Safety-Critical Machine Learning: Development & Testing

Aman Sinha
August 17, 2020

Motivation

- We are starting to apply machine learning to high-stakes decision-making



- The standard paradigms of ML aren't enough in safety-critical applications
  - Development: minimize average loss over a nominal training dataset
  - Testing: check average performance over a test dataset

- New paradigms for handling uncertainty
  - Development: build **robustness** against uncertainties
  - Testing: quantify **risk** (likelihood and severity) of failures

Key ideas of this talk

Robustness

Build models with guaranteed performance over uncertainty sets

Risk

Find failure modes and quantify the probability of failure

Key ideas of this talk

Robustness

Build models with guaranteed performance over uncertainty sets

$$\min_{\theta \in \Theta} \max_{Q \in \mathcal{P}} \mathbb{E}_Q[f(\theta; X)]$$

Risk

Find failure modes and quantify the probability of failure

Key ideas of this talk

Robustness

Build models with guaranteed performance over uncertainty sets

$$\min_{\theta \in \Theta} \max_{Q \in \mathcal{P}} \mathbb{E}_Q[f(\theta; X)]$$

- With small $\mathcal{P}$, can we solve this quickly?
- What about when $\mathcal{P}$ is large/unknown?

Risk

Find failure modes and quantify the probability of failure

Key ideas of this talk

Robustness

Build models with guaranteed performance over uncertainty sets

$$\min_{\theta \in \Theta} \max_{Q \in \mathcal{P}} \mathbb{E}_Q[f(\theta; X)]$$

- With small $\mathcal{P}$, can we solve this quickly?
- What about when $\mathcal{P}$ is large/unknown?

Risk

Find failure modes and quantify the probability of failure

$$\mathbb{P}_0(f(X) < \gamma)$$

Key ideas of this talk

Robustness

Build models with guaranteed performance over uncertainty sets

$$\min_{\theta \in \Theta} \max_{Q \in \mathcal{P}} \mathbb{E}_Q[f(\theta; X)]$$

- With small $\mathcal{P}$, can we solve this quickly?
- What about when $\mathcal{P}$ is large/unknown?

Risk

Find failure modes and quantify the probability of failure

$$\mathbb{P}_0(f(X) < \gamma)$$

- Why is this the right problem?
- How do we solve it quickly?

Key ideas of this talk

Robustness

Build models with guaranteed performance over uncertainty sets

$$\min_{\theta \in \Theta} \max_{Q \in \mathcal{P}} \mathbb{E}_Q[f(\theta; X)]$$

- With small $\mathcal{P}$, can we solve this quickly?
- What about when $\mathcal{P}$ is large/unknown?

Risk

Find failure modes and quantify the probability of failure

$$\mathbb{P}_0(f(X) < \gamma)$$

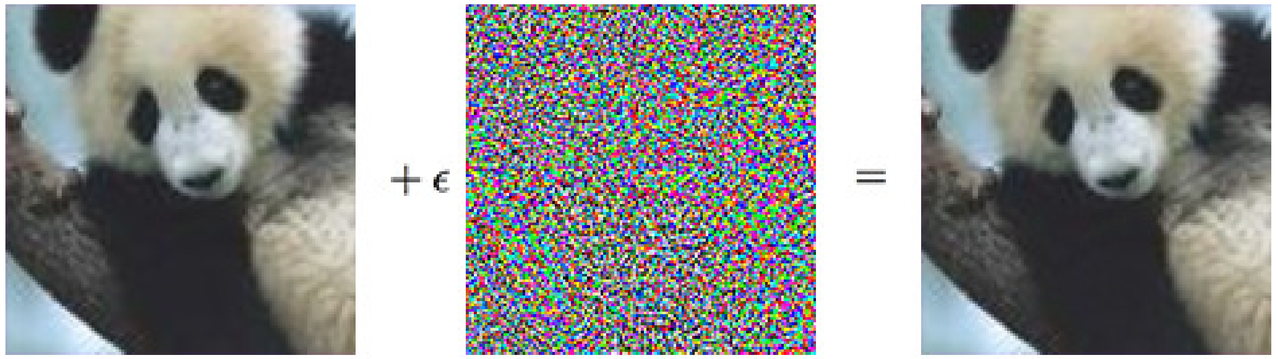
- Why is this the right problem?
- How do we solve it quickly?

Certifiable robustness against adversarial attacks

Sinha*, Namkoong*, Duchi. ICLR 2018.
Sinha*, Namkoong*, Volpi, Duchi. Under review.



Certifiable robustness against adversarial attacks



[Goodfellow et al. 2015]

[Athalye et al. 2017]

We want to increase the robustness of ML systems to adversarial attacks (small $\mathcal{P}$)

Current approaches

- Adversarial training heuristics: fast but no theoretical guarantees of robustness
  - Goodfellow et al. 2015, Kurakin et al. 2016, Papernot et al. 2016, He et al. 2017, Carlini & Wagner 2017, Tramer et al. 2017, Madry et al. 2018, etc.

- Formal verification: rigorous guarantees but slow
  - Huang et al. 2017, Katz et al. 2017, Kolter & Wong 2017, Tjeng & Tedrake 2017, Raghunathan et al. 2018

Our goal: balance efficiency with robustness guarantees

Our work: principled adversarial training

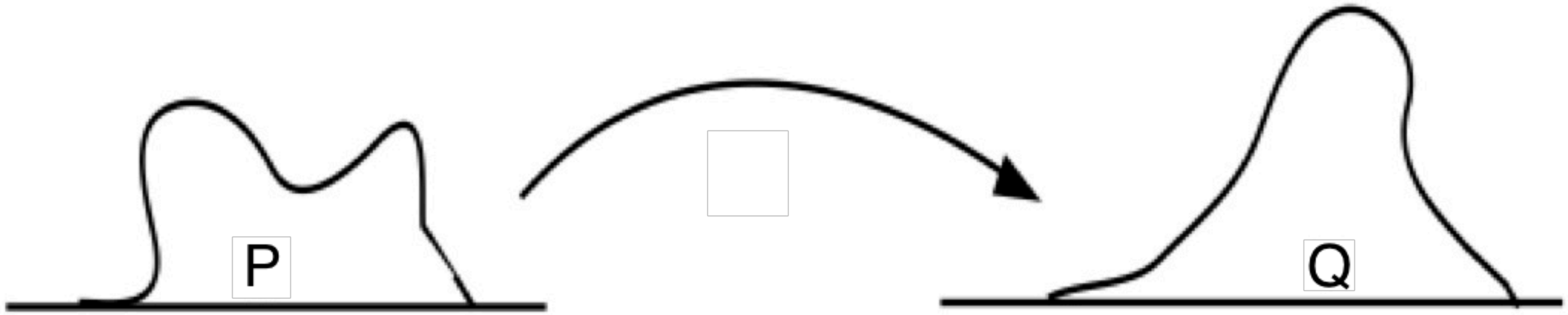
- Setup: model/network weights $\theta \in \Theta$, feature vector X, label Y, and loss $\ell(\theta; X, Y)$

Overall idea: replace $\ell(\theta; X, Y)$ with robust surrogate $\phi_\gamma(\theta; X, Y)$

- For moderate levels of desired robustness and smooth losses ℓ:
  - Provably fast convergence, 5-10x as fast as ERM
  - Statistical guarantees for performance on (perturbations to) the test set

Distributionally robust optimization (DRO)

- Goal: robustness to perturbations in a Wasserstein ball
- Generally intractable for arbitrary ρ



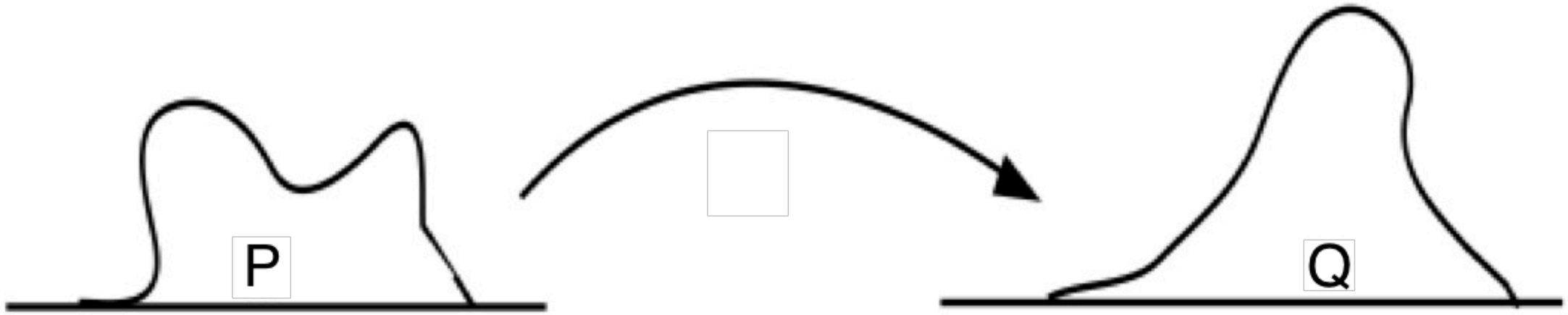
[Esfahani & Kuhn 2015; Shafieezadeh-Abadeh et al. 2015; Blanchet et al. 2016, Lee & Raginsky 2017]

Distributionally robust optimization (DRO)

$$\underset{\theta \in \Theta}{\text{minimize}} \ \mathbb{E}_{P_0}[\ell(\theta; X, Y)]$$

- Goal: robustness to perturbations in a Wasserstein ball

- Generally intractable for arbitrary ρ

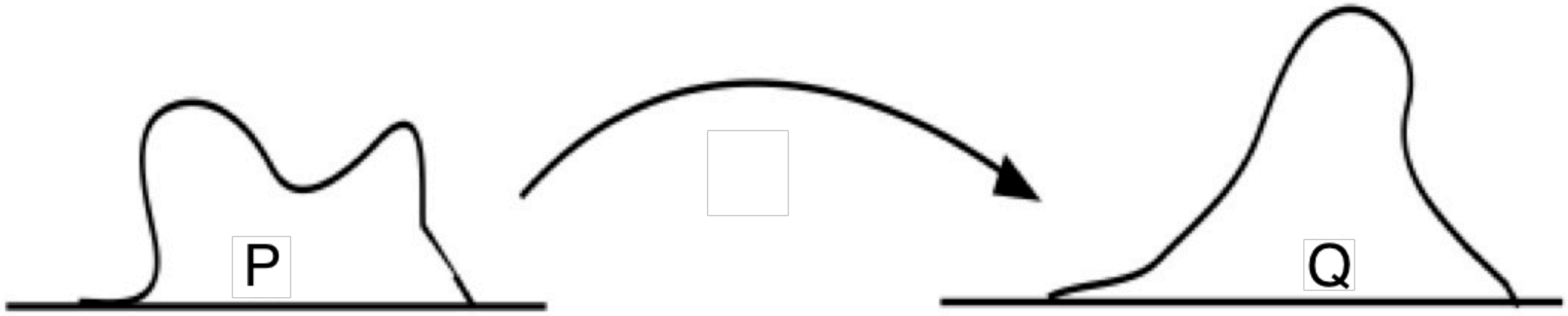


[Esfahani & Kuhn 2015; Shafieezadeh-Abadeh et al. 2015; Blanchet et al. 2016, Lee & Raginsky 2017]

Distributionally robust optimization (DRO)

$$\underset{\theta \in \Theta}{\text{minimize}} \quad \max_{Q} \{\mathbb{E}_Q[\ell(\theta; X, Y)] : D_c(Q, P_0) \leq \rho\}$$

- Goal: robustness to perturbations in a Wasserstein ball
- Generally intractable for arbitrary ρ



[Esfahani & Kuhn 2015; Shafieezadeh-Abadeh et al. 2015; Blanchet et al. 2016, Lee & Raginsky 2017]

Distributionally robust optimization (DRO)

- Lagrangian relaxation and its dual formulation
  - More robustness ↔ larger ρ ↔ smaller γ

$$\underset{\theta\in\Theta}{\text{minimize}} \ \max_{Q} \left\{ \mathbb{E}_Q[\ell(\theta; X, Y)] - \underbrace{\gamma D_c(Q, P_0)}_{\text{penalty}} \right\} =$$

$$\underset{\theta\in\Theta}{\text{minimize}} \ \mathbb{E}_{P_0}[\phi_\gamma(\theta; X, Y)]$$

$$\text{where} \ \ \phi_\gamma(\theta; x, y) := \max_{x'\in\mathcal{X}} \left\{ \ell(\theta; x', y) - \underbrace{\gamma \|x' - x\|^2}_{\text{penalty}} \right\}$$

- Compare to ERM: $\text{minimize}_{\theta\in\Theta} \ \mathbb{E}_{P_0}[\ell(\theta; X, Y)]$

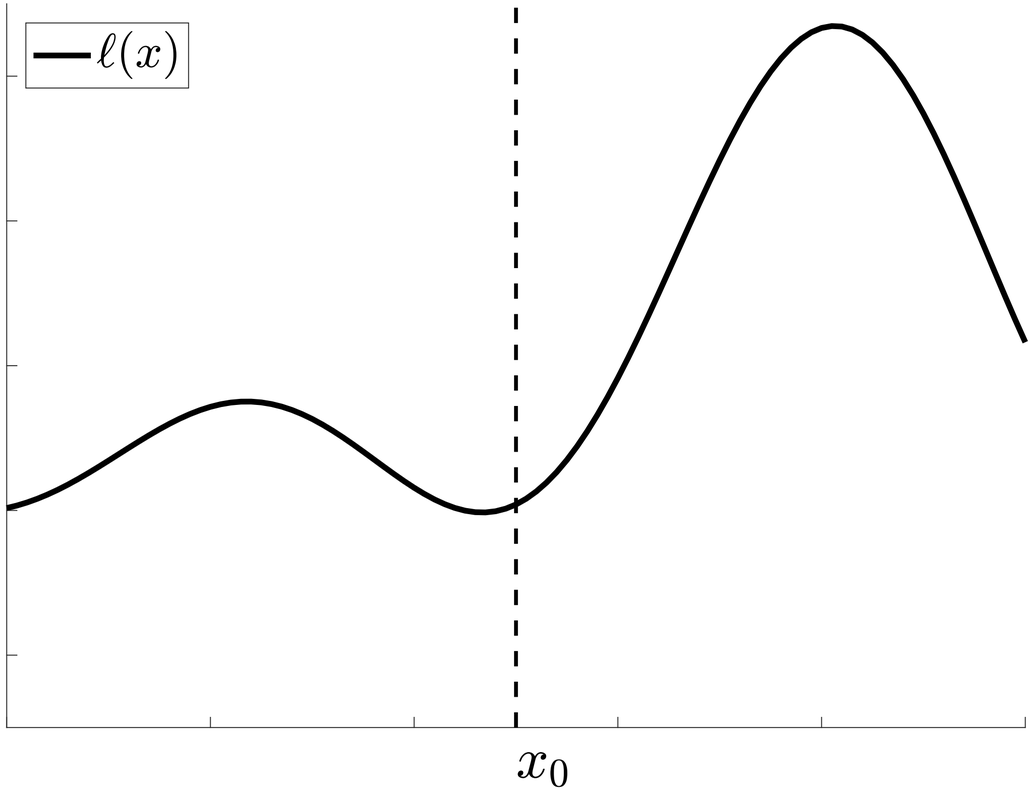
[Blanchet et al. 2016]

