td>
<td>3.27</td>
<td></td>
</tr>
<tr>
<td>EMNIST-L</td>
<td>😱</td>
<td>😱</td>
<td>1.80</td>
<td>1.58</td>
<td></td>
</tr>
<tr>
<td>CIFAR</td>
<td>😱</td>
<td>😱</td>
<td>🍰</td>
<td>4.38</td>
<td></td>
</tr>
<tr>
<td>Imagenet32</td>
<td>😱</td>
<td>😱</td>
<td>🍰</td>
<td>4.39</td>
<td></td>
</tr>
<tr>
<td>Imagenet64</td>
<td>😱</td>
<td>😱</td>
<td>🍰</td>
<td>4.12</td>
<td></td>
</tr>
</tbody>
</table>

- General-purpose architecture
- Custom GPU kernels
- Pruning without losing likelihood
- Latent Variable Distillation
Cute, but these models cannot compete?

<table>
<thead>
<tr>
<th>Model</th>
<th>bpd</th>
<th>2008-2020</th>
<th>2020-2021</th>
<th>ICLR 22</th>
<th>NeurIPS 22</th>
<th>ICLR 23</th>
<th>ICML 23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tabular</td>
<td>😞</td>
<td>😞</td>
<td>😋</td>
<td>😋</td>
<td>😋</td>
<td>😋</td>
<td>😋</td>
</tr>
<tr>
<td>MNIST</td>
<td>😞</td>
<td>😞 &gt; 1.67</td>
<td>1.20</td>
<td>1.14</td>
<td>😋</td>
<td>😋</td>
<td>😋</td>
</tr>
<tr>
<td>F-MNIST</td>
<td>😞</td>
<td>😞 &gt; 4.29</td>
<td>3.34</td>
<td>3.27</td>
<td>😋</td>
<td>😋</td>
<td>😋</td>
</tr>
<tr>
<td>EMNIST-L</td>
<td>😞</td>
<td>😞 &gt; 2.73</td>
<td>1.80</td>
<td>1.58</td>
<td>😋</td>
<td>😋</td>
<td>😋</td>
</tr>
<tr>
<td>CIFAR</td>
<td>😞</td>
<td>😞</td>
<td>😋 &gt; 5.50</td>
<td></td>
<td>4.38</td>
<td>3.87</td>
<td></td>
</tr>
<tr>
<td>Imagenet32</td>
<td>😞</td>
<td>😞</td>
<td>😋</td>
<td></td>
<td>4.39</td>
<td>4.06</td>
<td></td>
</tr>
<tr>
<td>Imagenet64</td>
<td>😞</td>
<td>😞</td>
<td>😋</td>
<td></td>
<td>4.12</td>
<td>3.80</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Flow</th>
<th>Hierarchical VAE</th>
<th>Diffusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR</td>
<td>3.35</td>
<td>3.08</td>
<td>2.65</td>
</tr>
<tr>
<td>Imagenet32</td>
<td>4.09</td>
<td>3.96</td>
<td>3.72</td>
</tr>
<tr>
<td>Imagenet64</td>
<td>3.81</td>
<td>-</td>
<td>3.40</td>
</tr>
</tbody>
</table>
Generate a sentence using "frisbee", "caught" and "dog", following the given order.
Generate a sentence using "frisbee", "caught" and "dog", following the given order.

After a perfect throw, the frisbee glided through the air, and the dog, with incredible agility, caught it mid-flight.
Generate a sentence using "frisbee", "caught" and "dog", following the given order.

After a perfect throw, the **frisbee** glided through the air, and the **dog**, with incredible agility, **caught** it mid-flight.

That's not correct. Generate a sentence using "frisbee", "caught" and "dog". The keywords should appear in the order as specified.

Here's the correct sentence: The **dog caught** the **frisbee** in mid-air, showing off its amazing catching skills.
A frisbee is caught by a dog.
A pair of frisbee players are caught in a dog fight.
Diffusion models are good at fine-grained details, but not so good at global consistency of generated images.
Inpainting/constrained generation is still challenging
What do we have?

