A Probabilistic Neuro-symbolic Layer for Algebraic Constraint Satisfaction

Leander Kurscheidt
University of Edinburgh, UK

Paolo Morettin
University of Trento, Italy

Roberto Sebastiani
University of Trento, Italy

Andrea Passerini
University of Trento, Italy

Antonio Vergari
University of Edinburgh, UK

UAI 2025 Oral

A Probabilistic Neuro-symbolic Layer for Algebraic Constraint Satisfaction

Leander Kurscheidt
University of Edinburgh, UK

Paolo Morettin
University of Trento, Italy

Roberto Sebastiani
University of Trento, Italy

Andrea Passerini
University of Trento, Italy

Antonio Vergari
University of Edinburgh, UK

UAI 2025 Oral

A Probabilistic Neuro-symbolic Layer for Algebraic Constraint Satisfaction

Leander Kurscheidt
University of Edinburgh, UK

Paolo Morettin
University of Trento, Italy

Roberto Sebastiani
University of Trento, Italy

Andrea Passerini
University of Trento, Italy

Antonio Vergari
University of Edinburgh, UK

UAI 2025 Oral

A Probabilistic Neuro-symbolic Layer for Algebraic Constraint Satisfaction

Leander Kurscheidt
University of Edinburgh, UK

Paolo Morettin
University of Trento, Italy

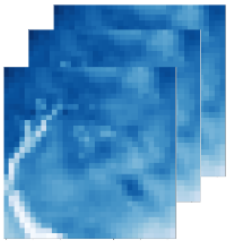
Roberto Sebastiani
University of Trento, Italy

Andrea Passerini
University of Trento, Italy

Antonio Vergari
University of Edinburgh, UK

UAI 2025 Oral

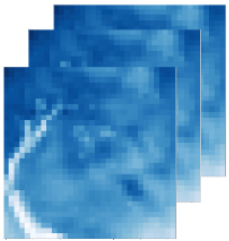
Climate Science



$$\sum y_i = y_{\text{obs}}$$

[Harder et al. 2023]

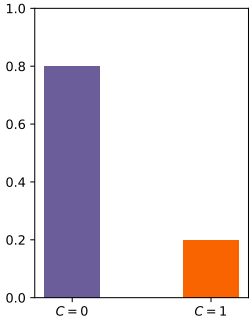
Climate Science



$$\sum y_i = y_{\text{obs}}$$

[Harder et al. 2023]

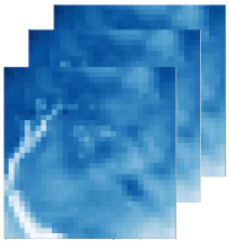
Fairness



$$|P(\mathcal{A}|C{=}0) - P(\mathcal{A}|C{=}1)| \leq \epsilon$$

[Dwork et al. 2012]

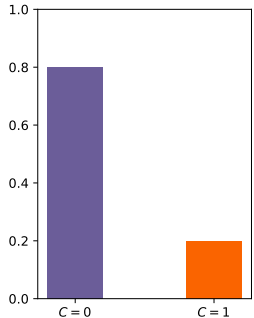
Climate Science



$$\sum y_i = y_{\text{obs}}$$

[Harder et al. 2023]

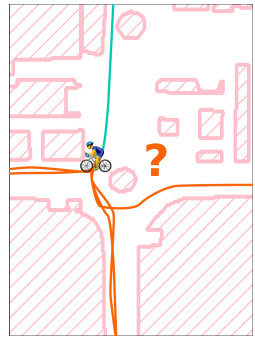
Fairness



$$|P(\mathcal{A}|C{=}0) - P(\mathcal{A}|C{=}1)| \leq \epsilon$$

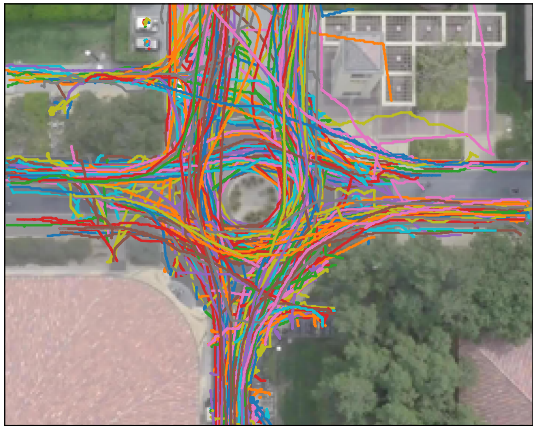
[Dwork et al. 2012]

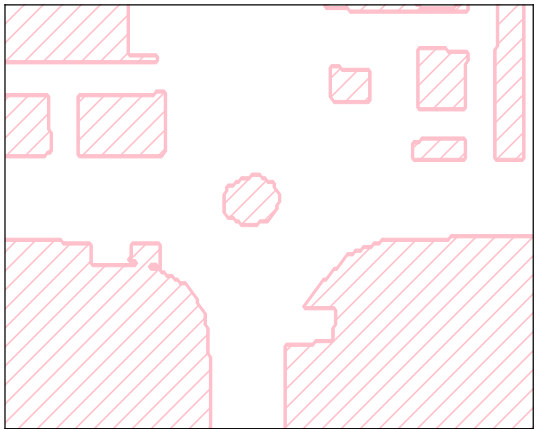
Trajectory Prediction

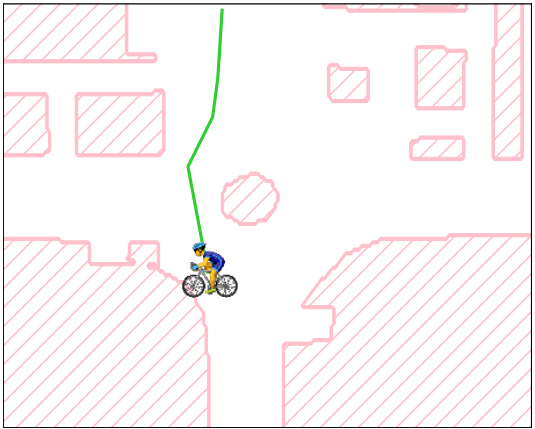


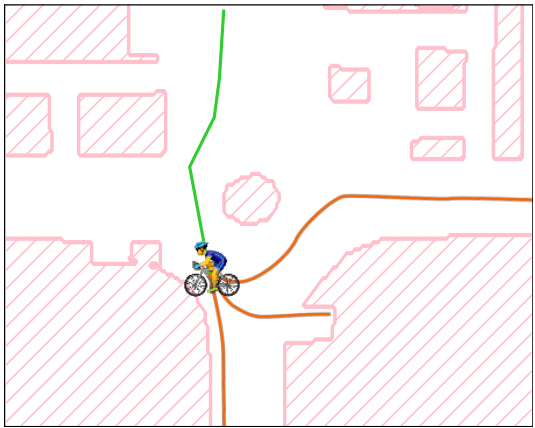
$$y \notin \boxed{}$$

[Robicquet et al. 2016]