Solving the optimization problem

$$\phi_\gamma(\theta; x_0, y_0) := \max_{x \in \mathcal{X}} \left\{ \ell(\theta; x, y_0) - \gamma \|x - x_0\|^2 \right\}$$

Key insight: $(x, y) \mapsto \ell(\theta; x, y) - \gamma \|x - x_0\|^2$ is strongly concave for **smooth** ℓ and large enough γ

- Curvature in $\|\cdot\|^2$ dwarfs out non-concavity of $\ell(\theta; \cdot)$

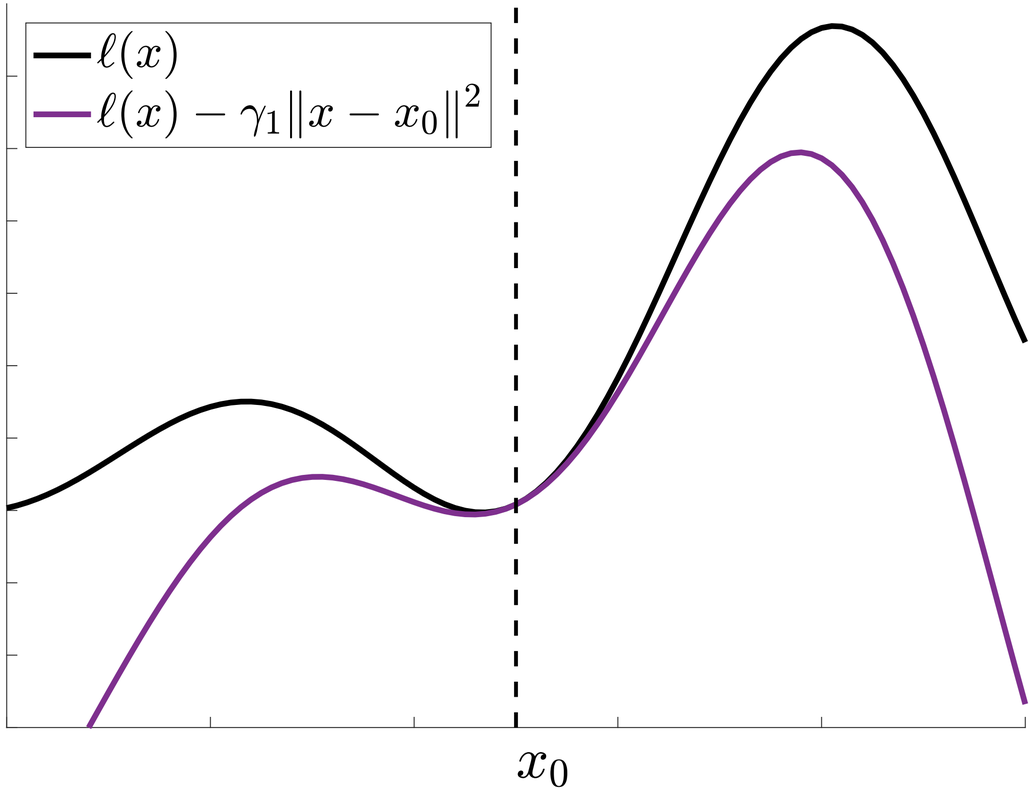


Solving the optimization problem

$$\phi_\gamma(\theta; x_0, y_0) := \max_{x \in \mathcal{X}} \left\{ \ell(\theta; x, y_0) - \gamma \|x - x_0\|^2 \right\}$$

Key insight: $(x, y) \mapsto \ell(\theta; x, y) - \gamma \|x - x_0\|^2$ is strongly concave for **smooth** ℓ and large enough γ

- Curvature in $\|\cdot\|^2$ dwarfs out non-concavity of $\ell(\theta; \cdot)$

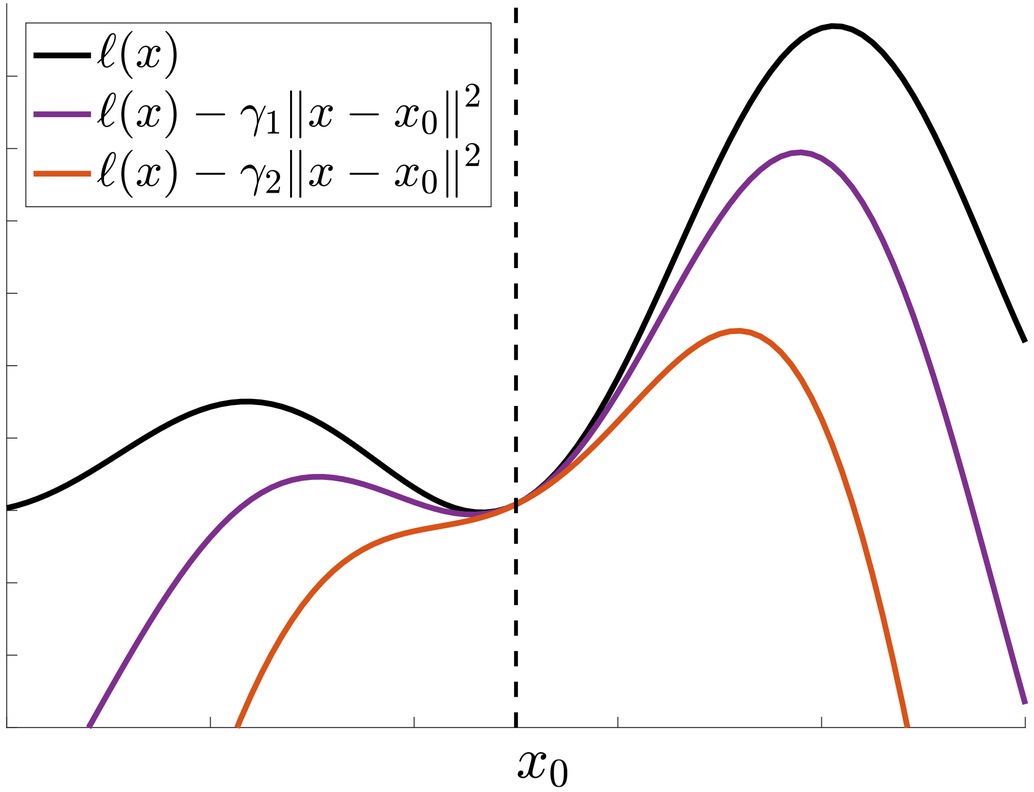


Solving the optimization problem

$$\phi_\gamma(\theta; x_0, y_0) := \max_{x \in \mathcal{X}} \left\{ \ell(\theta; x, y_0) - \gamma \|x - x_0\|^2 \right\}$$

Key insight: $(x, y) \mapsto \ell(\theta; x, y) - \gamma \|x - x_0\|^2$ is strongly concave for **smooth** ℓ and large enough γ

- Curvature in $\|\cdot\|^2$ dwarfs out non-concavity of $\ell(\theta; \cdot)$

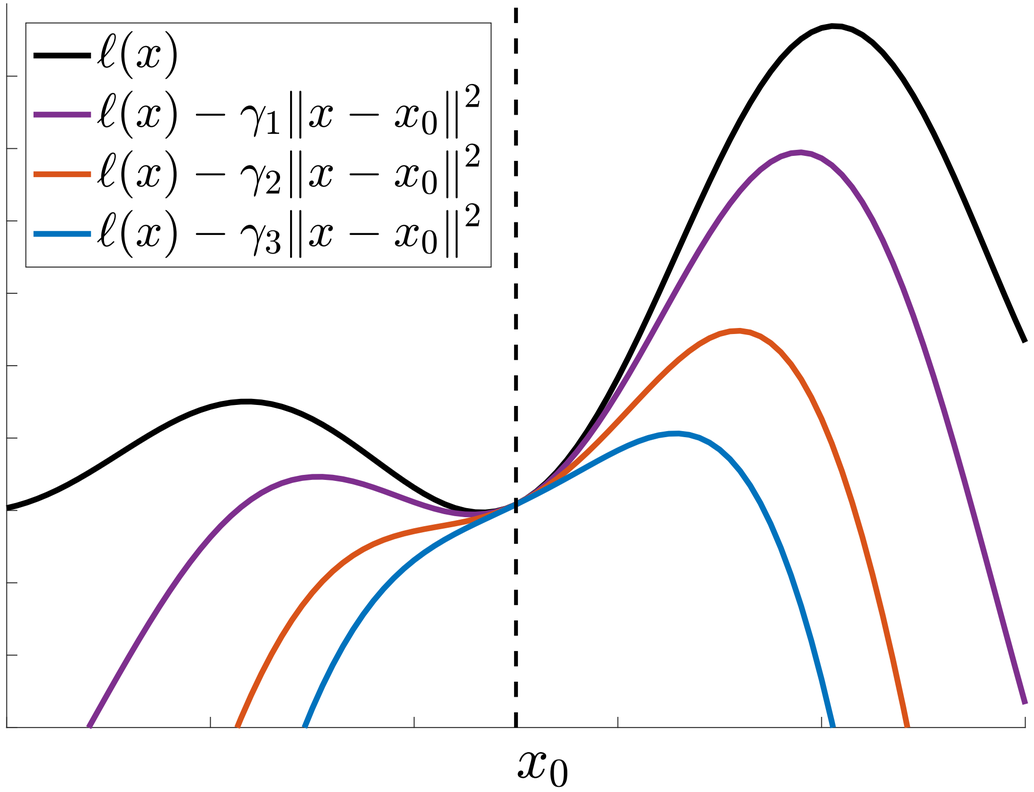


Solving the optimization problem

$$\phi_\gamma(\theta; x_0, y_0) := \max_{x \in \mathcal{X}} \left\{ \ell(\theta; x, y_0) - \gamma \|x - x_0\|^2 \right\}$$

Key insight: $(x, y) \mapsto \ell(\theta; x, y) - \gamma \|x - x_0\|^2$ is strongly concave for **smooth** ℓ and large enough γ

- Curvature in $\|\cdot\|^2$ dwarfs out non-concavity of $\ell(\theta; \cdot)$

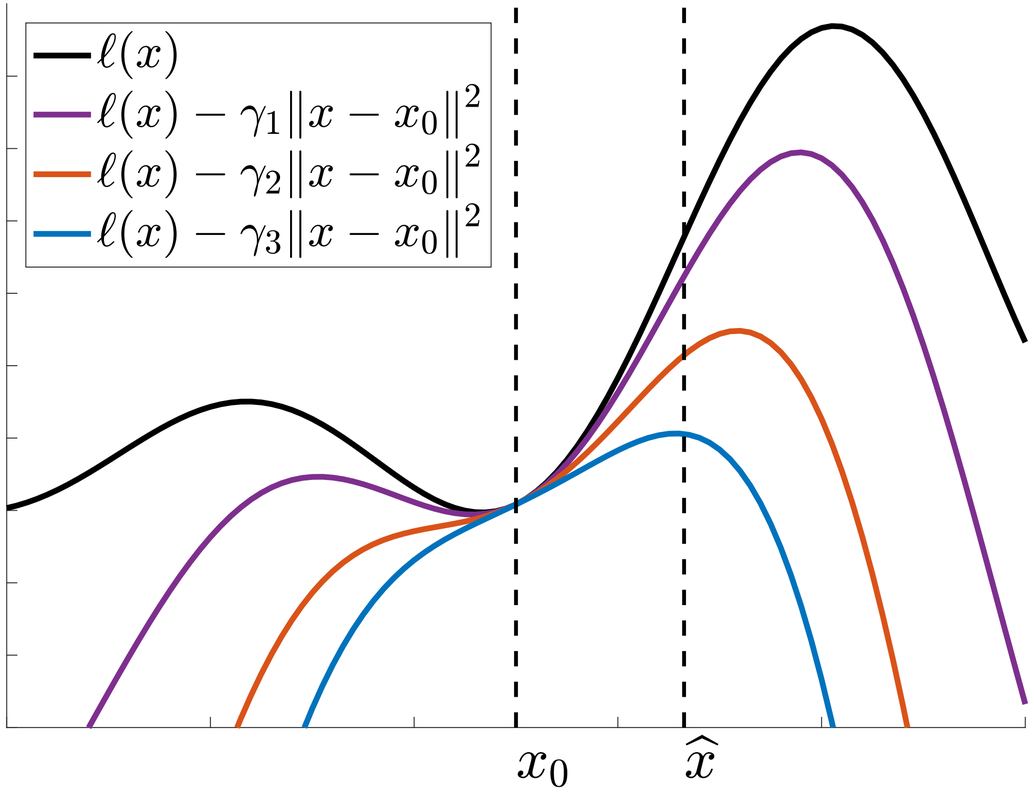


Solving the optimization problem

$$\phi_\gamma(\theta; x_0, y_0) := \max_{x \in \mathcal{X}} \left\{ \ell(\theta; x, y_0) - \gamma \|x - x_0\|^2 \right\}$$

Key insight: $(x, y) \mapsto \ell(\theta; x, y) - \gamma \|x - x_0\|^2$ is strongly concave for **smooth** ℓ and large enough γ

- Curvature in $\|\cdot\|^2$ dwarfs out non-concavity of $\ell(\theta; \cdot)$

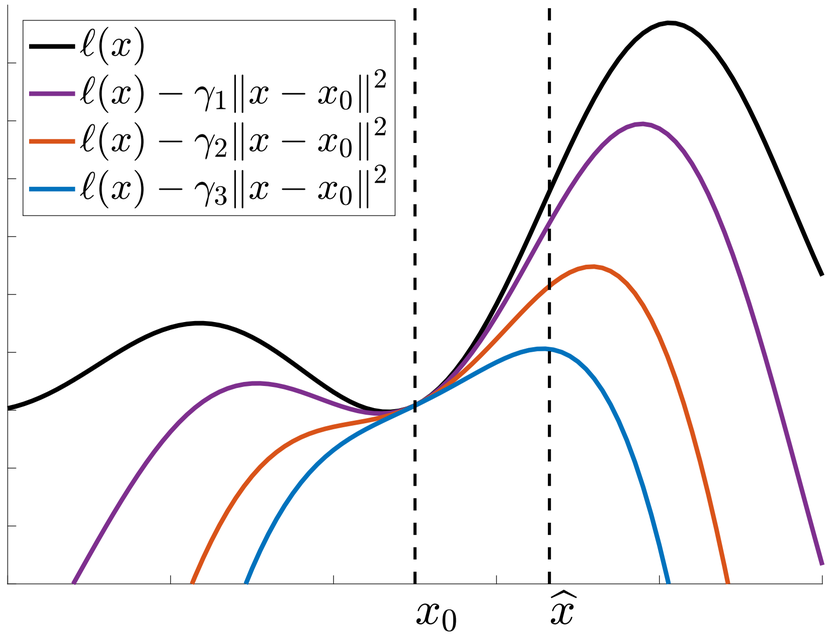


Solving the optimization problem

$$\phi_\gamma(\theta; x_0, y_0) := \max_{x \in \mathcal{X}} \left\{ \ell(\theta; x, y_0) - \gamma \|x - x_0\|^2 \right\}$$