Prefix: “The weather is”

Constraint $\alpha$: text contains “winter”

Model only does $p(\text{next-token}|\text{prefix}) = \begin{array}{c|c}
\text{cold} & 0.05 \\
\text{warm} & 0.10 \\
\end{array}$

Train some $q(. | \alpha)$ for a specific task distribution $\alpha \sim p_{\text{task}}$

(amortized inference, encoder, masked model, seq2seq, prompt tuning,...)

Train $q(\text{next-token}|\text{prefix}, \alpha)$
What do we need?

Prefix: “The weather is”

Constraint $\alpha$: text contains “winter”

Generate from $p(\text{next-token}|\text{prefix, } \alpha) =$

$$\propto \sum_{\text{text}} p(\text{next-token, text, prefix, } \alpha)$$

Marginalization!
Step 1: Distill an HMM $p_{hmm}$ that approximates $p_{gpt}$

1. HMM with 4096 hidden states and 50k emission tokens

2. Data sampled from GPT2-large (domain-adapted), minimizing $\text{KL}(p_{gpt} \parallel p_{HMM})$

3. Leverages latent variable distillation for training PCs at scale [ICLR 23]. (Cluster embeddings of examples to estimate latent $Z_i$)

CommonGen: a Challenging Benchmark

Given 3-5 keywords, generate a sentence using all keywords, in any order and any form of inflections. e.g.,

Input: snow drive car

Reference 1: A car drives down a snow covered road.

Reference 2: Two cars drove through the snow.

Constraint $\alpha$ in CNF: $$(w_{1,1} \lor \cdots \lor w_{1,d_1}) \land \cdots \land (w_{m,1} \lor \cdots \lor w_{m,d_m})$$

Each clause represents the inflections for one keyword.
Computing $p(\alpha \mid x_{1:t+1})$

For constraint $\alpha$ in CNF:

$$(w_{1,1} \lor \ldots \lor w_{1,d_1}) \land \ldots \land (w_{m,1} \lor \ldots \lor w_{m,d_m})$$

where each $w_{ij}$ is a keyword (i.e. a string of tokens), representing that $w_{ij}$ appears in the generated text.

e.g., $\alpha = ("swims" \lor "like swimming") \land ("lake" \lor "pool")$
Computing $p(\alpha | x_{1:t+1})$

For constraint $\alpha$ in CNF:

$$(w_{1,1} \lor \ldots \lor w_{1,d_1}) \land \ldots \land (w_{m,1} \lor \ldots \lor w_{m,d_m})$$

where each $w_{ij}$ is a keyword (i.e. a string of tokens), representing that $w_{ij}$ appears in the generated text.

E.g., $\alpha = ("swims" \lor "like swimming") \land ("lake" \lor "pool")$

Efficient algorithm:
For $m$ clauses and sequence length $n$, time-complexity for HMM generation is $O(2^{|m|n})$

Trick: dynamic programming with clever preprocessing and local belief updates
**GeLaTo**

**Overview**

**Lexical Constraint** $\alpha$: sentence contains keyword "winter"

**Constrained Generation**: $P_r(x_{t+1} | \alpha, x_{1:t} = "the weather is")$

- **Intractable**
  - Pre-trained Language Model
  - Minimize KL-divergence
  - $x_{t+1}$, $Pr_{LM}(x_{t+1} | x_{1:t})$
    - cold: 0.05
    - warm: 0.10

- **Efficient**
  - Tractable Probabilistic Model
  - $x_{t+1}$, $Pr_{TPM}(\alpha | x_{t+1}, x_{1:t})$
    - cold: 0.50
    - warm: 0.01

**Lexical Constraint** $\alpha$: sentence contains keyword “winter”

**Constrained Generation**: $\Pr(x_{t+1} | \alpha, x_{1:t} = "the weather is")$

---

**Pre-trained Language Model**

- $x_{t+1}$
- $\Pr_{LM}(x_{t+1} | x_{1:t})$
- cold: 0.05
- warm: 0.10

**Tractable Probabilistic Model**

- $x_{t+1}$
- $\Pr_{TPM}(\alpha | x_{t+1}, x_{1:t})$
- cold: 0.50
- warm: 0.01

---

$\Pr_{LM}(x_{t+1} | x_{1:t})$

- cold: 0.025
- warm: 0.001

---

Step 2: Control $p_{gpt}$ via $p_{hmm}$

Unsupervised

Language model is not fine-tuned/prompted to satisfy constraints

By Bayes rule:

$$p_{gpt}(x_{t+1} | x_{1:t}, \alpha) \propto p_{gpt}(\alpha | x_{1:t+1}) \cdot p_{gpt}(x_{t+1} | x_{1:t})$$