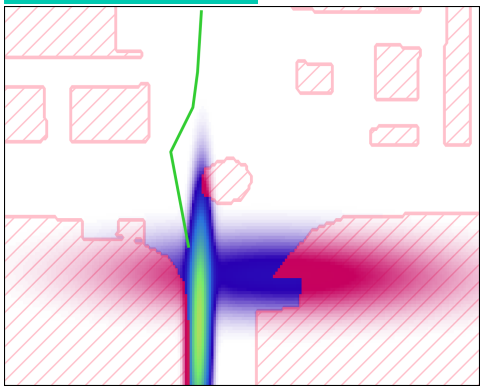




$$p(\mathbf{y}|\mathbf{x}) = ?$$

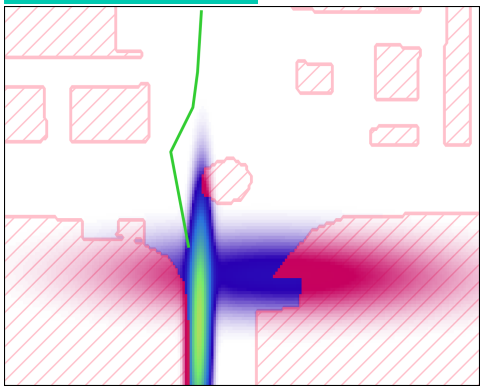
How to model $p(\mathbf{y}|\mathbf{x})$ with a neural network?

NN+GMM



Bishop, “Mixture density networks”, Tech Report, 1994

NN+GMM

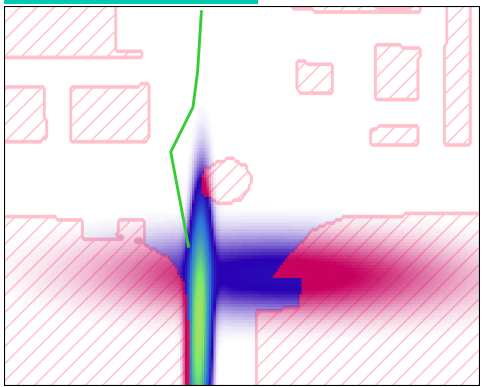


$$p(\mathbf{y}|\mathbf{x}) = \mathrm{GMM}(\mathbf{y}; \boldsymbol{\lambda}{=}f_{\boldsymbol{\theta}}(\mathbf{x}))$$

with $\boldsymbol{\lambda} = (\boldsymbol{\pi}, \{(\mu_i, \Sigma_i)\}_{i=1}^{k})$

Bishop, "Mixture density networks", Tech Report, 1994

NN+GMM



issue: put densities/mass where constraints are not satisfied!

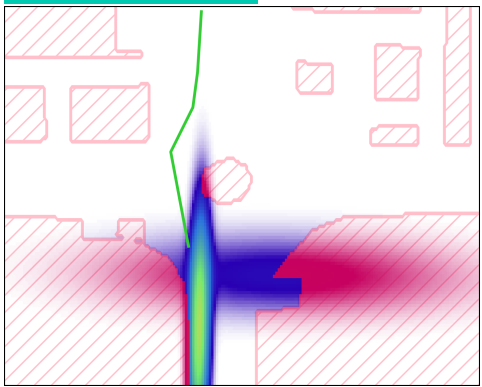
⇒ *loosing probability mass*

⇒ *sampling invalid points*

⇒ *unreliable predictions*

Bishop, "Mixture density networks", Tech Report, 1994

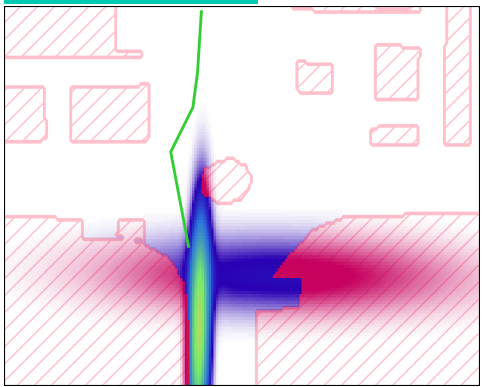
NN+GMM



Can we do better?

Bishop, "Mixture density networks", Tech Report, 1994

NN+GMM



Can we do better?

Yes! A plug&play layer that

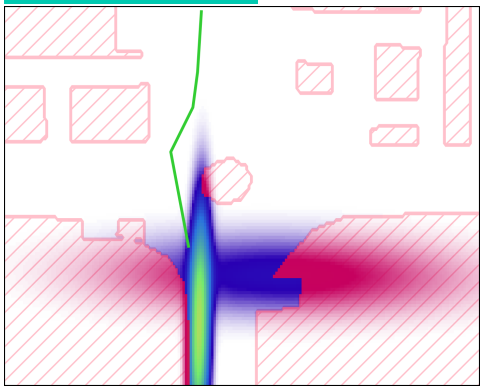
⇒ *puts mass inside constrains*

⇒ *guarantees valid predictions*

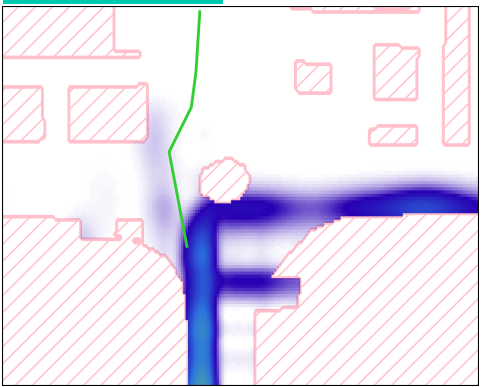
⇒ *end-to-end differentiable*

Bishop, "Mixture density networks", Tech Report, 1994

NN+GMM



NN+PAL



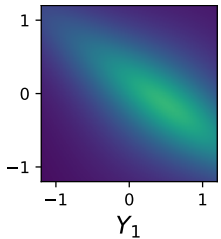
Bishop, “Mixture density networks”, Tech Report, 1994

Enter PAL

PAL Equation

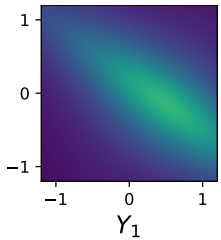
$$p_{\boldsymbol{\Theta}}(\mathbf{Y} \mid \mathbf{x}) = \frac{q(\mathbf{Y}; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x})) \cdot \mathbb{1}\{\mathbf{Y} \models \varphi\}}{\int q(\mathbf{y}'; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x})) \mathbb{1}\{\mathbf{y}' \models \varphi\} \, d\mathbf{Y}'}$$