Key insight: $(x, y) \mapsto \ell(\theta; x, y) - \gamma \|x - x_0\|^2$ is strongly concave for **smooth** ℓ and large enough γ

- Curvature in $\|\cdot\|^2$ dwarfs out non-concavity of $\ell(\theta; \cdot)$



Deep nets with smooth activations (ELUs, sigmoid, etc.) are smooth

Optimization guarantees

Algorithm: SGD for $\min_\theta \mathbb{E}_{P_0}[\phi_\gamma(\theta; X, Y)]$

- Sample $(x^t, y^t) \sim P_0$
- Compute adversarial example:
  (approximate) maximizer $\widehat{x}^t$ of $\ell(\theta^t; x, y^t) - \gamma\|x - x^t\|^2$
- $\theta^{t+1} \leftarrow \theta^t - \alpha\nabla_\theta \ell(\theta^t; \widehat{x}^t, y^t)$

- For large enough γ we can compute $\widehat{x}^t$ in 10-20 gradient ascent steps

- Theorem: converges at standard nonconvex-SGD rate

Certificate of robustness

- Algorithm **generalizes**: we learn to prevent attacks on the **test** set

- $\theta_{\mathrm{WRM}} =$ output of Algorithm, $\mathfrak{Comp}_n =$ size of Θ, $C =$ problem-dependent constant, $\widehat{P}_n =$ empirical training distribution

Theorem (Robustness Certificate)

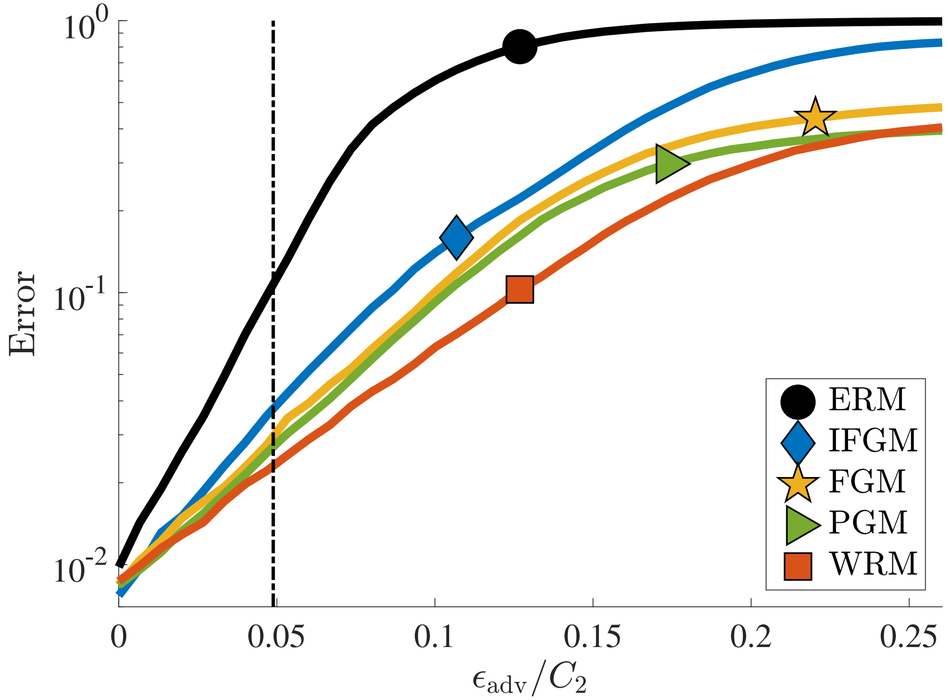
With high probability, for any $\rho \geq 0$

$$\max_{Q: D_c(Q, P_0) \leq \rho} \mathbb{E}_Q[\ell(\theta_{\mathrm{WRM}}; X, Y)] \leq \gamma\rho + \mathbb{E}_{\widehat{P}_n}[\phi_\gamma(\theta_{\mathrm{WRM}}; X, Y)] + C\frac{\mathfrak{Comp}_n}{\sqrt{n}}$$

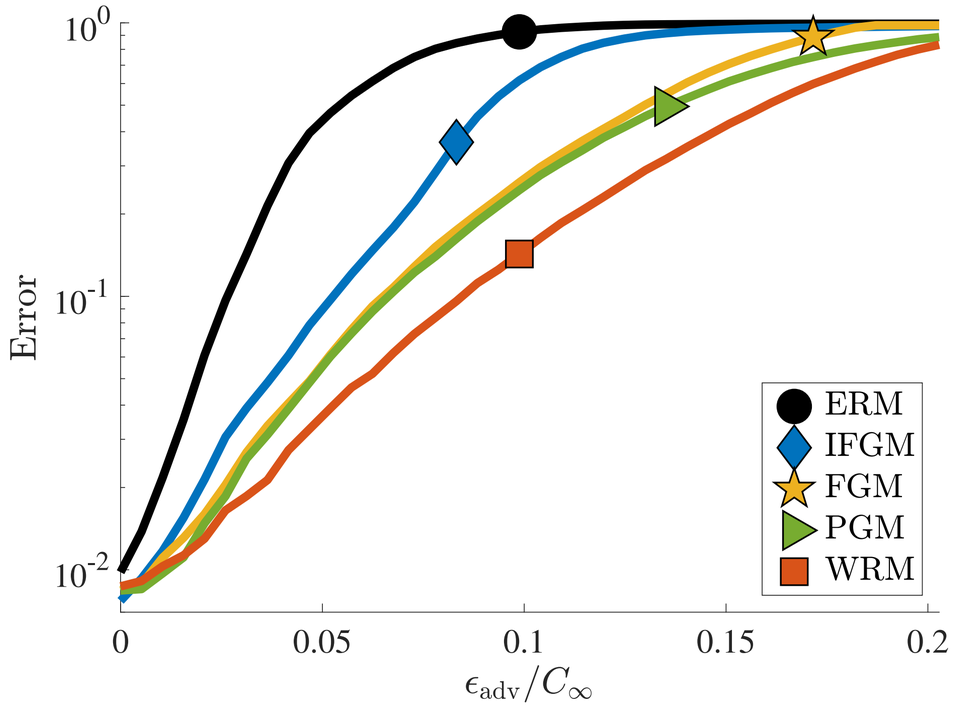
MNIST digit classification

MNIST classification

- Compare our method (WRM) with fast-gradient (FGM), iterated FGM (IFGM), and projected gradient method (PGM)



Test error vs. ϵ_{adv} for PGM $\|\cdot\|_2$ attack

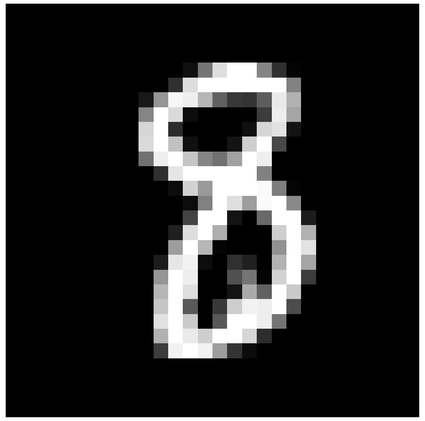


Test error vs. ϵ_{adv} for PGM $\|\cdot\|_\infty$ attack

[Goodfellow et al. 2015, Kurakin et al. 2016, Madry et al. 2017]

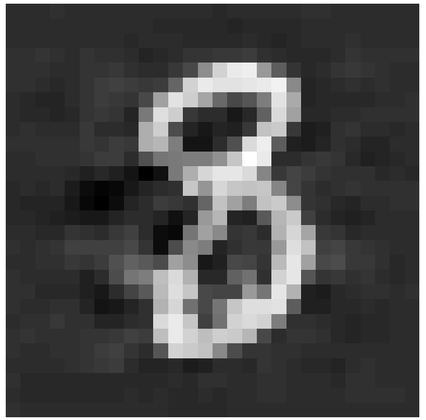
When the model misclassifies

- Minimum perturbation forcing WRM to misclassify is perceptible

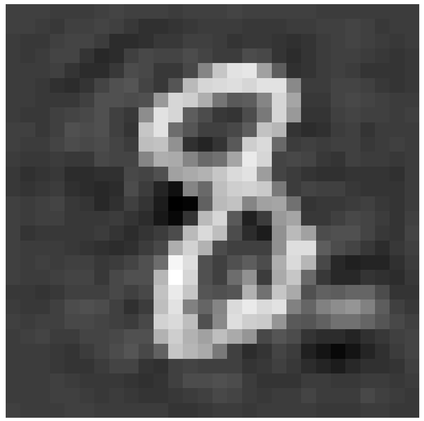


Original

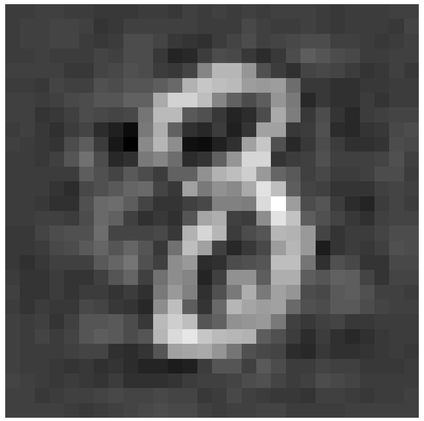
ERM



FGM



IFGM



PGM

WRM

Key ideas of this talk

Robustness

Build models with guaranteed performance over uncertainty sets

$$\min_{\theta \in \Theta} \max_{Q \in \mathcal{P}} \mathbb{E}_Q[f(\theta; X)]$$

- With small $\mathcal{P}$, can we solve this quickly?
- What about when $\mathcal{P}$ is large/unknown?

Risk

Find failure modes and quantify the probability of failure

$$\mathbb{P}_0(f(X) < \gamma)$$

- Why is this the right problem?
- How do we solve it quickly?

Balancing safety & performance in high-uncertainty regimes

Sinha*, O'Kelly*, Zheng*, Mangharam, Duchi, Tedrake. ICML 2020.

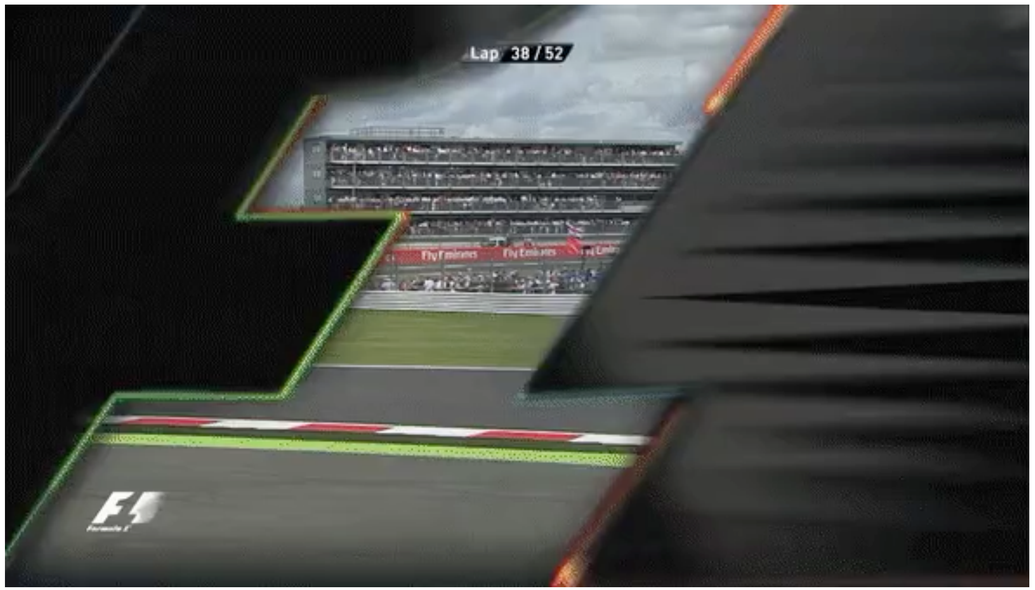




Balancing safety & performance in high-uncertainty regimes

- When $\mathcal{P}$ is large/unknown, balance is critical:
  - Conservativeness leads to poor performance
  - Aggressiveness is dangerous

- Autonomous racing is an extreme limit of autonomous driving
  - Strategies are secret
  - Crashing is expensive/dangerous (and makes winning hard)



Robust reinforcement learning

$$\text{minimize} \sum_t \lambda^t \mathbb{E}[c(o(t))]$$

Robust reinforcement learning

$$\text{minimize} \max_{P_{sa} \in \mathcal{P}} \sum_t \lambda^t \mathbb{E}[c(o(t))]$$

If we knew the opponent's behavior,
we wouldn't need the inner max

- State-action transition probabilities P_{sa}, observations o, discount factor λ, cost c

Robust reinforcement learning

$$\text{minimize} \max_{P_{sa} \in \mathcal{P}} \sum_t \lambda^t \mathbb{E}[c(o(t))]$$

If we knew the opponent's behavior,
we wouldn't need the inner max

- State-action transition probabilities P_{sa}, observations o, discount factor λ, cost c

- Overall idea: learn a useful parametrization for $\mathcal{P}$ and then proceed as before
  - Offline population synthesis: self-play to learn $\mathcal{P}$, a population of good racers
  - Online robust planning: robust belief-space planning against an opponent

Related work

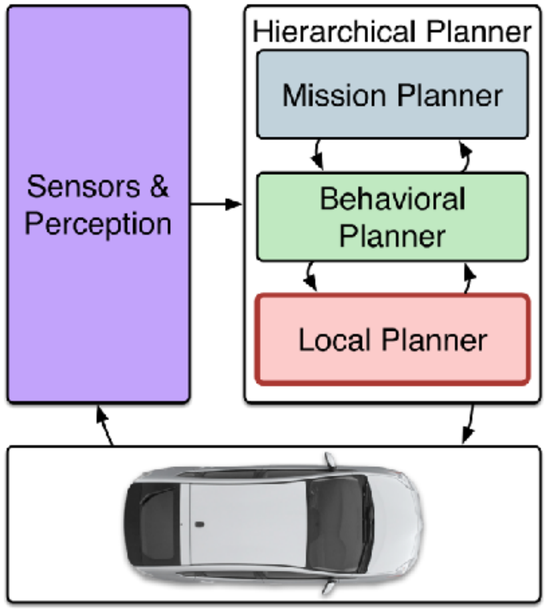
- Robust RL/control
  - Robust MDP [Nilim & El Ghaoui 2005]
  - POMDP [Kaelbling et al. 1998]
  - Adversarial RL [Mandlekar et al. 2017, Pinto et al. 2017]