Assume $p_{hmm}(\alpha | x_{1:t+1}) \approx p_{gpt}(\alpha | x_{1:t+1})$, we generate from:

$$p(x_{t+1} | x_{1:t}, \alpha) \propto p_{hmm}(\alpha | x_{1:t+1}) \cdot p_{gpt}(x_{t+1} | x_{1:t})$$

<table>
<thead>
<tr>
<th>Method</th>
<th>ROUGE-L</th>
<th>Generation Quality</th>
<th>Constraint Satisfaction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dev</td>
<td>test</td>
<td>Coverage</td>
</tr>
<tr>
<td>Unsupervised</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InsNet (Lu et al., 2022a)</td>
<td>-</td>
<td>-</td>
<td>100.0</td>
</tr>
<tr>
<td>NeuroLogic (Lu et al., 2021)</td>
<td>-</td>
<td>41.9</td>
<td>24.7</td>
</tr>
<tr>
<td>A*esque (Lu et al., 2022b)</td>
<td>-</td>
<td>44.3</td>
<td>28.6</td>
</tr>
<tr>
<td>NADÔ (Meng et al., 2022)</td>
<td>-</td>
<td>-</td>
<td>26.2</td>
</tr>
<tr>
<td>GeLaTo</td>
<td>44.6</td>
<td>44.1</td>
<td>29.9</td>
</tr>
</tbody>
</table>

Step 2: Control $p_{gpt}$ via $p_{hmm}$

**Supervised**

Language model is fine-tuned to perform constrained generation (e.g. seq2seq)

Empirically $p_{HMM}(\alpha | x_{1:t+1}) \approx p_{gpt}(\alpha | x_{1:t+1})$ does not hold well enough;

we view $p_{HMM}(x_{t+1} | x_{1:t}, \alpha)$ and $p_{gpt}(x_{t+1} | x_{1:t})$ as classifiers trained for the same task with different biases; thus we generate from their *weighted geometric mean*:

$$p(x_{t+1} | x_{1:t}, \alpha) \propto p_{hmm}(x_{t+1} | x_{1:t}, \alpha)^w \cdot p_{gpt}(x_{t+1} | x_{1:t})^{1-w}$$

<table>
<thead>
<tr>
<th>Method</th>
<th>ROUGE-L</th>
<th>Generation Quality</th>
<th>Constraint Satisfaction</th>
<th>Success Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dev test</td>
<td>dev test</td>
<td>dev test</td>
<td>dev test</td>
</tr>
<tr>
<td>Supervised</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NeuroLogic (Lu et al., 2021)</td>
<td>- 42.8</td>
<td>- 26.7</td>
<td>- 14.7</td>
<td>- 97.7</td>
</tr>
<tr>
<td>A*esque (Lu et al., 2022b)</td>
<td>- 43.6</td>
<td>- 28.2</td>
<td>- 15.2</td>
<td>- 97.8</td>
</tr>
<tr>
<td>NADO (Meng et al., 2022)</td>
<td>44.4†</td>
<td>30.8</td>
<td>16.1†</td>
<td>97.1</td>
</tr>
<tr>
<td>GeLaTo</td>
<td>46.0</td>
<td>45.6</td>
<td>34.1</td>
<td>100.0</td>
</tr>
</tbody>
</table>

Advantages of GeLaTo:

1. Constraint $\alpha$ is guaranteed to be satisfied: for any next-token $x_{t+1}$ that would make $\alpha$ unsatisfiable, $p(x_{t+1} \mid x_{1:t}, \alpha) = 0$.