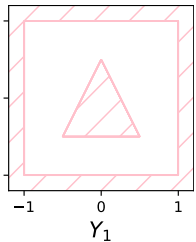
PAL Equation



$$p_{\Theta}(\mathbf{Y} \mid \mathbf{x}) = \frac{q(\mathbf{Y}; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x})) \cdot \mathbb{1}\{\mathbf{Y} \models \varphi\}}{\int q(\mathbf{y}'; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x})) \mathbb{1}\{\mathbf{y}' \models \varphi\} \, d\mathbf{Y}'}$$

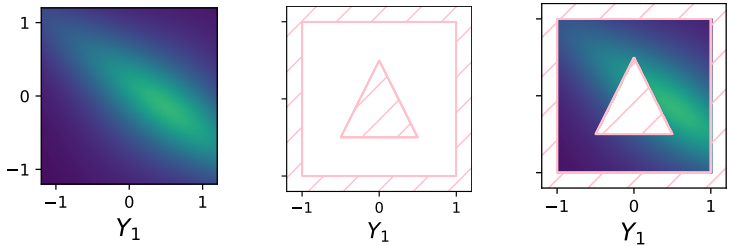
PAL Equation





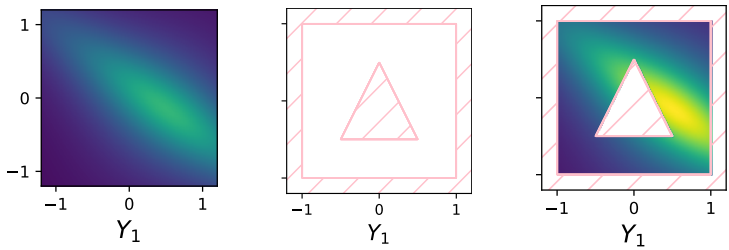
$$p_{\boldsymbol{\Theta}}(\mathbf{Y} \mid \mathbf{x}) = \frac{q(\mathbf{Y}; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x})) \cdot \mathbb{1}\{\mathbf{Y} \models \varphi\}}{\int q(\mathbf{y}'; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x})) \mathbb{1}\{\mathbf{y}' \models \varphi\} \, d\mathbf{Y}'}$$

PAL Equation



$$p_{\Theta}(\mathbf{Y} \mid \mathbf{x}) = \frac{q(\mathbf{Y}; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x})) \cdot \mathbb{1}\{\mathbf{Y} \models \varphi\}}{\int q(\mathbf{y}'; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x})) \mathbb{1}\{\mathbf{y}' \models \varphi\} \, d\mathbf{Y}'}$$

PAL Equation



$$p_{\boldsymbol{\Theta}}(\mathbf{Y} \mid \mathbf{x}) = \frac{q(\mathbf{Y}; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x})) \cdot \mathbb{1}\{\mathbf{Y} \models \varphi\}}{\int q(\mathbf{y}'; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x})) \mathbb{1}\{\mathbf{y}' \models \varphi\} \, d\mathbf{Y}'}$$

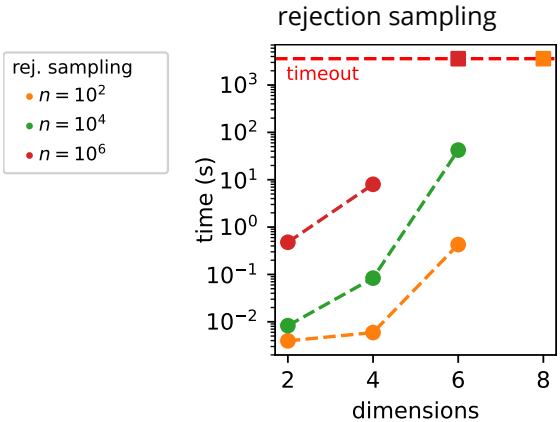
Belle, Passerini, and Van den Broeck, "Probabilistic Inference in Hybrid Domains by Weighted Model Integration", IJCAI, 2015

How to compute the normalization constant?

How to compute the normalization constant?

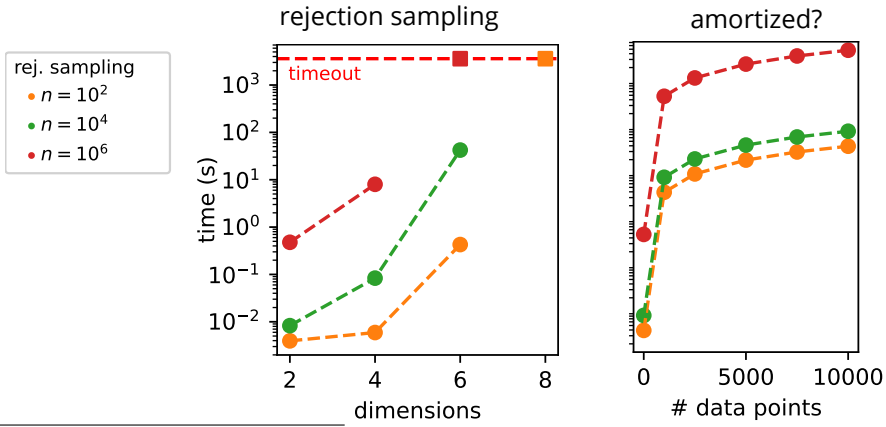
$\Rightarrow$ ***numerical approximation!?***

Numerical Approximation



Smet et al., "Neural probabilistic logic programming in discrete-continuous domains", UAI, 2023

Numerical Approximation



Smet et al., "Neural probabilistic logic programming in discrete-continuous domains", UAI, 2023

How to compute the normalization constant?

$\Rightarrow$ ***numerical approximation!?*** **no!**

Can we pick density q and constraints φ such that integration is efficient — and only needed once?

PAL Equation

$$p_{\boldsymbol{\Theta}}(\mathbf{Y} \mid \mathbf{x}) = \frac{q(\mathbf{Y}; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x})) \cdot \mathbb{1}\{\mathbf{Y} \models \varphi\}}{\int q(\mathbf{y}'; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x})) \mathbb{1}\{\mathbf{y}' \models \varphi\} \, d\mathbf{Y}'}$$

The Constraints

$$\mathbb{1}\{\mathbf{Y} \models \varphi\}$$

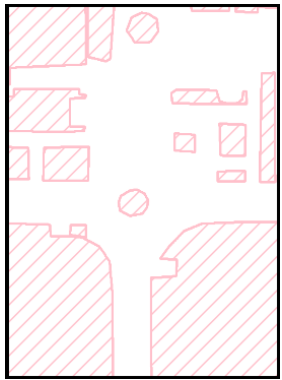
$\varphi \in$ satisfiability modulo linear real arithmetic (SMT($\mathcal{LRA}$))

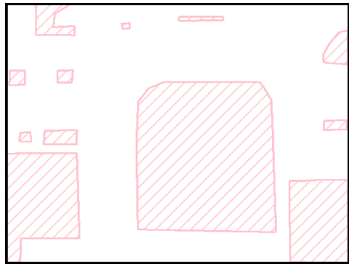
Barrett and Tinelli, "Satisfiability modulo theories", Handbook of Model Checking, 2018