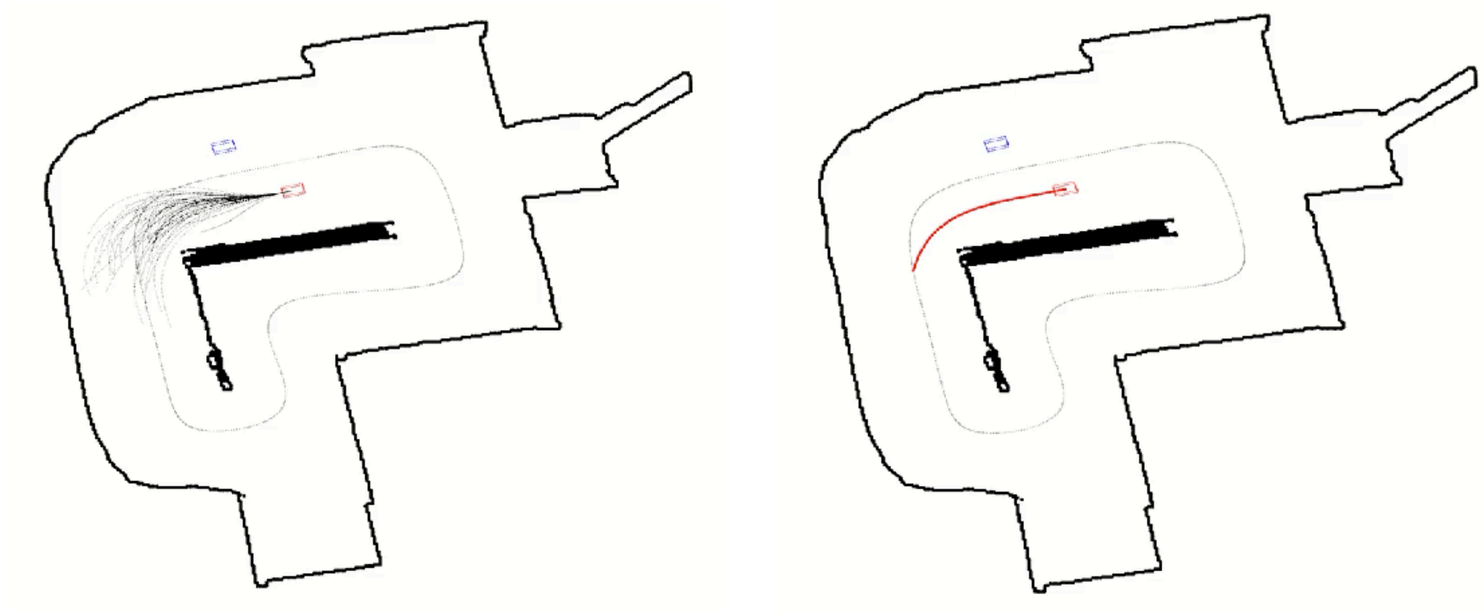
- Belief-space planning [Van Den Berg et al. 2011, Galceran et al. 2015, Kochenderfer 2015]

- DRO [Namkoong & Duchi 2017]

Offline population synthesis

Goal: generate a diverse set of competitive agent behaviors





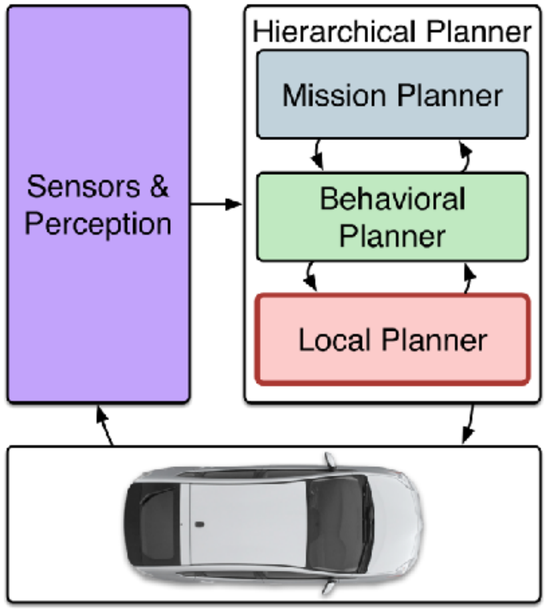
θ x

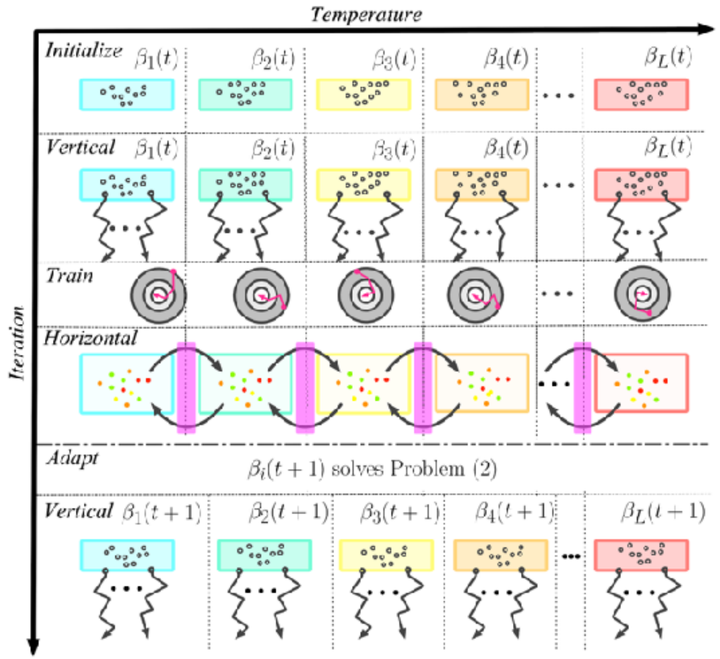
Policy parametrization

- Goal generator: neural net (IAF) weights θ
- Goal evaluator: non-differentiable cost weights x

Offline population synthesis

Goal: generate a diverse set of competitive agent behaviors





Policy parametrization

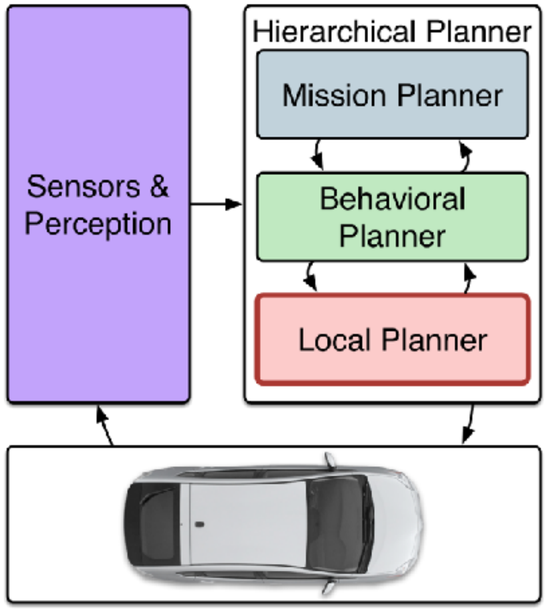
- Goal generator: neural net (IAF) weights θ
- Goal evaluator: non-differentiable cost weights x

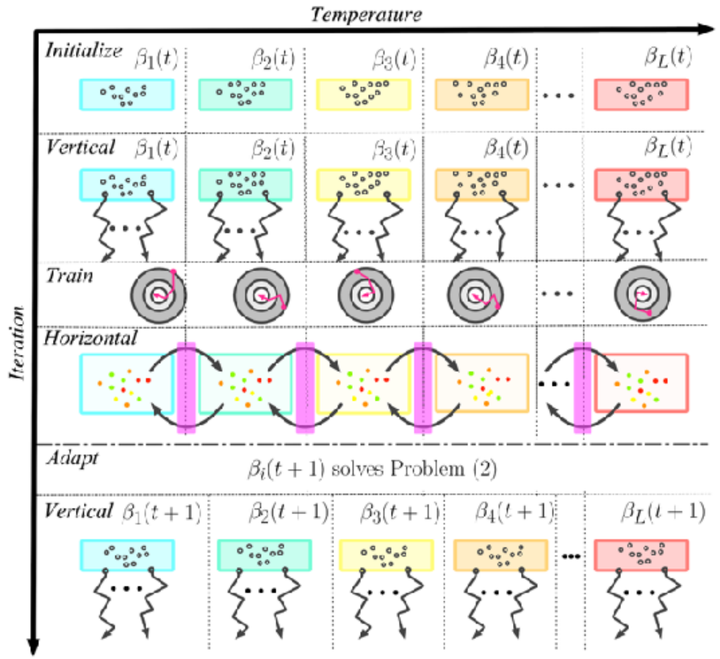
Search algorithm

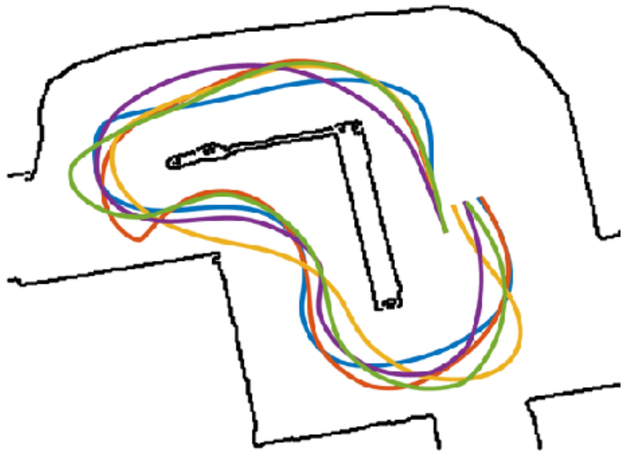
- Employs self-play to generate competitive agents

Offline population synthesis

Goal: generate a diverse set of competitive agent behaviors







Policy parametrization

- Goal generator: neural net (IAF) weights θ
- Goal evaluator: non-differentiable cost weights x

Search algorithm

- Employs self-play to generate competitive agents

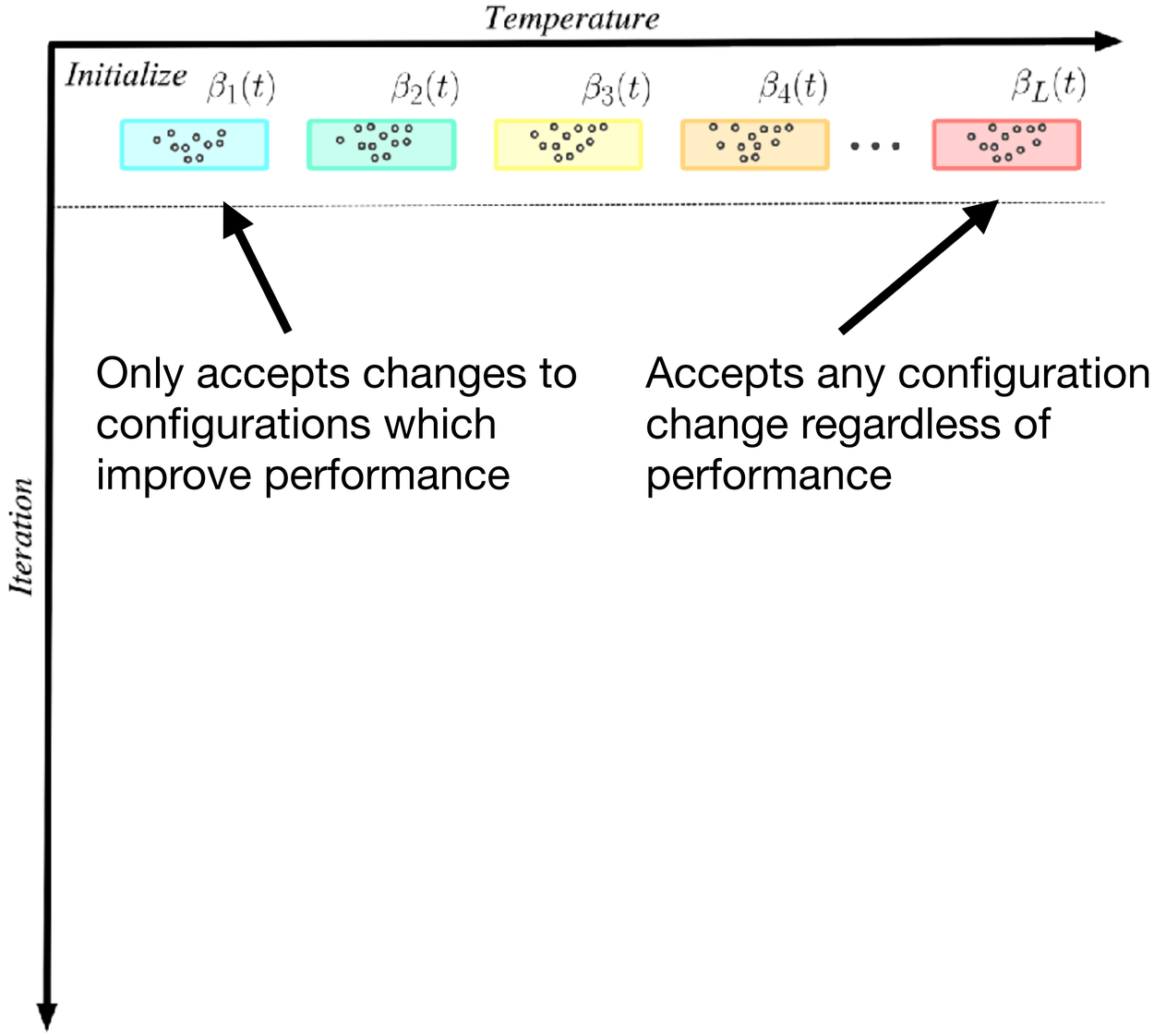
Diverse population of agents

- Described by their parameters (x, θ)

Step 1: initialize populations

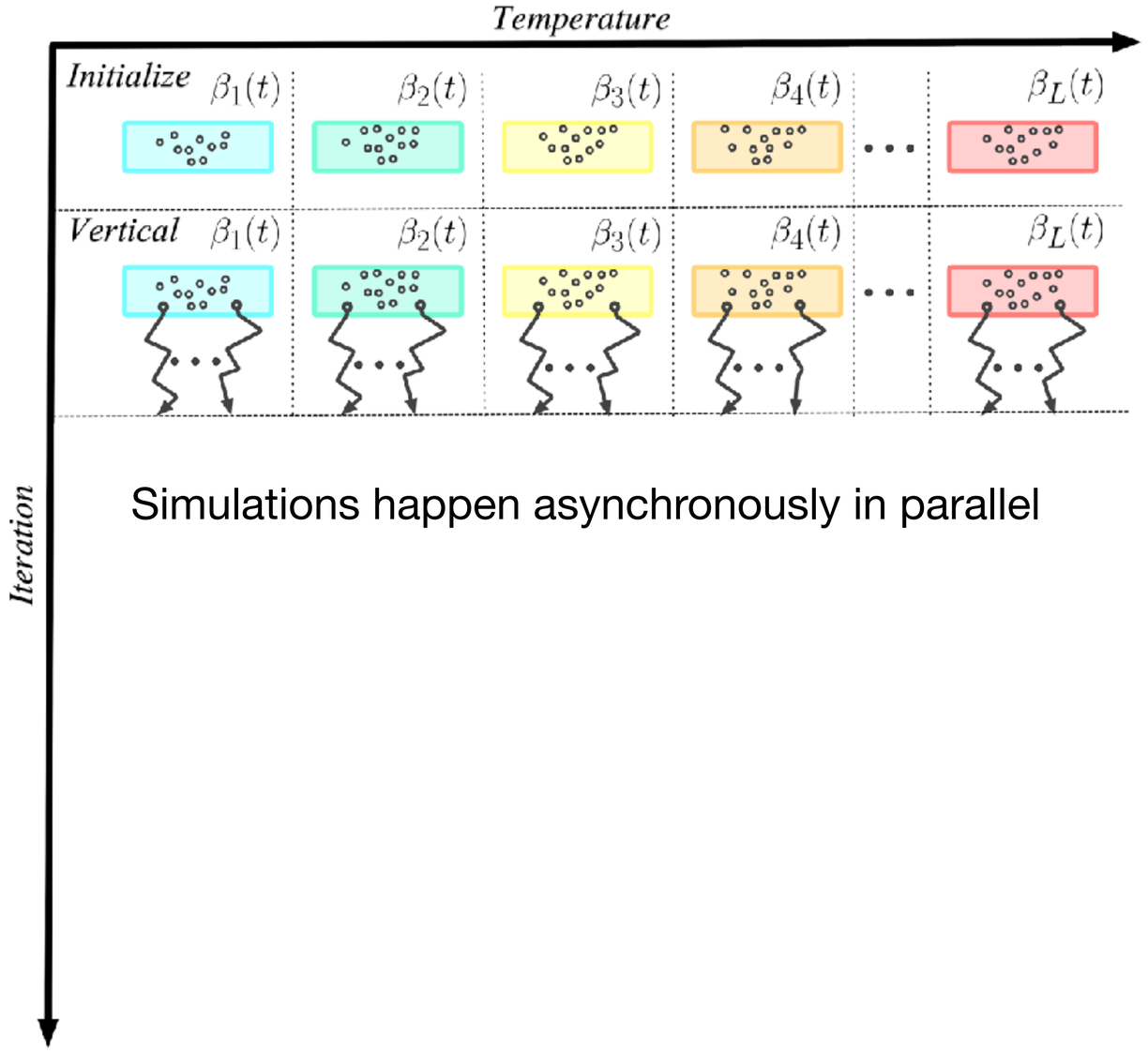
- Builds upon parallel tempering [Marinari & Parisi 1992]