2. Training $p_{\text{hmm}}$ does not depend on $\alpha$, which is only imposed at inference (generation) time.

3. Can impose additional tractable constraints:
   - keywords follow a particular order
   - keywords appear at a particular position
   - keywords must not appear

Conclusion: you can control an intractable generative model using a tractable probabilistic circuit.
Inpainting/constrained generation is still challenging
Constrained posterior in diffusion models

Unconstrained denoising step:  \( p_\theta(x_{t-1}|x_t) := \sum_{\tilde{x}_0} q(x_{t-1}|\tilde{x}_0, x_t) \cdot p_\theta(\tilde{x}_0|x_t) \)

Constraint \( c \) on the generated image (e.g., inpainting)

Constrained denoising step:  \( p_\theta(x_{t-1}|x_t, c) := \sum_{\tilde{x}_0} q(x_{t-1}|\tilde{x}_0, x_t) \cdot p_\theta(\tilde{x}_0|x_t, c) \)

Computing or sampling from the constrained posterior \( p_\theta(\tilde{x}_0|x_t, c) \) is intractable for diffusion models.
Denoising \( p(\tilde{x}_0|x_t, x_0^k) \propto p_{DM}(\tilde{x}_0|x_t, x_0^k)\alpha \cdot p_{TPM}(\tilde{x}_0|x_t, x_0^k)^{1-\alpha} \)

From the diffusion model:
- Good at generating vivid details

From the probabilistic circuit:
- Exact samples – better global coherence
Controlling the denoiser with a probabilistic circuit
## High-resolution image benchmarks

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Mask</th>
<th>Tasks</th>
<th>Tiramisu (ours)</th>
<th>CoPaint</th>
<th>RePaint</th>
<th>DDNM</th>
<th>DDRM</th>
<th>DPS</th>
<th>Resampling</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CelebA-HQ</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Left</td>
<td></td>
<td></td>
<td>0.189</td>
<td><strong>0.185</strong></td>
<td>0.195</td>
<td>0.254</td>
<td>0.275</td>
<td>0.201</td>
<td>0.257</td>
</tr>
<tr>
<td>Top</td>
<td></td>
<td></td>
<td>0.187</td>
<td><strong>0.182</strong></td>
<td>0.187</td>
<td>0.248</td>
<td>0.267</td>
<td>0.187</td>
<td>0.251</td>
</tr>
<tr>
<td>Expand1</td>
<td></td>
<td></td>
<td><strong>0.454</strong></td>
<td>0.468</td>
<td>0.504</td>
<td>0.597</td>
<td>0.682</td>
<td>0.466</td>
<td>0.613</td>
</tr>
<tr>
<td>Expand2</td>
<td></td>
<td></td>
<td>0.442</td>
<td>0.455</td>
<td>0.480</td>
<td>0.585</td>
<td>0.686</td>
<td><strong>0.434</strong></td>
<td>0.601</td>
</tr>
<tr>
<td>V-strip</td>
<td></td>
<td></td>
<td><strong>0.487</strong></td>
<td>0.502</td>
<td>0.517</td>
<td>0.625</td>
<td>0.724</td>
<td>0.535</td>
<td>0.647</td>
</tr>
<tr>
<td>H-strip</td>
<td></td>
<td></td>
<td><strong>0.484</strong></td>
<td>0.488</td>
<td>0.517</td>
<td>0.626</td>
<td>0.731</td>
<td>0.492</td>
<td>0.639</td>
</tr>
<tr>
<td><strong>ImageNet</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Left</td>
<td></td>
<td></td>
<td><strong>0.286</strong></td>
<td>0.289</td>
<td>0.296</td>
<td>0.410</td>
<td>0.369</td>
<td>0.327</td>
<td>0.369</td>
</tr>
<tr>
<td>Top</td>
<td></td>
<td></td>
<td><strong>0.308</strong></td>
<td>0.312</td>
<td>0.336</td>
<td>0.427</td>
<td>0.373</td>
<td>0.343</td>
<td>0.368</td>
</tr>
<tr>
<td>Expand1</td>
<td></td>
<td></td>
<td><strong>0.616</strong></td>
<td>0.623</td>
<td>0.691</td>
<td>0.786</td>
<td>0.726</td>
<td>0.621</td>
<td>0.711</td>
</tr>
<tr>
<td>Expand2</td>
<td></td>
<td></td>
<td><strong>0.597</strong></td>
<td>0.607</td>
<td>0.692</td>
<td>0.799</td>
<td>0.724</td>
<td>0.618</td>
<td>0.721</td>
</tr>
<tr>
<td>V-strip</td>
<td></td>
<td></td>
<td>0.646</td>
<td>0.654</td>
<td>0.741</td>
<td>0.851</td>
<td>0.761</td>
<td><strong>0.637</strong></td>
<td>0.759</td>
</tr>
<tr>
<td>H-strip</td>
<td></td>
<td></td>
<td>0.657</td>
<td>0.660</td>
<td>0.744</td>
<td>0.851</td>
<td>0.753</td>
<td><strong>0.647</strong></td>
<td>0.774</td>
</tr>
<tr>
<td><strong>LSUN-Bedroom</strong></td>
<td></td>
<td></td>
<td><strong>0.285</strong></td>
<td>0.287</td>
<td>0.314</td>
<td>0.345</td>
<td>0.366</td>
<td>0.314</td>
<td>0.367</td>
</tr>
<tr>
<td>Left</td>
<td></td>
<td></td>
<td><strong>0.310</strong></td>
<td>0.323</td>
<td>0.347</td>
<td>0.376</td>
<td>0.368</td>
<td>0.355</td>
<td>0.372</td>
</tr>
<tr>
<td>Top</td>
<td></td>
<td></td>
<td><strong>0.615</strong></td>
<td>0.637</td>
<td>0.676</td>
<td>0.716</td>
<td>0.695</td>
<td>0.641</td>
<td>0.699</td>
</tr>
<tr>
<td>Expand1</td>
<td></td>
<td></td>
<td><strong>0.635</strong></td>
<td>0.641</td>
<td>0.666</td>
<td>0.720</td>
<td>0.691</td>
<td>0.638</td>
<td>0.690</td>
</tr>
<tr>
<td>Expand2</td>
<td></td>
<td></td>
<td><strong>0.672</strong></td>
<td>0.676</td>
<td>0.711</td>
<td>0.760</td>
<td>0.721</td>
<td>0.674</td>
<td>0.725</td>
</tr>
<tr>
<td>V-strip</td>
<td></td>
<td></td>
<td><strong>0.679</strong></td>
<td>0.686</td>
<td>0.722</td>
<td>0.766</td>
<td>0.726</td>
<td><strong>0.674</strong></td>
<td>0.724</td>
</tr>
<tr>
<td>H-strip</td>
<td></td>
<td></td>
<td><strong>0.474</strong></td>
<td>0.481</td>
<td>0.518</td>
<td>0.596</td>
<td>0.591</td>
<td>0.489</td>
<td>0.571</td>
</tr>
</tbody>
</table>