The Constraints

$$\mathbb{1}\{\mathbf{Y} \models \varphi\}$$

$\varphi \in$ satisfiability modulo linear real arithmetic ($\mathsf{SMT}(\mathcal{LRA})$)

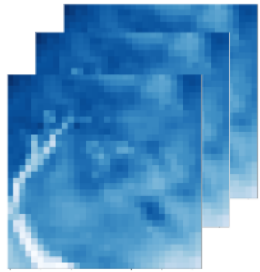




Barrett and Tinelli, "Satisfiability modulo theories", Handbook of Model Checking, 2018

The Constraints

$$\mathbb{1}\{\mathbf{Y} \models \varphi\}$$



$\varphi \in$ satisfiability modulo linear real arithmetic ($\mathsf{SMT}(\mathcal{LRA})$)

Barrett and Tinelli, "Satisfiability modulo theories", Handbook of Model Checking, 2018

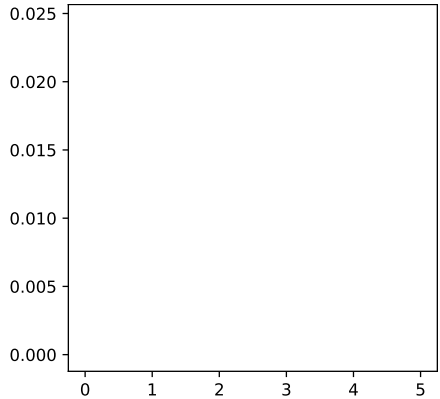
PAL Equation

$$p_{\Theta}(\mathbf{Y} \mid \mathbf{x}) = \frac{q(\mathbf{Y}; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x})) \cdot \mathbb{1}\{\mathbf{Y} \models \varphi\}}{\int q(\mathbf{y}'; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x})) \mathbb{1}\{\mathbf{y}' \models \varphi\} \, d\mathbf{Y}'}$$

The Distribution

$$q(\mathbf{Y}; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x}))$$

$\Rightarrow$ *piecewise polynomials, e.g., splines*



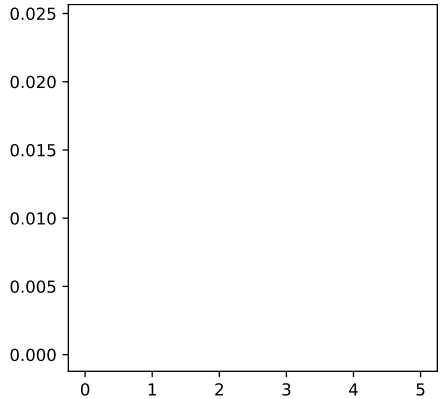
Zeng et al., "Probabilistic Inference with Algebraic Constraints: Theoretical Limits and Practical Approximations", NeurIPS, 2020

The Distribution

$$q(\mathbf{Y}; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x}))$$

$\Rightarrow$ *piecewise polynomials, e.g., splines*

$\Rightarrow$ *squared for non-negativity*



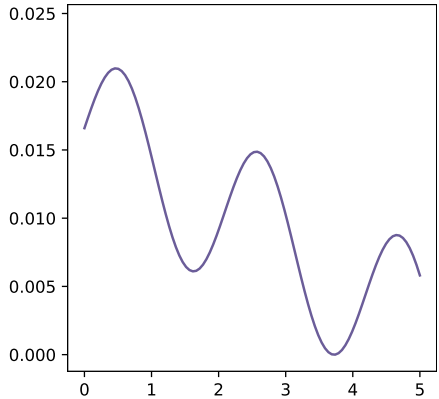
Zeng et al., "Probabilistic Inference with Algebraic Constraints: Theoretical Limits and Practical Approximations", NeurIPS, 2020

The Distribution

$$q(\mathbf{Y}; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x}))$$

$\Rightarrow$ *piecewise polynomials, e.g., splines*

$\Rightarrow$ *squared for non-negativity*



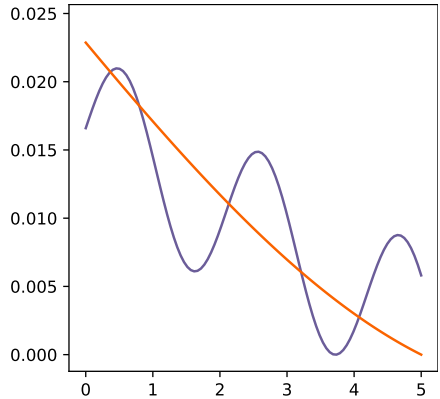
Zeng et al., "Probabilistic Inference with Algebraic Constraints: Theoretical Limits and Practical Approximations", NeurIPS, 2020

The Distribution

$$q(\mathbf{Y}; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x}))$$

$\Rightarrow$ *piecewise polynomials, e.g., splines*

$\Rightarrow$ *squared for non-negativity*



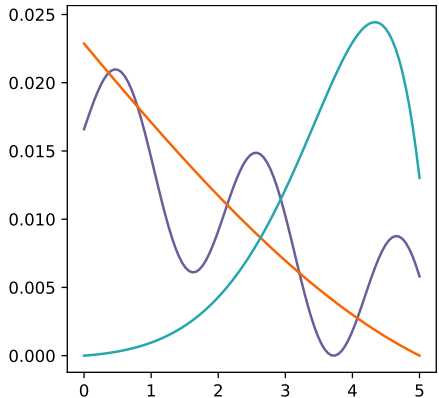
Zeng et al., "Probabilistic Inference with Algebraic Constraints: Theoretical Limits and Practical Approximations", NeurIPS, 2020

The Distribution

$$q(\mathbf{Y}; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x}))$$

$\Rightarrow$ *piecewise polynomials, e.g., splines*

$\Rightarrow$ *squared for non-negativity*



Zeng et al., "Probabilistic Inference with Algebraic Constraints: Theoretical Limits and Practical Approximations", NeurIPS, 2020

The Distribution

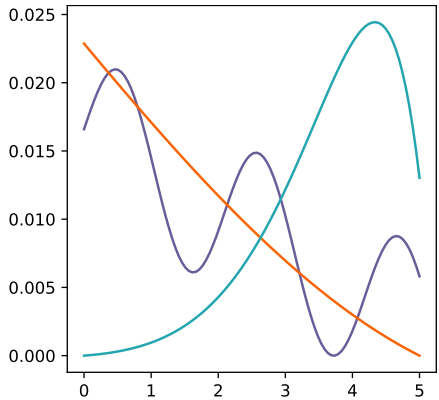
$$q(\mathbf{Y}; \boldsymbol{\lambda} = f_{\psi}(\mathbf{x}))$$