- Initialize several “baths” of configurations (x, θ)



Step 2: self-play (vertical MCMC)

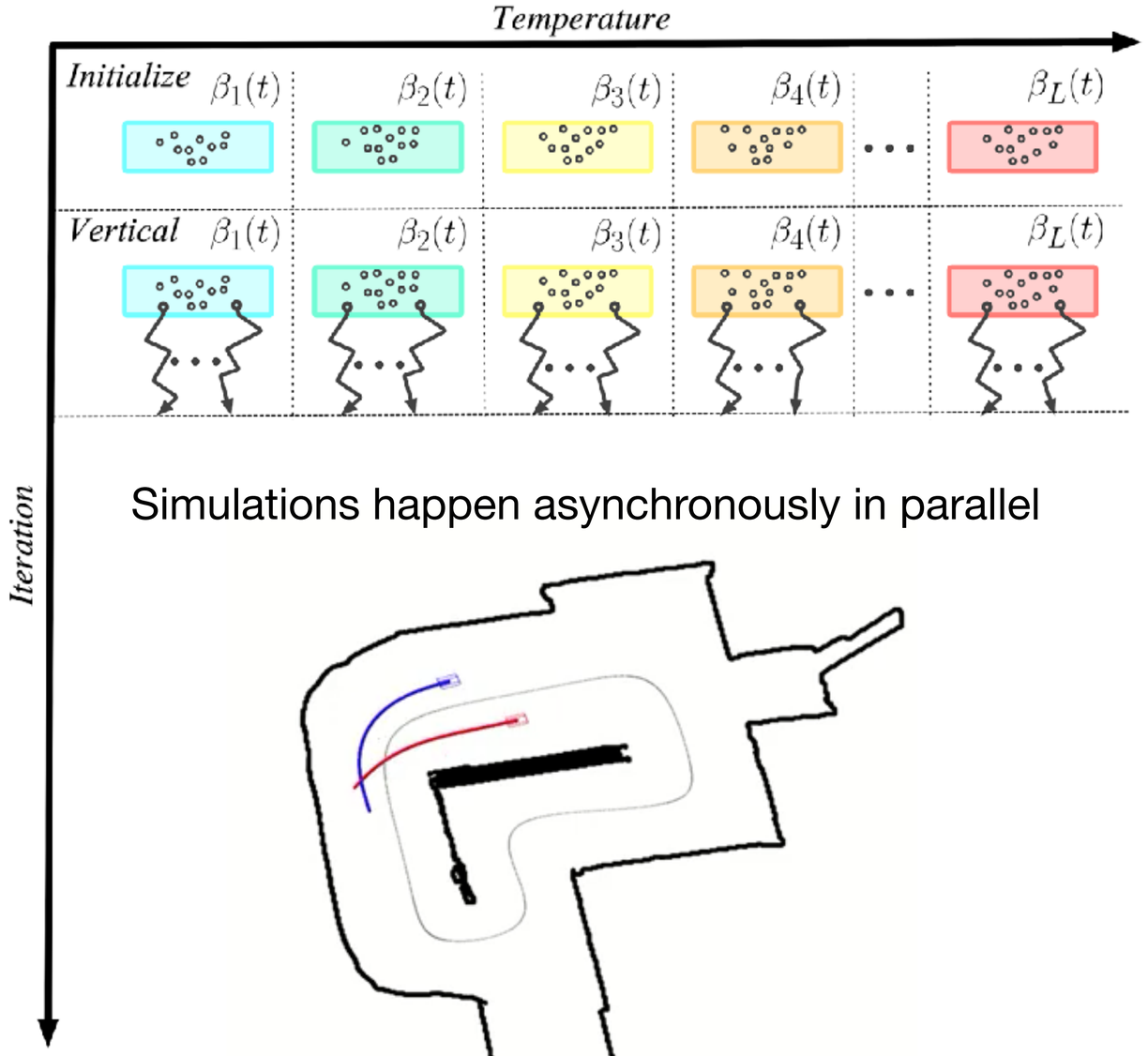
- Explore new proposals for x
- Evaluate each proposal by a race between the old and new configurations



Simulations happen asynchronously in parallel

Step 2: self-play (vertical MCMC)

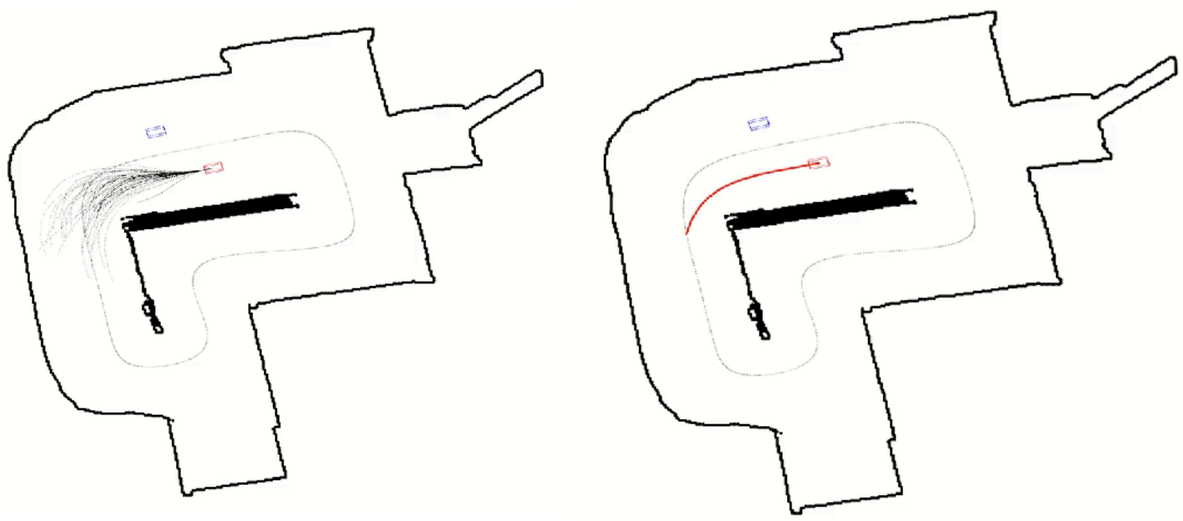
- Explore new proposals for x
- Evaluate each proposal by a race between the old and new configurations

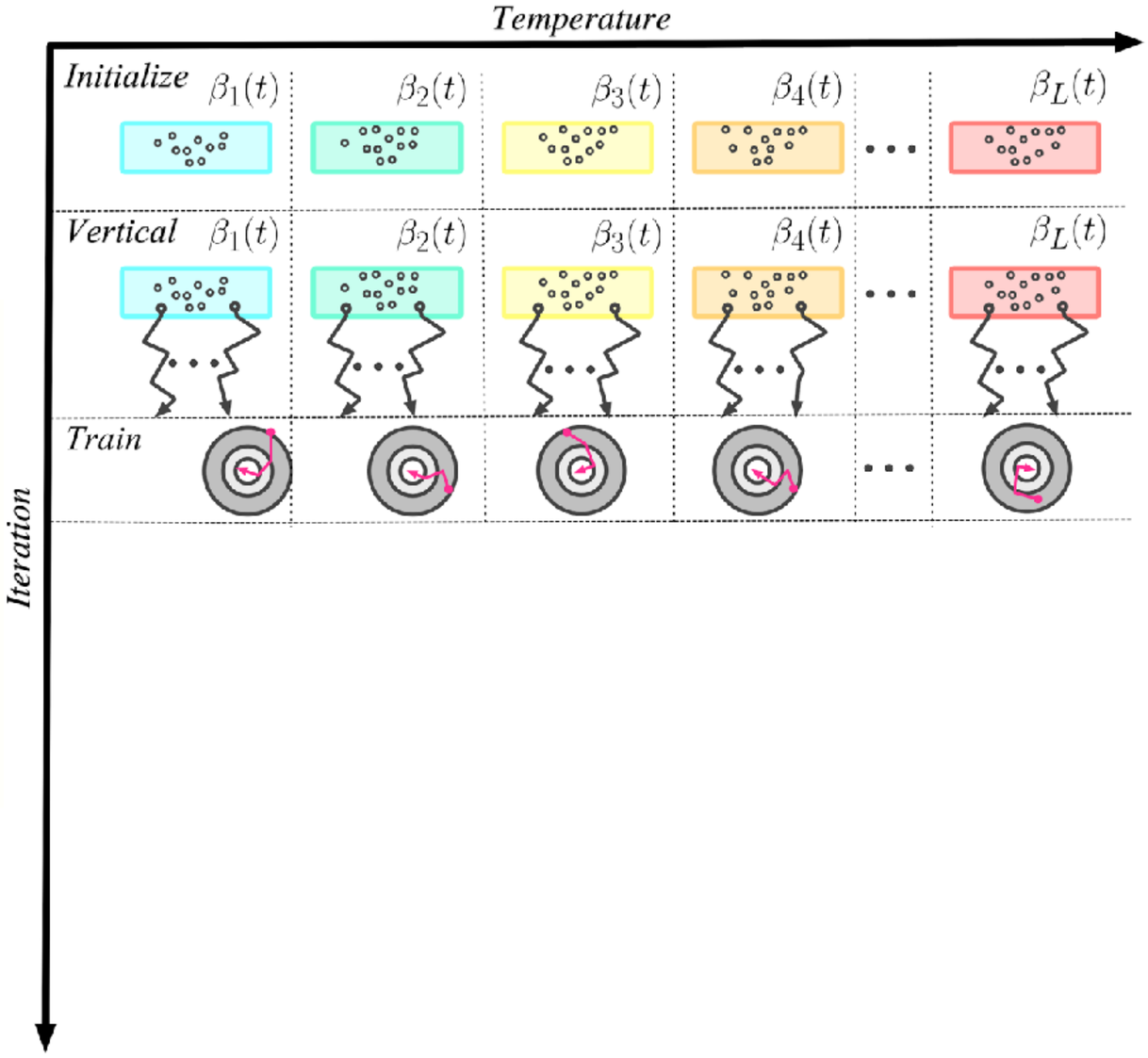


Simulations happen asynchronously in parallel

Step 3: differentiable parameter update

- Optimize θ (neural network weights)

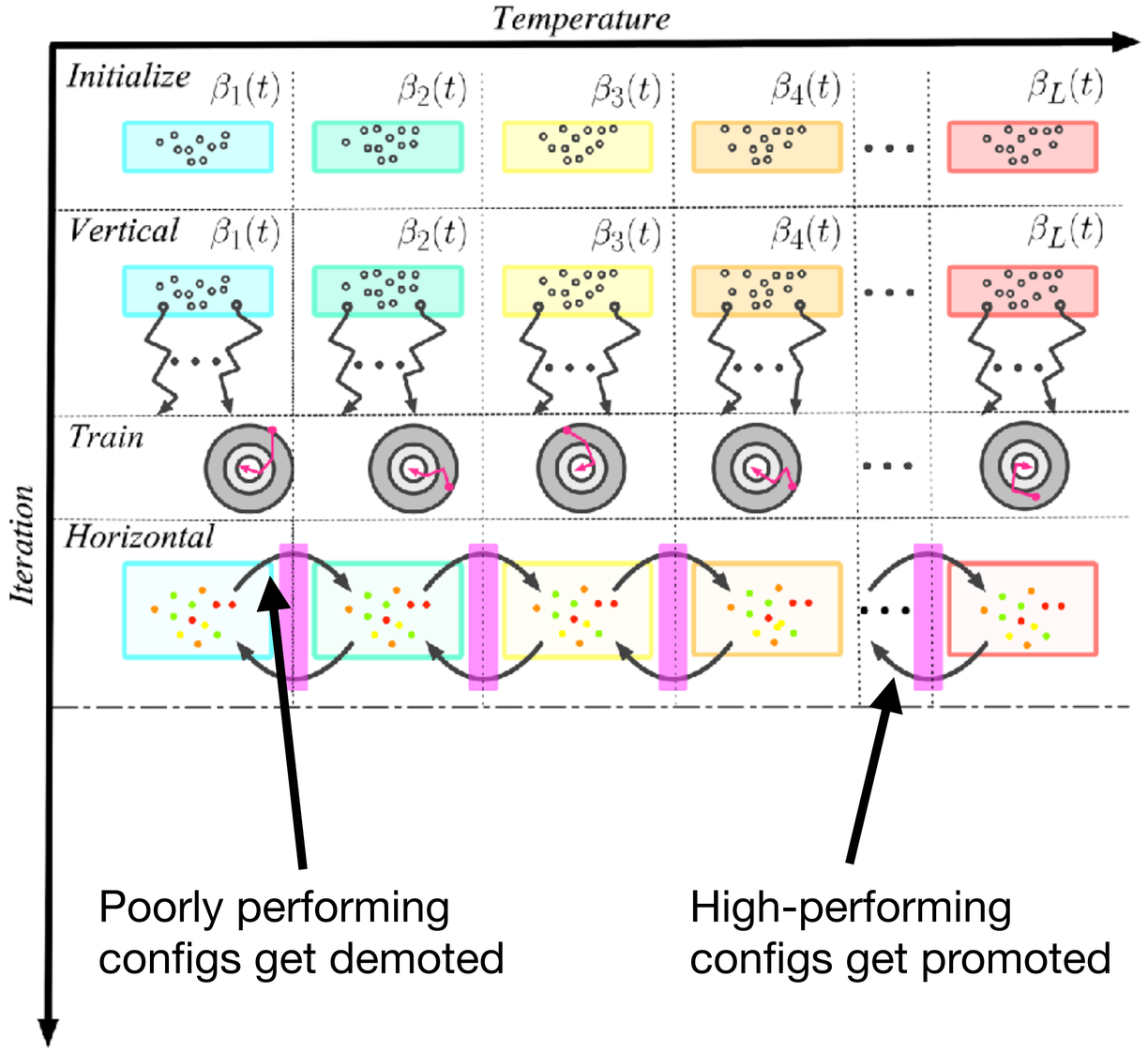




Step 4: configuration swaps (horizontal MCMC)