Average: **0.474** 0.481 0.518 0.596 0.591 0.489 0.571
Qualitative results on high-resolution image datasets

<table>
<thead>
<tr>
<th></th>
<th>CelebA-HQ</th>
<th></th>
<th></th>
<th>ImageNet</th>
<th></th>
<th></th>
<th>LSUN-Bedrooms</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Left</td>
<td>Expand1</td>
<td>Expand2</td>
<td>V-strip</td>
<td>Left</td>
<td>Expand1</td>
<td>Expand2</td>
<td>V-strip</td>
</tr>
</tbody>
</table>
Outline

1. What are probabilistic circuits?
   *tractable deep generative models*

2. What are they useful for?
   *controlling generative AI*

3. What is the underlying theory?
   *probability generating polynomials*
Probabilistic circuits seem awfully general. Are all tractable probabilistic models probabilistic circuits?
Enter: Determinantal Point Processes (DPPs)

DPPs are models where probabilities are specified by (sub)determinants

\[ L = \begin{bmatrix}
1 & 0.9 & 0.8 & 0 \\
0.9 & 0.97 & 0.96 & 0 \\
0.8 & 0.96 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \]

- Global Negative Dependence
- Diversity in recommendation systems
- Tractable likelihoods and marginals

\[ \Pr_L(X_1 = 1, X_2 = 0, X_3 = 1, X_4 = 0) = \frac{1}{\det(L + I)} \det(L_{\{1,2\}}) \]
Are all tractable probabilistic models probabilistic circuits?