⇒ *piecewise polynomials, e.g., splines*

⇒ *squared for non-negativity*

⇒ *exact integration on a simplex*

[Grundmann and Möller 1978]



Zeng et al., "Probabilistic Inference with Algebraic Constraints: Theoretical Limits and Practical Approximations", NeurIPS, 2020

PAL Equation

$$p_{\boldsymbol{\Theta}}(\mathbf{Y} \mid \mathbf{x}) = \frac{q(\mathbf{Y}; \boldsymbol{\lambda} = f_{\boldsymbol{\psi}}(\mathbf{x})) \cdot \mathbb{1}\{\mathbf{Y} \models \varphi\}}{\int q(\mathbf{y}'; \boldsymbol{\lambda} = f_{\boldsymbol{\psi}}(\mathbf{x})) \mathbb{1}\{\mathbf{y}' \models \varphi\} \, d\mathbf{Y}'}$$

The Integral

$$\int q(\mathbf{y}'; \boldsymbol{\lambda}) \mathbb{1}\{\mathbf{y}' \models \varphi\} \, d\mathbf{Y}'$$

I. we introduce a new, GPU-accelerated integration backend called **GASP!**

II. it calculates the symbolic integral in seconds
   $\Rightarrow$ **8 seconds** for a **81 piece** squared spline (**deg. 12**) on the SDD-Dataset
   $\Rightarrow$ up to 2 **orders of magnitudes** faster than previous WMI-Solvers

III. the result is a function of $\boldsymbol{\lambda}$
   $\Rightarrow$ **compile only once,** evaluate cheaply many times
   $\Rightarrow$ **training independent on constraint complexity**

The Integral

$$\int q(\mathbf{y}';\boldsymbol{\lambda})\mathbb{1}\{\mathbf{y}' \models \varphi\}\, d\mathbf{Y}'$$

I. we introduce a new, GPU-accelerated integration backend called **GASP!**

II. it calculates the symbolic integral in seconds
   $\Rightarrow$ **8 seconds** for a **81 piece** squared spline (**deg. 12**) on the SDD-Dataset
   $\Rightarrow$ up to 2 **orders of magnitudes** faster than previous WMI-Solvers

III. the result is a function of $\boldsymbol{\lambda}$
   $\Rightarrow$ **compile only once,** evaluate cheaply many times
   $\Rightarrow$ **training independent on constraint complexity**

The Integral

$$\int q(\mathbf{y}'; \boldsymbol{\lambda}) \mathbb{1}\{\mathbf{y}' \models \varphi\} \, d\mathbf{Y}'$$

$$= I(\boldsymbol{\lambda})$$

I. we introduce a new, GPU-accelerated integration backend called **GASP!**

II. it calculates the symbolic integral in seconds

$\Rightarrow$ **8 seconds** for a **81 piece** squared spline (**deg. 12**) on the SDD-Dataset

$\Rightarrow$ up to 2 **orders of magnitudes** faster than previous WMI-Solvers

III. the result is a function of $\boldsymbol{\lambda}$

$\Rightarrow$ **compile only once,** evaluate cheaply many times

$\Rightarrow$ **training independent on constraint complexity**

The Integral

$$\int q(\mathbf{y}'; \boldsymbol{\lambda}) \mathbb{1}\{\mathbf{y}' \models \varphi\} \, d\mathbf{Y}'$$

$$= I(\boldsymbol{\lambda})$$

I. we introduce a new, GPU-accelerated integration backend called **GASP!**

II. it calculates the symbolic integral in seconds

$\Rightarrow$ **8 seconds** for a **81 piece** squared spline (**deg. 12**) on the SDD-Dataset

$\Rightarrow$ up to 2 **orders of magnitudes** faster than previous WMI-Solvers

III. the result is a function of $\boldsymbol{\lambda}$

$\Rightarrow$ **compile only once,** evaluate cheaply many times

$\Rightarrow$ **training independent on constraint complexity**

The Integral

$$\int q(\mathbf{y}'; \boldsymbol{\lambda}) \mathbb{1}\{\mathbf{y}' \models \varphi\} \, d\mathbf{Y}'$$

$$= I(\boldsymbol{\lambda})$$

I. we introduce a new, GPU-accelerated integration backend called **GASP!**

II. it calculates the symbolic integral in seconds
   $\Rightarrow$ **8 seconds** for a **81 piece** squared spline (**deg. 12**) on the SDD-Dataset
   $\Rightarrow$ up to 2 **orders of magnitudes** faster than previous WMI-Solvers

III. the result is a function of $\boldsymbol{\lambda}$
   $\Rightarrow$ **compile only once,** evaluate cheaply many times
   $\Rightarrow$ **training independent on constraint complexity**

The Integral

$$\int q(\mathbf{y}'; \boldsymbol{\lambda}) \mathbb{1}\{\mathbf{y}' \models \varphi\} \, d\mathbf{Y}'$$

$$= I(\boldsymbol{\lambda})$$

I. we introduce a new, GPU-accelerated integration backend called **GASP!**

II. it calculates the symbolic integral in seconds
   $\Rightarrow$ **8 seconds** for a **81 piece** squared spline (**deg. 12**) on the SDD-Dataset
   $\Rightarrow$ up to 2 **orders of magnitudes** faster than previous WMI-Solvers

III. the result is a function of $\boldsymbol{\lambda}$
   $\Rightarrow$ **compile only once,** evaluate cheaply many times
   $\Rightarrow$ **training independent on constraint complexity**

The Integral

$$\int q(\mathbf{y}'; \boldsymbol{\lambda}) \mathbb{1}\{\mathbf{y}' \models \varphi\} \, d\mathbf{Y}'$$

$$= I(\boldsymbol{\lambda})$$

I. we introduce a new, GPU-accelerated integration backend called **GASP!**

II. it calculates the symbolic integral in seconds
   $\Rightarrow$ **8 seconds** for a **81 piece** squared spline (**deg. 12**) on the SDD-Dataset
   $\Rightarrow$ up to 2 **orders of magnitudes** faster than previous WMI-Solvers

III. the result is a function of $\boldsymbol{\lambda}$
   $\Rightarrow$ **compile only once,** evaluate cheaply many times
   $\Rightarrow$ **training independent on constraint complexity**

The Integral

$$\int q(\mathbf{y}'; \boldsymbol{\lambda}) \mathbb{1}\{\mathbf{y}' \models \varphi\} \, d\mathbf{Y}'$$

$$= I(\boldsymbol{\lambda})$$

I. we introduce a new, GPU-accelerated integration backend called **GASP!**

II. it calculates the symbolic integral in seconds
   $\Rightarrow$ **8 seconds** for a **81 piece** squared spline (**deg. 12**) on the SDD-Dataset
   $\Rightarrow$ up to 2 **orders of magnitudes** faster than previous WMI-Solvers

III. the result is a function of $\boldsymbol{\lambda}$
   $\Rightarrow$ **compile only once,** evaluate cheaply many times
   $\Rightarrow$ **training independent on constraint complexity**

GASP!