- Propose random swaps of configurations between adjacent baths

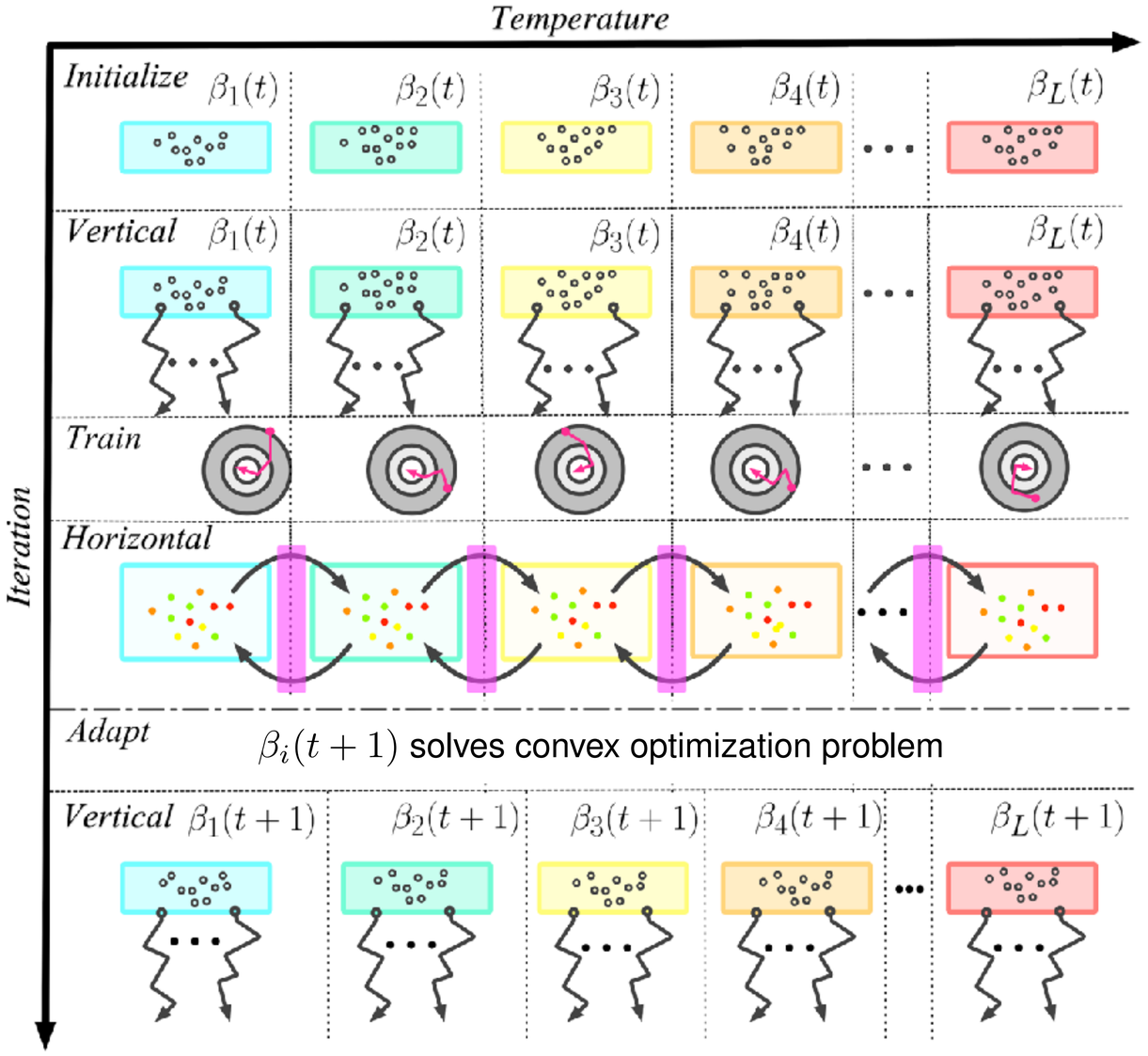
- Efficient way to encourage mixing because no new simulations needed



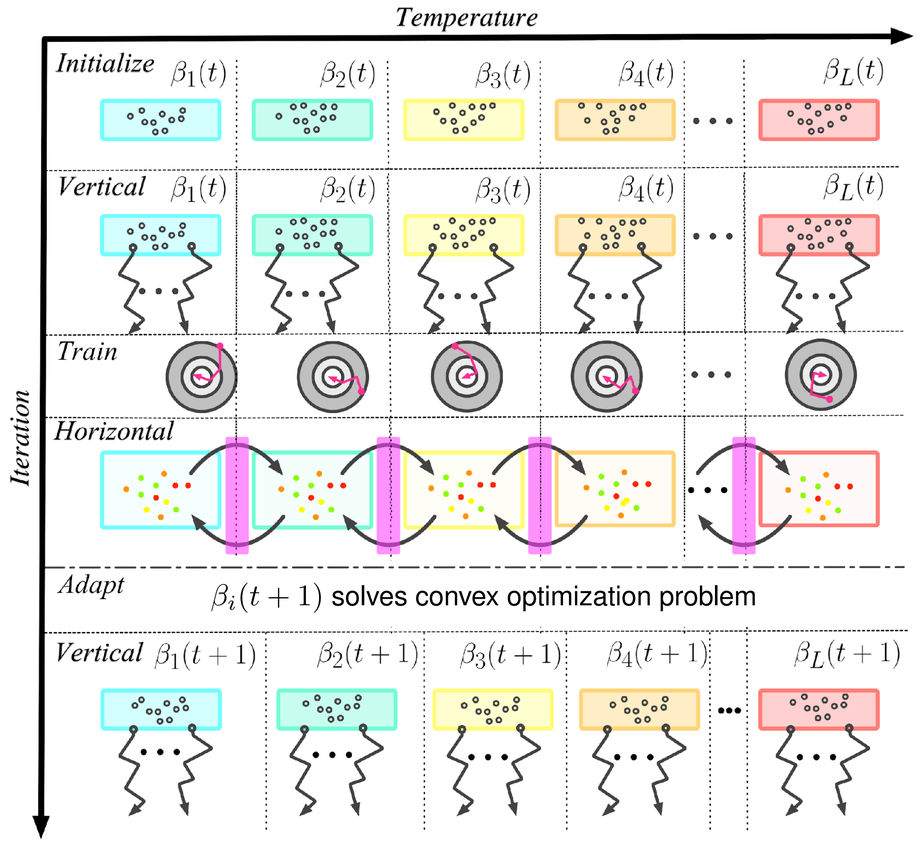
Step 5: update temperatures (annealing)

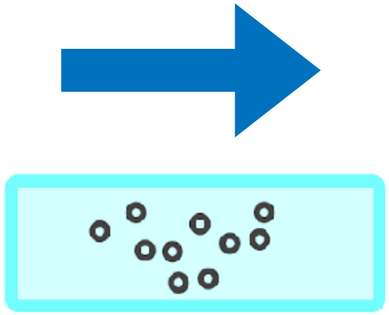
- Adaptive annealing scheme: adjust temperatures by annealing swap-acceptance probability
  - Convex optimization problem

- Crucial in our setting because we don't have any prior knowledge of good race times (no priors for good temperatures)

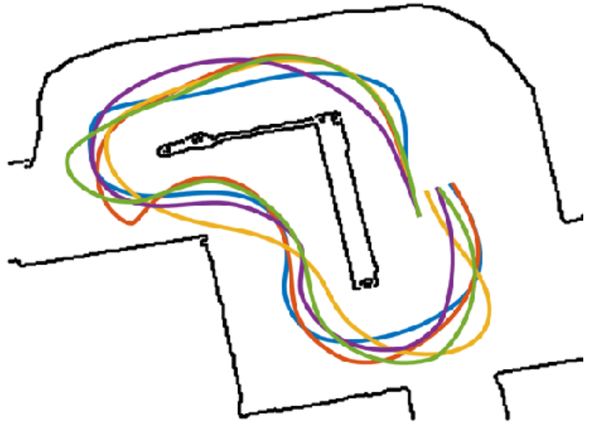


End result: diverse population of opponent prototypes

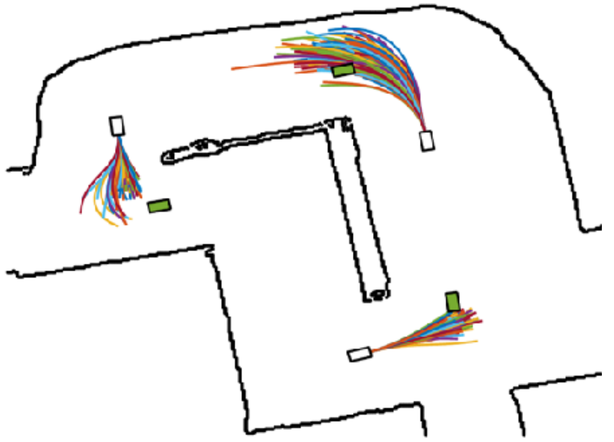




$\beta_1(t)$



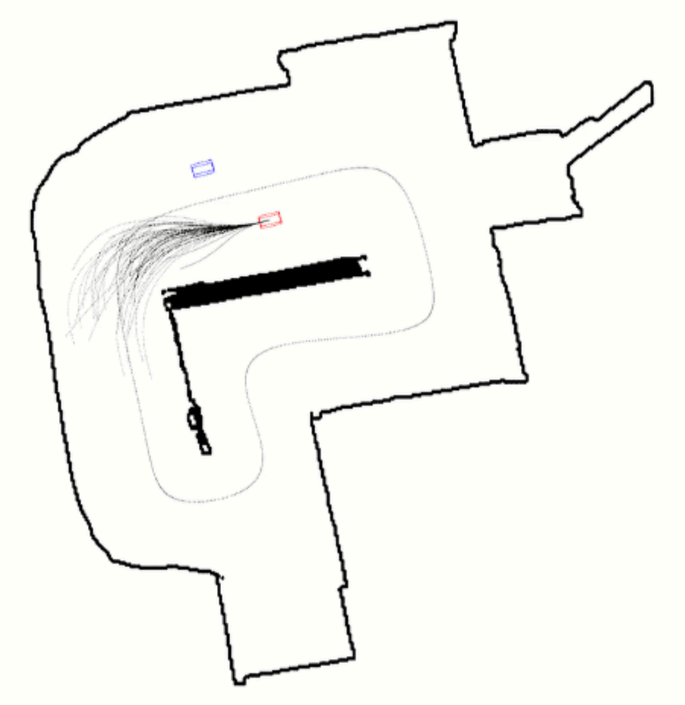
Diversity in isolated laps



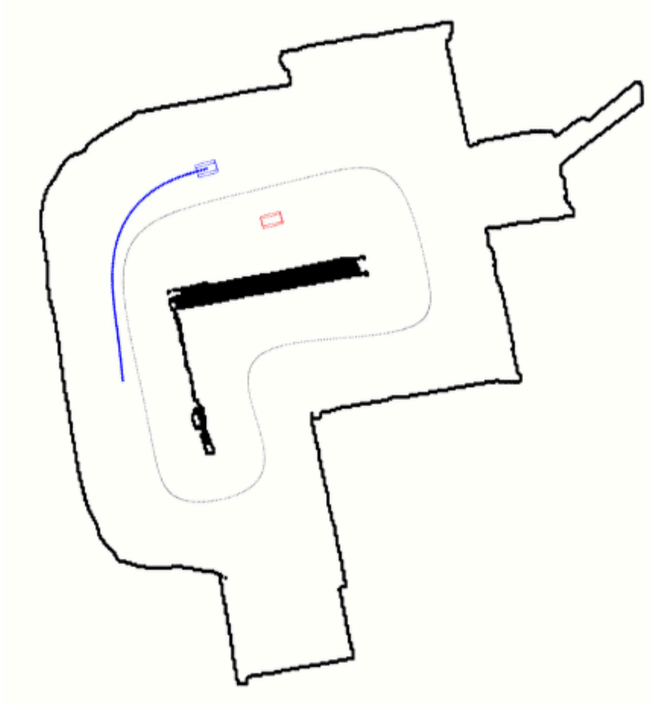
Diversity in maneuvering near an opponent

Distributionally robust planning

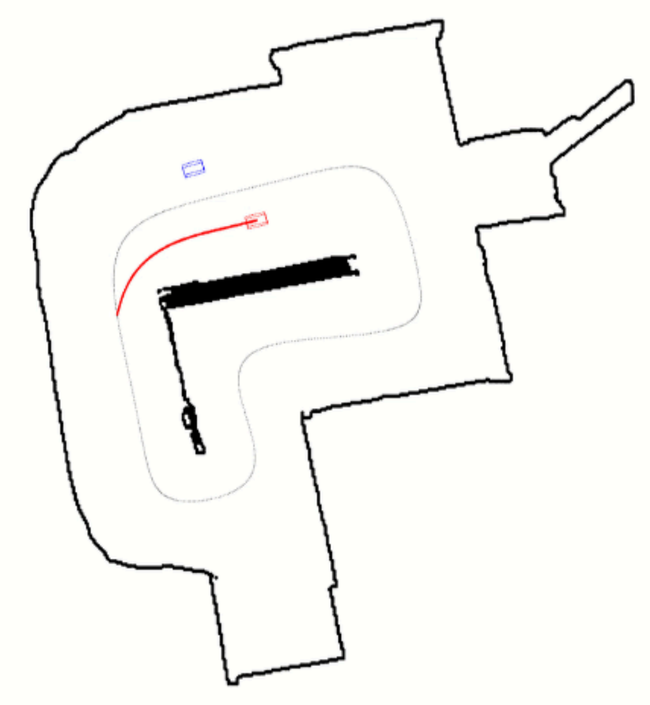
- When racing against an opponent, we maintain a belief vector $w(t)$ of their behavior over the learned population of prototypes
- $\mathcal{P}$ is an uncertainty ball around this belief (χ^2-divergence)