Probabilistic Circuits

Determinantal Point Processes

Bounded Treewidth Graphical Models
Are all tractable probabilistic models probabilistic circuits?

Probabilistic Circuits

Positive Dependence

Fully Factorized

Determinantal Point Processes

A separation between PCs and DPPs

**Theorem** (Martens and Medabalimi, 2014). Let $P_n$ be the uniform distribution over spanning trees on $K_n$. For $n \geq 20$, the size of any smooth and decomposable PC that represents $P_n$ is at least $2^{n/30240}$.

**Theorem** (Snell, 1995). The uniform distribution over spanning trees on the complete graph $K_n$ is a DPP over $\binom{n}{2}$ edges.
Probabilistic Generating Circuits

A Tractable Unifying Framework for PCs and DPPs

### Probability Generating Functions

$$g_\beta = 0.16z_1z_2z_3 + 0.04z_1z_2 + 0.08z_1z_3 + 0.02z_1 + 0.48z_2z_3 + 0.12z_2 + 0.08z_3 + 0.02.$$
Probability Generating Functions

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$\Pr_\beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.08</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.12</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.48</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.08</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.04</td>
</tr>
<tr>
<td><strong>1</strong></td>
<td><strong>1</strong></td>
<td><strong>1</strong></td>
<td><strong>0.16</strong></td>
</tr>
</tbody>
</table>

$$g_\beta = 0.16z_1z_2z_3 + 0.04z_1z_2 + 0.08z_1z_3 + 0.02z_1 + 0.48z_2z_3 + 0.12z_2 + 0.08z_3 + 0.02.$$

$$g_\beta = (0.1(z_1 + 1)(6z_2 + 1) - 0.4z_1z_2)(0.8z_3 + 0.2).$$
1. Sum nodes \( \bigoplus \) with weighted edges to children.
2. Product nodes \( \bigotimes \) with unweighted edges to children.
3. Leaf nodes: \( z_i \) or constant.

\[
g_\beta = (0.1(z_1 + 1)(6z_2 + 1) - 0.4z_1z_2)(0.8z_3 + 0.2)
\]
PGCs Support Tractable **Likelihoods**

**How to extract the right monomial’s coefficient?**

\[ \Pr(X_1 = 1, X_2 = 0, \ldots) = ? \]
PGCs Support Tractable **Likelihoods**

$z_i = \begin{cases} \text{t} & \text{if } X_i = 1 \\ 0 & \text{if } X_i = 0 \end{cases}$

Monomials setting to true variables that must be false are 0-ed out.

- Only the monomial that sets all required variables to true has max degree.

$\Pr(X_1 = 1, X_2 = 0, ...) = ?$

$p(t) = \alpha_k t^k + \cdots + \alpha_1 t$

How to extract the right monomial’s coefficient?

$O$(circuit size x degree)

$\alpha_k$ gives the answer
PGCs Support Tractable Marginals

How to sum the right monomial’s coefficients?

Pr(X₁ = 1, X₂ = 0, ...) =?
How to sum the right monomial’s coefficients?

\[ z_i = \begin{cases} 
  t & \text{if } X_i = 1 \\
  0 & \text{if } X_i = 0 \\
  1 & \text{if } X_i = \text{?} 
\end{cases} \]

- Monomials setting to true variables that must be false are 0-ed out
- Other monomials contribute to result.
- Only monomials that set all required variables to true have max degree.

\[ p(t) = \alpha_k t^k + \cdots + \alpha_1 t \]
Example

\[ z_i = \begin{cases} 
  t & \text{if } X_i = 1 \\
  0 & \text{if } X_i = 0 \\
  1 & \text{if } X_i = ? 
\end{cases} \]