$q: \quad \sum_i \lambda_i \prod_j y_j^{\alpha_{ij}}$

$\phi:$ 

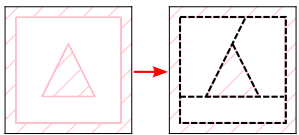
GASP!

$$q: \quad \sum_i \lambda_i \prod_j y_j^{\alpha_{ij}} \longrightarrow \left(\lambda_1 \prod_j y_j^{\alpha_{1j}}, \lambda_2 \prod_j y_j^{\alpha_{2j}}, ..., \lambda_M \prod_j y_j^{\alpha_{Mj}} \right)$$

$\phi:$ 

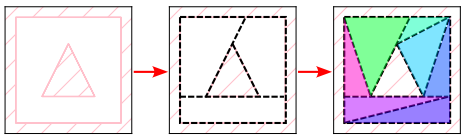
GASP!

$$q: \quad \sum_i \lambda_i \prod_j y_j^{\alpha_{ij}} \longrightarrow \left(\lambda_1 \prod_j y_j^{\alpha_{1j}}, \lambda_2 \prod_j y_j^{\alpha_{2j}}, ..., \lambda_M \prod_j y_j^{\alpha_{Mj}} \right)$$

$\phi:$ 

GASP!

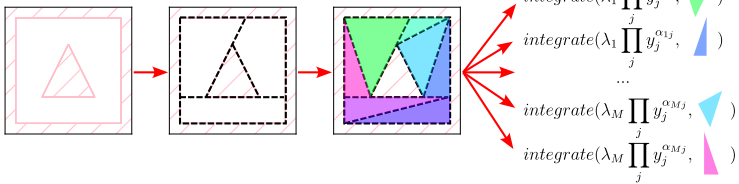
$$q: \quad \sum_i \lambda_i \prod_j y_j^{\alpha_{ij}} \longrightarrow \left(\lambda_1 \prod_j y_j^{\alpha_{1j}}, \lambda_2 \prod_j y_j^{\alpha_{2j}}, ..., \lambda_M \prod_j y_j^{\alpha_{Mj}} \right)$$

$\phi:$ 

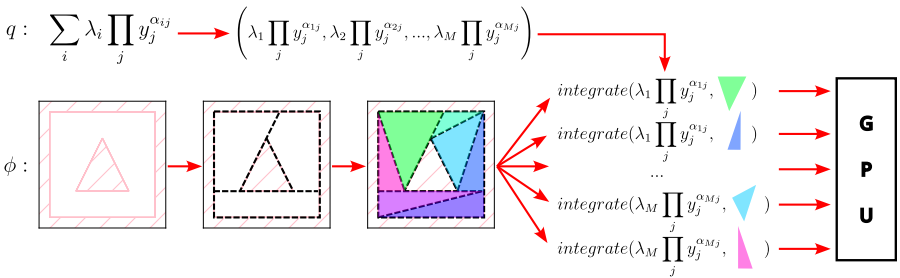
GASP!

$$q: \quad \sum_i \lambda_i \prod_j y_j^{\alpha_{ij}} \longrightarrow \left(\lambda_1 \prod_j y_j^{\alpha_{1j}}, \lambda_2 \prod_j y_j^{\alpha_{2j}}, ..., \lambda_M \prod_j y_j^{\alpha_{Mj}} \right)$$

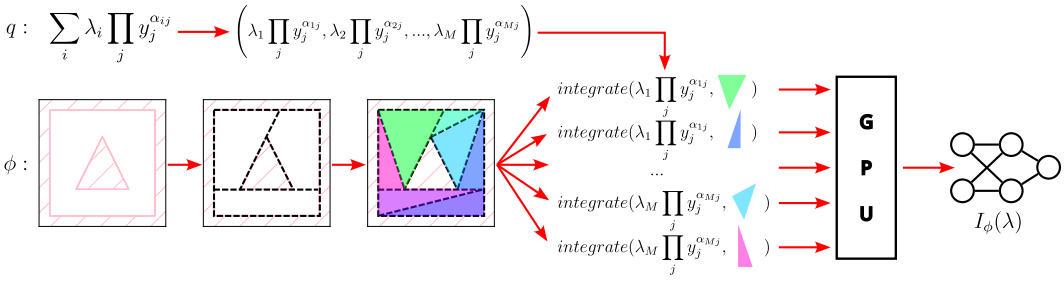
$\phi:$



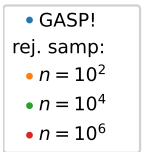
GASP!

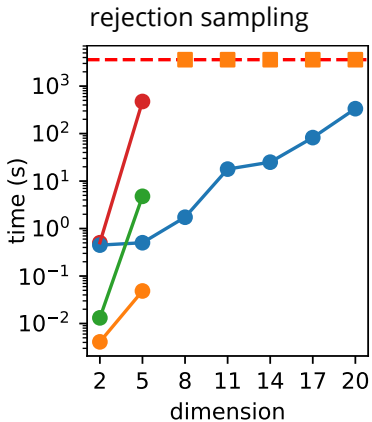

$q: \sum_i \lambda_i \prod_j y_j^{\alpha_{ij}} \rightarrow \left(\lambda_1 \prod_j y_j^{\alpha_{1j}}, \lambda_2 \prod_j y_j^{\alpha_{2j}}, ..., \lambda_M \prod_j y_j^{\alpha_{Mj}}\right)$
$\phi:$
$integrate(\lambda_1 \prod_j y_j^{\alpha_{1j}}, \triangle)$
$integrate(\lambda_1 \prod_j y_j^{\alpha_{1j}}, \triangle)$
...
$integrate(\lambda_M \prod_j y_j^{\alpha_{Mj}}, \triangle)$
$integrate(\lambda_M \prod_j y_j^{\alpha_{Mj}}, \triangle)$
G
P
U

GASP!

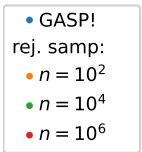

$q:\ \sum_i \lambda_i \prod_j y_j^{\alpha_{ij}} \rightarrow \left(\lambda_1 \prod_j y_j^{\alpha_{1j}}, \lambda_2 \prod_j y_j^{\alpha_{2j}}, ..., \lambda_M \prod_j y_j^{\alpha_{Mj}}\right)$
$\phi:$
$integrate(\lambda_1 \prod_j y_j^{\alpha_{1j}}, \ )$
$integrate(\lambda_1 \prod_j y_j^{\alpha_{1j}}, \ )$
...
$integrate(\lambda_M \prod_j y_j^{\alpha_{Mj}}, \ )$
$integrate(\lambda_M \prod_j y_j^{\alpha_{Mj}}, \ )$
GPU
$I_\phi(\lambda)$

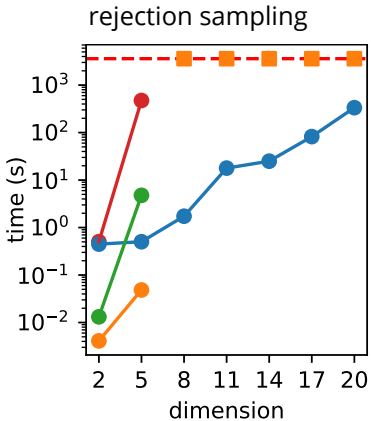
GASP!