Draw candidate goal



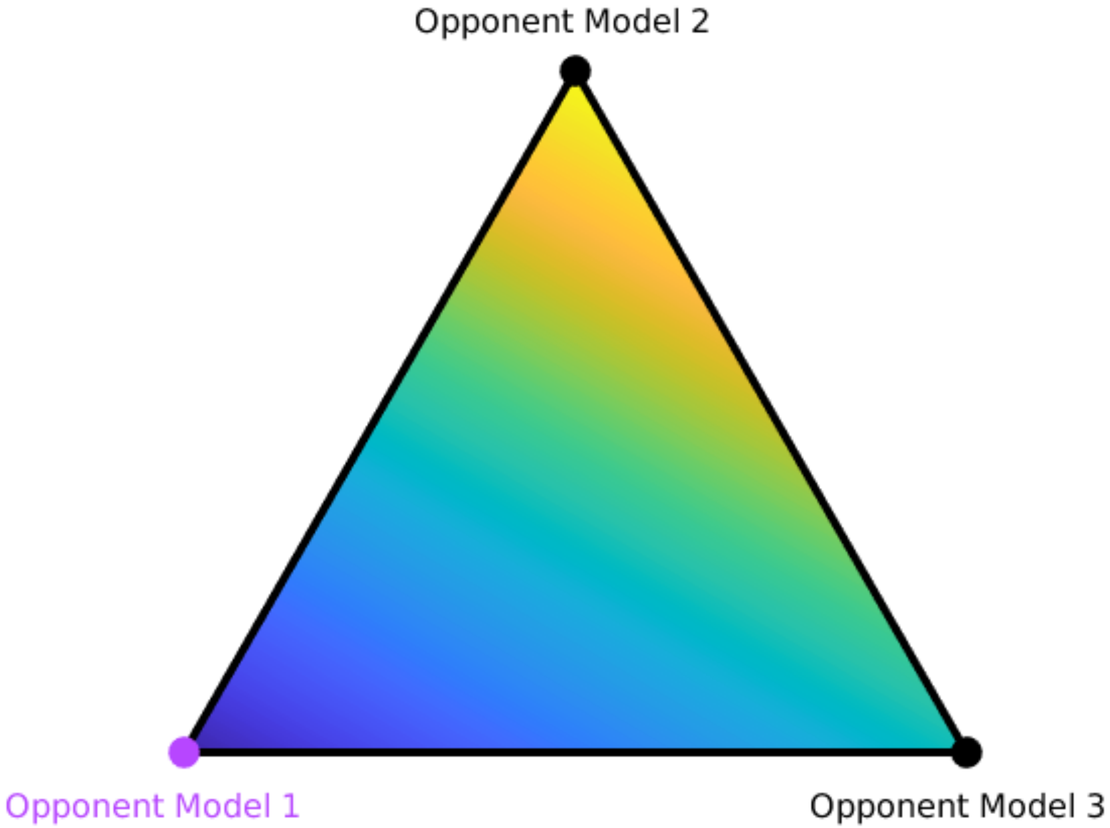
Predict opponent behavior



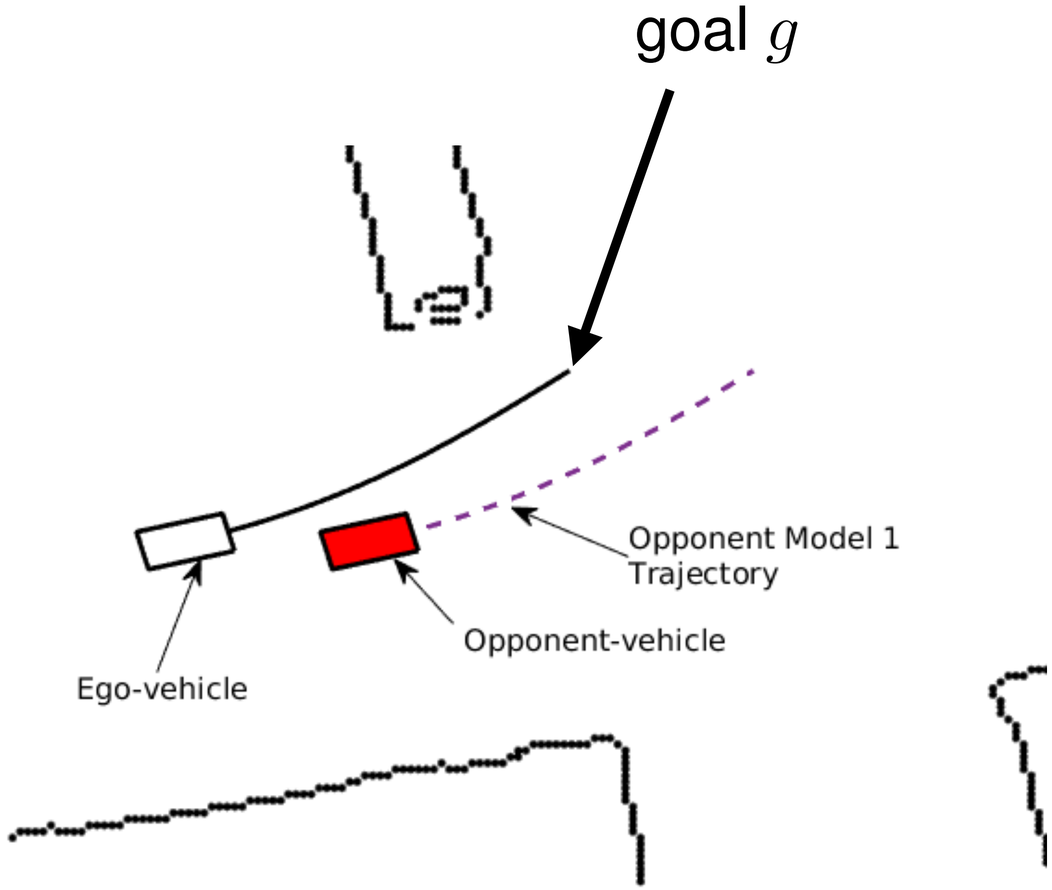
Choose goal

Distributionally robust planning

$$c_1(t;g) := \sum_{s>t} \lambda^{s-t} \mathbb{E}[c(o(s);g)]$$

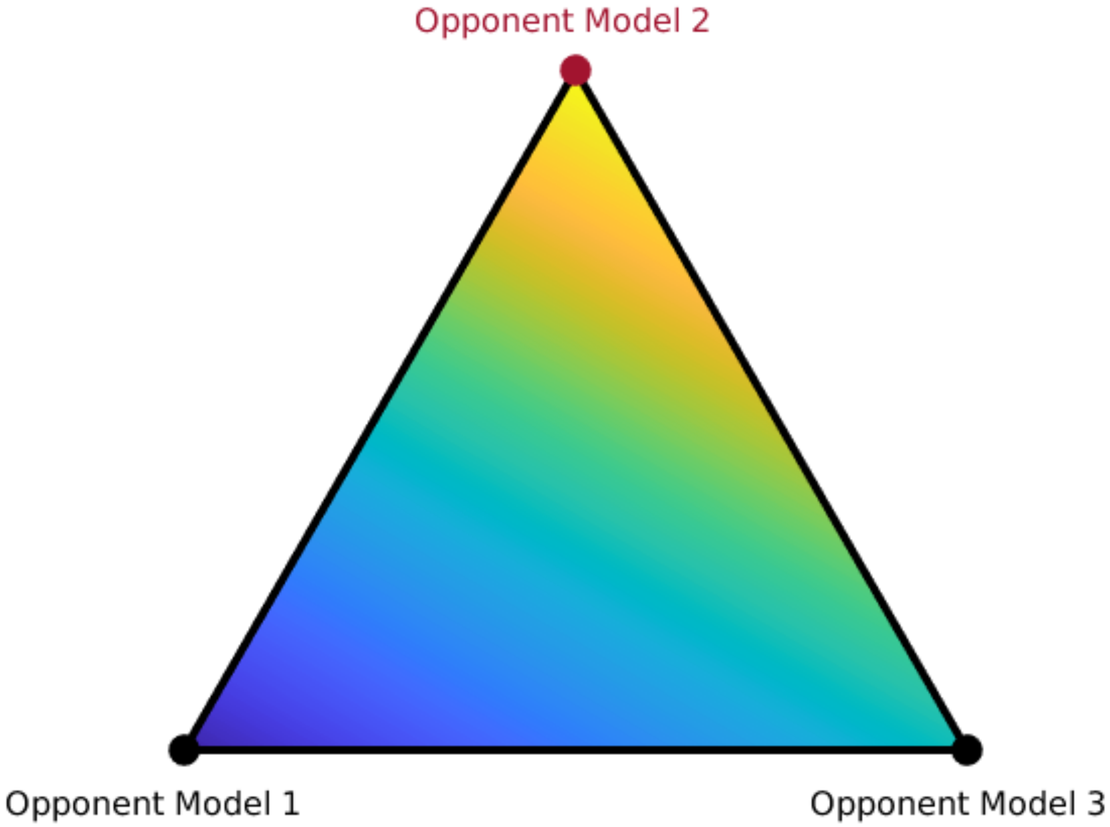


Belief space

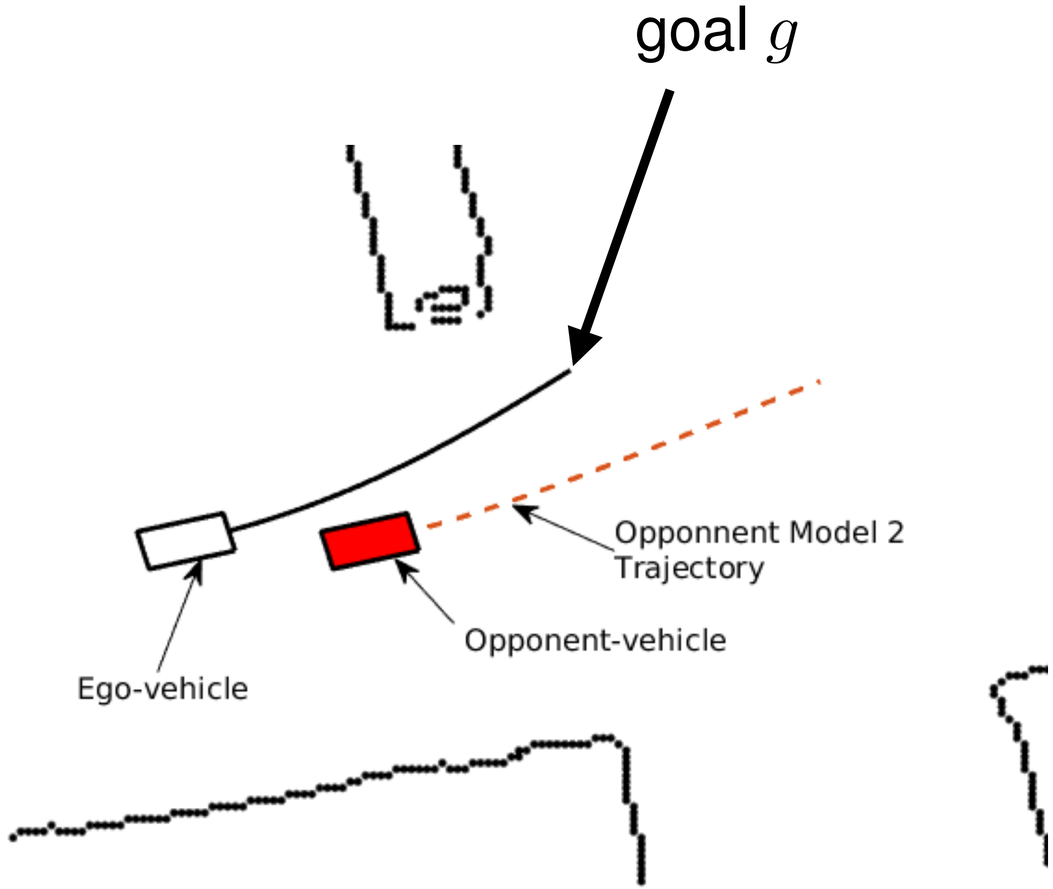


Distributionally robust planning

$$c_2(t; g) := \sum_{s>t} \lambda^{s-t} \mathbb{E}[c(o(s); g)]$$

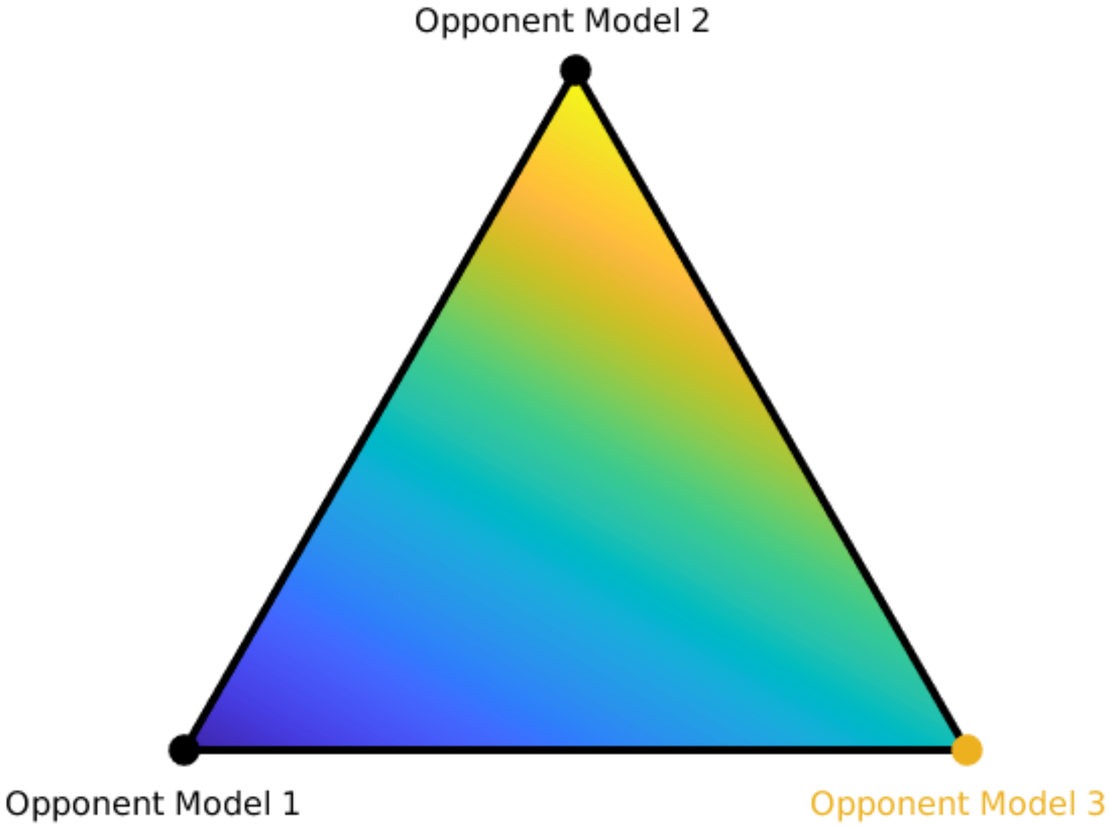


Belief space

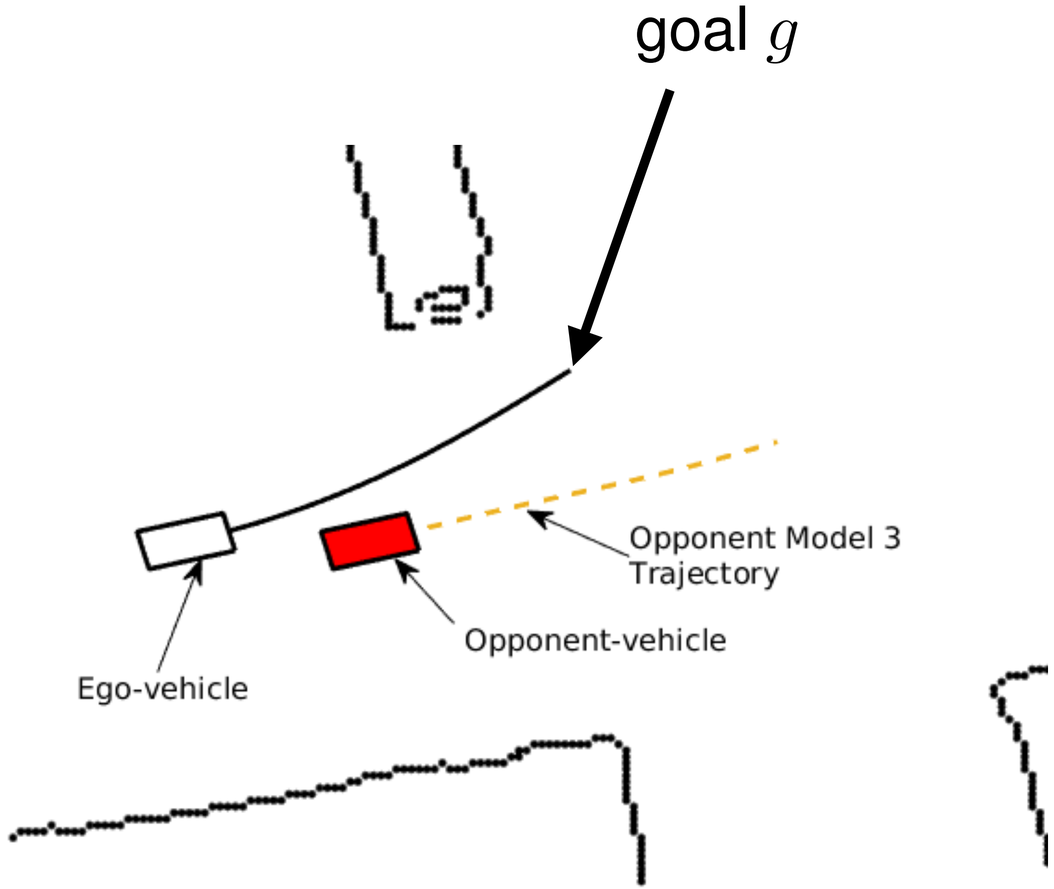


Distributionally robust planning