Pr(\(X_2 = 1, X_3 = 0\)) =?
Example

\[ z_i = \begin{cases} 
  t & \text{if } X_i = 1 \\
  0 & \text{if } X_i = 0 \\
  1 & \text{if } X_i = ? 
\end{cases} \]

\[ \Pr(X_2 = 1, X_3 = 0) = ? \]
Example

\[ z_i = \begin{cases} 
  t & \text{if } X_i = 1 \\
  0 & \text{if } X_i = 0 \\
  1 & \text{if } X_i = ? 
\end{cases} \]

\[ \Pr(X_2 = 1, X_3 = 0) =? \]

\[
\begin{array}{cccc|c}
X_1 & X_2 & X_3 & \Pr_{\beta} \\
0 & 0 & 0 & 0.02 \\
0 & 0 & 1 & 0.08 \\
0 & 1 & 0 & 0.12 \\
0 & 1 & 1 & 0.48 \\
1 & 0 & 0 & 0.02 \\
1 & 0 & 1 & 0.08 \\
1 & 1 & 0 & 0.04 \\
1 & 1 & 1 & 0.16 \\
\end{array}
\]
Probabilistic circuits are probabilistic generating circuits

PCs represents probability mass functions:

\[ m_\beta = 0.16X_1X_2X_3 + 0.04X_1X_2\overline{X}_3 + 0.08X_1\overline{X}_2X_3 + 0.02X_1\overline{X}_2\overline{X}_3 \]
\[ + 0.48\overline{X}_1X_2X_3 + 0.12\overline{X}_1X_2\overline{X}_3 + 0.08\overline{X}_1\overline{X}_2X_3 + 0.02\overline{X}_1\overline{X}_2\overline{X}_3 \]

PGCs represent probability generating functions:

\[ g_\beta = 0.16z_1z_2z_3 + 0.04z_1z_2 + 0.08z_1z_3 + 0.02z_1 \]
\[ + 0.48z_2z_3 + 0.12z_2 + 0.08z_1z_3 + 0.02 \]

Given a smooth & decomposable PC, by setting \( \overline{X}_i \) to 1, and \( X_i \) to \( z_i \), we obtain an equivalent PGC.
DPPs are probabilistic generating circuits

The generating polynomial for a DPP with kernel $L$ is given by:

$$g_L = \frac{1}{\det(L + I)} \det(I + L \text{diag}(z_1, \ldots, z_n)).$$

We need it as a sum of products to obtain a Probabilistic Generating Circuit.
DPPs are probabilistic generating circuits

The generating polynomial for a DPP with kernel $L$ is given by:

$$g_L = \frac{1}{\det(L + I)} \det(I + L \text{diag}(z_1, \ldots, z_n)).$$

We need it as a sum of products to obtain a Probabilistic Generating Circuit.
DPPs are probabilistic generating circuits

The generating polynomial for a DPP with kernel $L$ is given by:

$$g_L = \frac{1}{\det(L + I)} \det(I + L \text{diag}(z_1, \ldots, z_n)).$$

**Constant**

**Division-free determinant algorithm**
(Samuelson-Berkowitz algorithm)

$g_L$ can be represented as a PGC of size $O(n^4)$

Probabilistic **generating** circuits seem awfully general.

Are all tractable probabilistic models probabilistic **generating** circuits?
Queries as pipelines: KLD

\[ \text{KLD}(p \parallel q) = \int p(x) \times \log((p(x)/q(x)))dX \]
Queries as pipelines: Cross Entropy

\[ H(p, q) = \int p(x) \times \log(q(x)) \, dx \]

\[ p \quad \rightarrow \quad \bigtimes \quad \rightarrow \quad \int \]

\[ q \quad \rightarrow \quad \log \quad \rightarrow \quad r \]

⇒ we can reuse the operations!

Determinism

A sum node is \textit{deterministic} if only one of its children outputs non-zero for any input.

\[ \arg\max_x p(x) \]

\textit{Deterministic circuit}

Darwiche and Marquis, “A Knowledge Compilation Map”, 2002
<table>
<thead>
<tr>
<th>Operation</th>
<th>Tractability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LOG</strong></td>
<td><strong>Input conditions</strong></td>
</tr>
<tr>
<td>( \log(p) )</td>
<td>Sm, Dec, Det</td>
</tr>
</tbody>
</table>