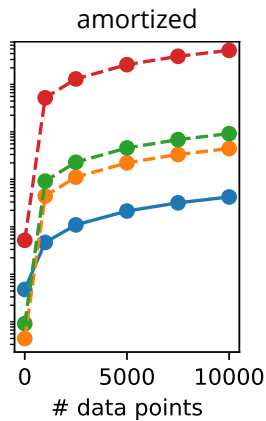

GASP!
rej. samp:
$n = 10^2$
$n = 10^4$
$n = 10^6$


rejection sampling
time (s)
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
2
5
8
11
14
17
20
dimension

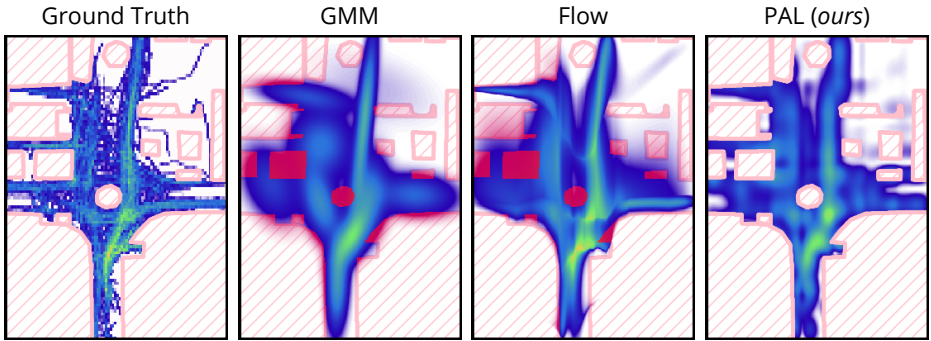
GASP!


GASP!
rej. samp:
$n = 10^2$
$n = 10^4$
$n = 10^6$


rejection sampling
time (s)
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
2 5 8 11 14 17 20
dimension

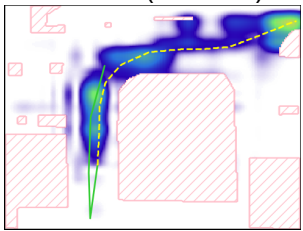

amortized
0 5000 10000
data points

$\mathbf{P(Y)}$

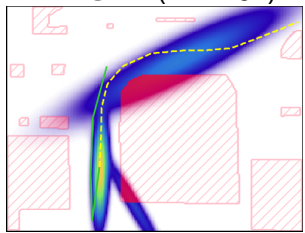


$\mathbf{P(Y|X)}$

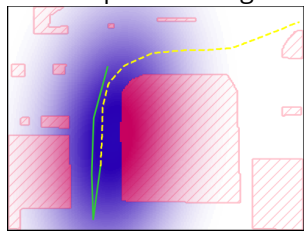
NN+PAL (14 knots)



NN+GMM ($K = 32$)



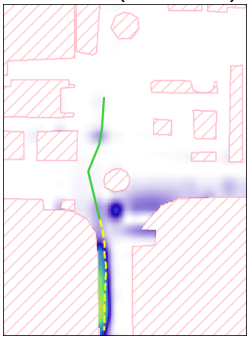
DeepSeaProblog



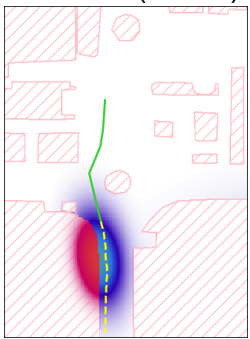
Smet et al., "Neural probabilistic logic programming in discrete-continuous domains", UAI, 2023

$\mathbf{P(Y|X)}$

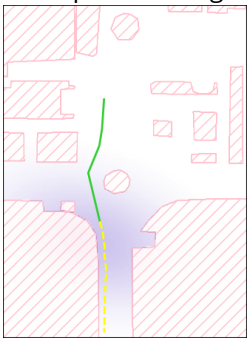
NN+PAL (14 Knots)



NN+GMM ($K=80$)



DeepSeaProblog



Smet et al., "Neural probabilistic logic programming in discrete-continuous domains", UAI, 2023

Results

		NN + PAL		NN + GMM		DeepSeaProblog
scene		10 knots	14 knots	$K{=}50$	$K{=}100$	-
1	log-like	$\mathbf{-2.08}$	-2.27	-2.64	-2.83	-3.87
	$\Pr(\neg\varphi)$	$\mathbf{0\%}$	$\mathbf{0\%}$	$\approx 21\%$	$\approx 20\%$	$\approx 49\%$

Results

		NN + PAL		NN + GMM		DeepSeaProblog
scene		10 knots	14 knots	$K{=}50$	$K{=}100$	-
1	log-like	$\mathbf{-2.08}$	-2.27	-2.64	-2.83	-3.87
	$\Pr(\neg\varphi)$	$\mathbf{0\%}$	$\mathbf{0\%}$	$\approx 21\%$	$\approx 20\%$	$\approx 49\%$
2	log-like	-2.23	$\mathbf{-2.09}$	-2.39	-2.42	-3.61
	$\Pr(\neg\varphi)$	$\mathbf{0\%}$	$\mathbf{0\%}$	$\approx 15\%$	$\approx 14\%$	$\approx 36\%$

Results

		NN + PAL		NN + GMM		DeepSeaProblog
scene		10 knots	14 knots	$K{=}50$	$K{=}100$	-
1	log-like	$\mathbf{-2.08}$	-2.27	-2.64	-2.83	-3.87
	Pr($\neg\varphi$)	$\mathbf{0\%}$	$\mathbf{0\%}$	$\approx 21\%$	$\approx 20\%$	$\approx 49\%$
2	log-like	-2.23	$\mathbf{-2.09}$	-2.39	-2.42	-3.61
	Pr($\neg\varphi$)	$\mathbf{0\%}$	$\mathbf{0\%}$	$\approx 15\%$	$\approx 14\%$	$\approx 36\%$

are we done?