**smooth, decomposable, deterministic**

**smooth, decomposable**
## Tractable Circuit Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Input properties</th>
<th>Tractability</th>
<th>Hardness</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sum</strong></td>
<td>$\theta_1 p + \theta_2 q$</td>
<td>(+Cmp)</td>
<td>(+SD)</td>
</tr>
<tr>
<td><strong>Product</strong></td>
<td>$p \cdot q$</td>
<td>Cmp (+Det, +SD)</td>
<td>Dec (+Det, +SD)</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>$p^n, n \in \mathbb{N}$</td>
<td>SD (+Det)</td>
<td>SD (+Det)</td>
</tr>
<tr>
<td></td>
<td>$p^\alpha, \alpha \in \mathbb{R}$</td>
<td>Sm, Dec, Det (+SD)</td>
<td>Sm, Dec, Det (+SD)</td>
</tr>
<tr>
<td><strong>Quotient</strong></td>
<td>$p/q$</td>
<td>Cmp; $q$ Det (+p Det, +SD)</td>
<td>Dec (+Det, +SD)</td>
</tr>
<tr>
<td><strong>Log</strong></td>
<td>$\log(p)$</td>
<td>Sm, Dec, Det</td>
<td>Sm, Dec</td>
</tr>
<tr>
<td><strong>Exp</strong></td>
<td>$\exp(p)$</td>
<td>linear</td>
<td>SD</td>
</tr>
</tbody>
</table>

Inference by tractable operations

systematically derive tractable inference algorithm of complex queries

<table>
<thead>
<tr>
<th></th>
<th>Query</th>
<th>Tract. Conditions</th>
<th>Hardness</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CROSS ENTROPY</strong></td>
<td>$-\int p(x) \log q(x) , dX$</td>
<td>Cmp, q Det</td>
<td>#P-hard w/o Det</td>
</tr>
<tr>
<td><strong>SHANNON ENTROPY</strong></td>
<td>$-\sum p(x) \log p(x)$</td>
<td>Sm, Dec, Det</td>
<td>coNP-hard w/o Det</td>
</tr>
<tr>
<td><strong>Rényi Entropy</strong></td>
<td>$(1-\alpha)^{-1} \log \int p^\alpha(x) , dX$, $\alpha \in \mathbb{N}$</td>
<td>SD</td>
<td>#P-hard w/o SD</td>
</tr>
<tr>
<td><strong>Mutual Information</strong></td>
<td>$\int p(x,y) \log(p(x,y)/(p(x)p(y)))$</td>
<td>Sm, SD, Det*</td>
<td>coNP-hard w/o SD</td>
</tr>
<tr>
<td><strong>Kullback-Leibler Div.</strong></td>
<td>$\int p(x) \log(p(x)/q(x)) , dX$</td>
<td>Cmp, Det</td>
<td>#P-hard w/o Det</td>
</tr>
<tr>
<td><strong>Rényi’s Alpha Div.</strong></td>
<td>$(1-\alpha)^{-1} \log \int p^\alpha(x)q^{1-\alpha}(x) , dX$, $\alpha \in \mathbb{N}$</td>
<td>Cmp, q Det</td>
<td>#P-hard w/o Det</td>
</tr>
<tr>
<td><strong>Itakura-Saito Div.</strong></td>
<td>$(1-\alpha)^{-1} \log \int p^\alpha(x)q^{1-\alpha}(x) , dX$, $\alpha \in \mathbb{R}$</td>
<td>Cmp, Det</td>
<td>#P-hard w/o Det</td>
</tr>
<tr>
<td><strong>Cauchy-Schwarz Div.</strong></td>
<td>$\int [p(x)/q(x) - \log(p(x)/q(x)) - 1] , dX$</td>
<td>Cmp</td>
<td>#P-hard w/o Det</td>
</tr>
<tr>
<td><strong>Squared Loss</strong></td>
<td>$-\log \frac{\int p(x)q(x) , dX}{\sqrt{\int p^2(x)dX \int q^2(x)dX}}$</td>
<td>Cmp</td>
<td>#P-hard w/o Cmp</td>
</tr>
<tr>
<td></td>
<td>$\int (p(x) - q(x))^2 , dX$</td>
<td></td>
<td>#P-hard w/o Cmp</td>
</tr>
</tbody>
</table>
Conclusions

1. What are probabilistic circuits?
   *tractable deep generative models*

2. What are they useful for?
   *controlling generative AI*

3. What is the underlying theory?
   *probability generating polynomials*
Thanks

This was the work of many wonderful students/postdocs/collaborators!

References: http://starai.cs.ucla.edu/publications/