Limitations & Future Work

I. Scaling to high-dimensional problems
   $\Rightarrow$ **hybridize with approximations**

II. Parallelize sampling
   $\Rightarrow$ **Exact integral can help with inverse-transform sampling**

III. Maximum a Posteriori (MAP)
   $\Rightarrow$ **Picking the right piece to simplify the problem**

IV. Generalize the constraints to curves (e.g. NURBS)
   $\Rightarrow$ **go beyond** $\mathrm{SMT}(\mathcal{LRA})$

Chin and Sukumar, "An efficient method to integrate polynomials over polytopes and curved solids", Computer Aided Geometric Design, 2020

Limitations & Future Work

I. Scaling to high-dimensional problems

$\Rightarrow$ **hybridize with approximations**

II. Parallelize sampling

$\Rightarrow$ **Exact integral can help with inverse-transform sampling**

III. Maximum a Posteriori (MAP)

$\Rightarrow$ **Picking the right piece to simplify the problem**

IV. Generalize the constraints to curves (e.g. NURBS)

$\Rightarrow$ **go beyond** $\mathrm{SMT}(\mathcal{LRA})$

Chin and Sukumar, "An efficient method to integrate polynomials over polytopes and curved solids", Computer Aided Geometric Design, 2020

Limitations & Future Work

I. Scaling to high-dimensional problems

$\Rightarrow$ **hybridize with approximations**

II. Parallelize sampling

$\Rightarrow$ **Exact integral can help with inverse-transform sampling**

III. Maximum a Posteriori (MAP)

$\Rightarrow$ **Picking the right piece to simplify the problem**

IV. Generalize the constraints to curves (e.g. NURBS)

$\Rightarrow$ **go beyond** $\mathrm{SMT}(\mathcal{LRA})$

Chin and Sukumar, "An efficient method to integrate polynomials over polytopes and curved solids", Computer Aided Geometric Design, 2020

Limitations & Future Work

I. Scaling to high-dimensional problems

$\Rightarrow$ **hybridize with approximations**

II. Parallelize sampling

$\Rightarrow$ **Exact integral can help with inverse-transform sampling**

III. Maximum a Posteriori (MAP)

$\Rightarrow$ **Picking the right piece to simplify the problem**

IV. Generalize the constraints to curves (e.g. NURBS)

$\Rightarrow$ **go beyond** $\mathrm{SMT}(\mathcal{LRA})$

Chin and Sukumar, "An efficient method to integrate polynomials over polytopes and curved solids", Computer Aided Geometric Design, 2020

Limitations & Future Work

I. Scaling to high-dimensional problems

$\Rightarrow$ **hybridize with approximations**

II. Parallelize sampling

$\Rightarrow$ **Exact integral can help with inverse-transform sampling**

III. Maximum a Posteriori (MAP)

$\Rightarrow$ **Picking the right piece to simplify the problem**

IV. Generalize the constraints to curves (e.g. NURBS)

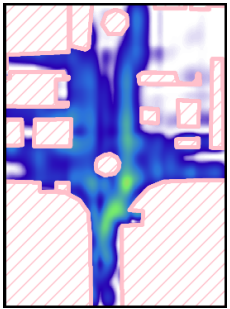
$\Rightarrow$ **go beyond** $\mathrm{SMT}(\mathcal{LRA})$

Chin and Sukumar, "An efficient method to integrate polynomials over polytopes and curved solids", Computer Aided Geometric Design, 2020

Takeaway

PAL combines:

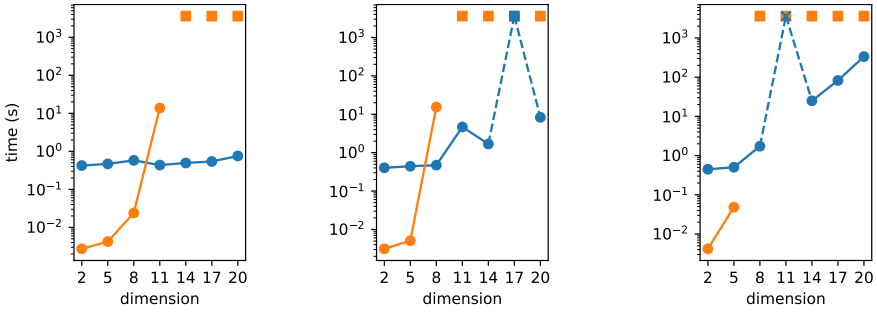
- **I.** **expressive** distribution
- **II.** **rich** and **flexible** constraints
- **III.** **fast** and **amortized** integration
- **=>** competitive on real-world data



PAPER

Code

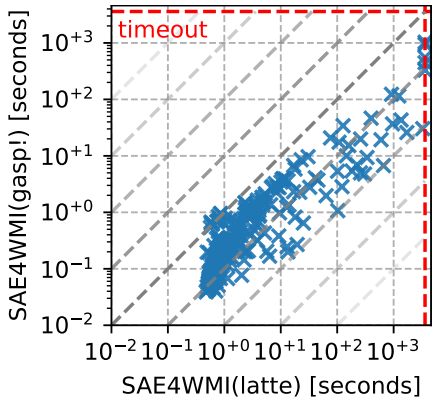
Additional: Numerical Approximation



The Integral - Detail Decomposition

$$
\begin{aligned}
I(\boldsymbol{\lambda}) &= \int q(\mathbf{y}'; \boldsymbol{\lambda}) \qquad\qquad \mathbb{1}\{\mathbf{y}' \models \varphi\}\ d\mathbf{Y}' \\
&= \int \left(\sum_{\boldsymbol{\alpha}} \lambda_{\boldsymbol{\alpha}} \mathbf{y}'^{\boldsymbol{\alpha}} \right) \qquad \mathbb{1}\{\mathbf{y}' \models \varphi\}\ d\mathbf{Y}' \\
&= \sum_{\boldsymbol{\alpha}} \left(\int \lambda_{\boldsymbol{\alpha}} \mathbf{y}'^{\boldsymbol{\alpha}} \qquad \mathbb{1}\{\mathbf{y}' \models \varphi\}\ d\mathbf{Y}' \right)
\end{aligned}
\tag{1}
$$

Benchmark: GASP! vs LATTE



Benchmark: GASP! vs Rejection Sampling

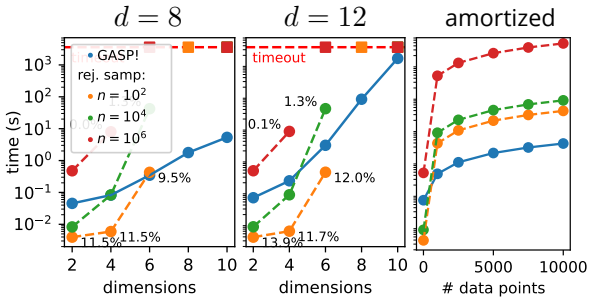


Figure: Left and center: runtime of GASP! and rejection sampling (using $10^{\{2,4,6\}}$ samples) on integrals over 4 random simplices for increasing dimensions. We report the relative error ($\%$) for rejection sampling. Right: Our amortization speed-up compared to rejection-sampling in 2-dimensions for the 12-degree polynomial.