$$c_3(t;g) := \sum_{s>t} \lambda^{s-t} \mathbb{E}[c(o(s);g)]$$

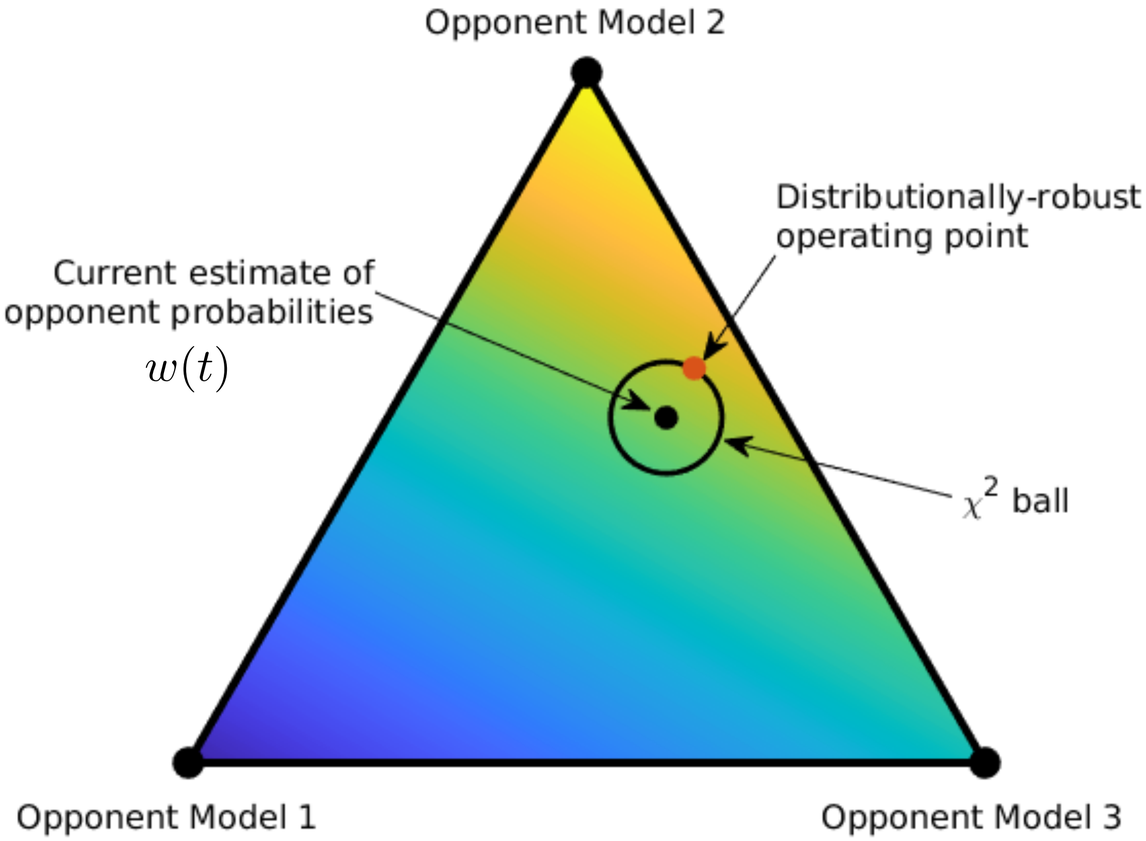


Belief space

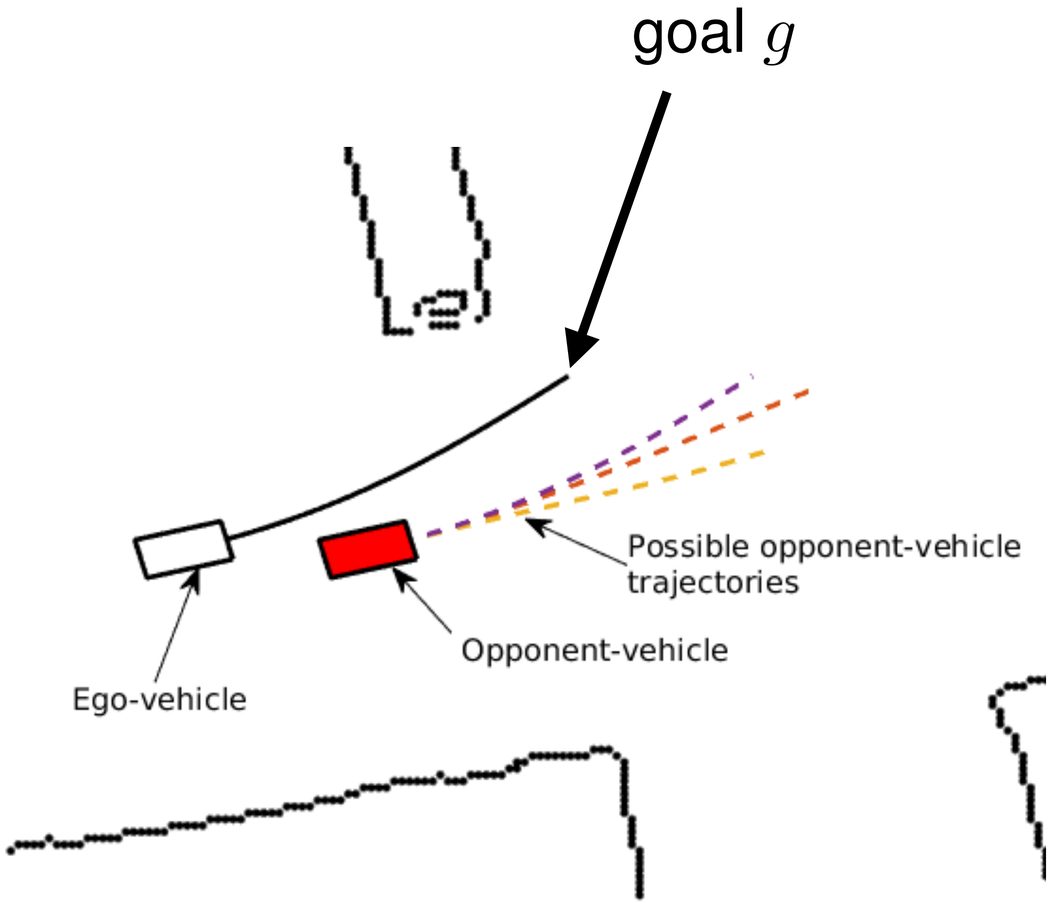


Distributionally robust planning

$$\max_{q:\sum_i w_i(\frac{q_i}{w_i})^2 \leq \rho+1} \sum_i q_i c_i(t;g)$$

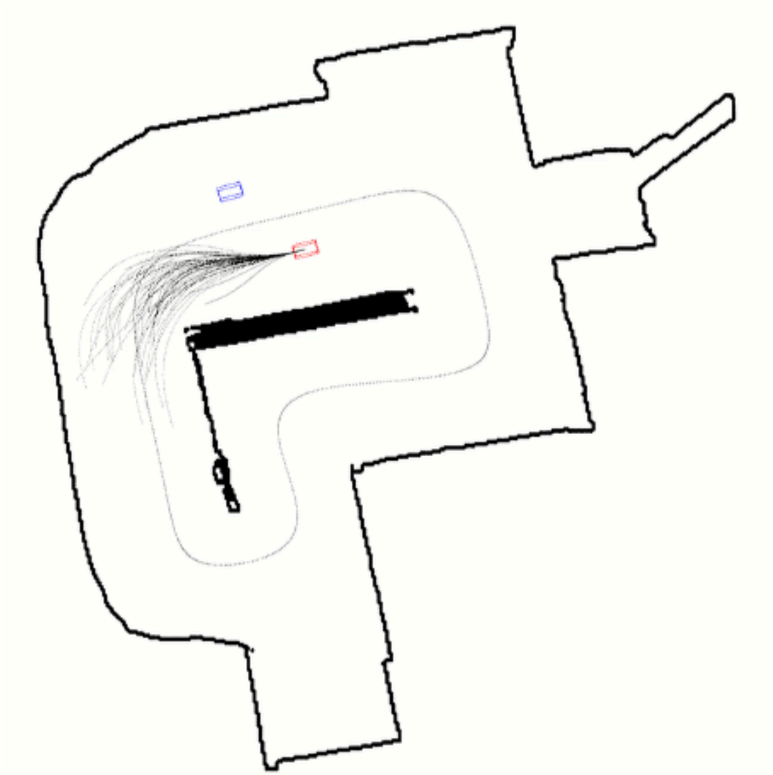


Belief space

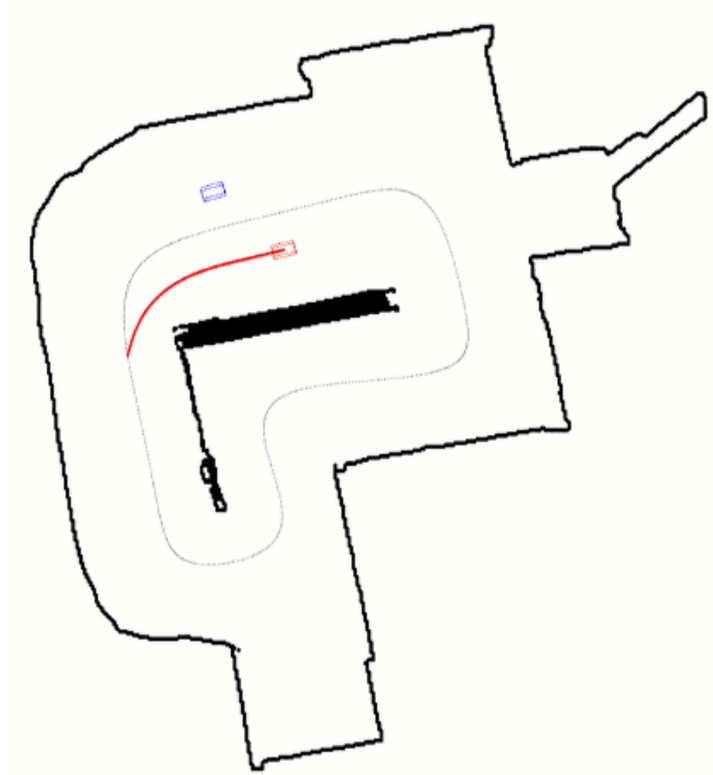


Distributionally robust planning

Repeat this for every motion planning goal and select the goal with the lowest robust cost

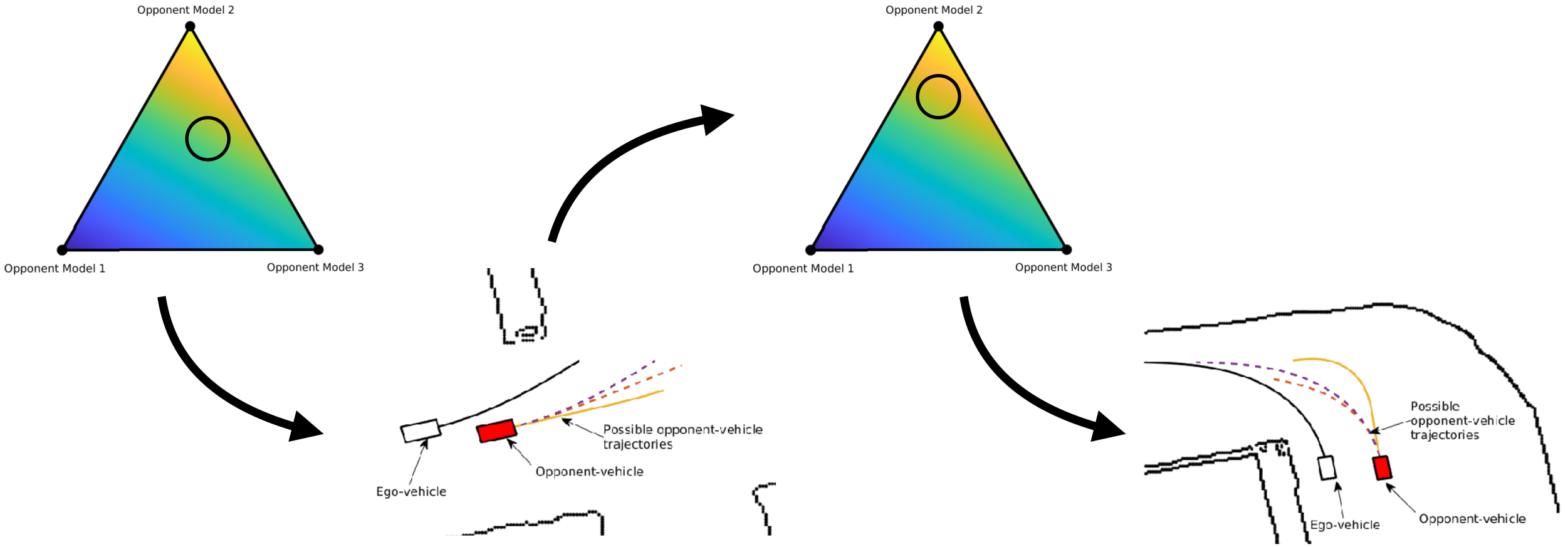


$$\max_{q:\sum_i w_i(\frac{q_i}{w_i})^2 \leq \rho+1} \sum_i q_i c_i(t;g)$$

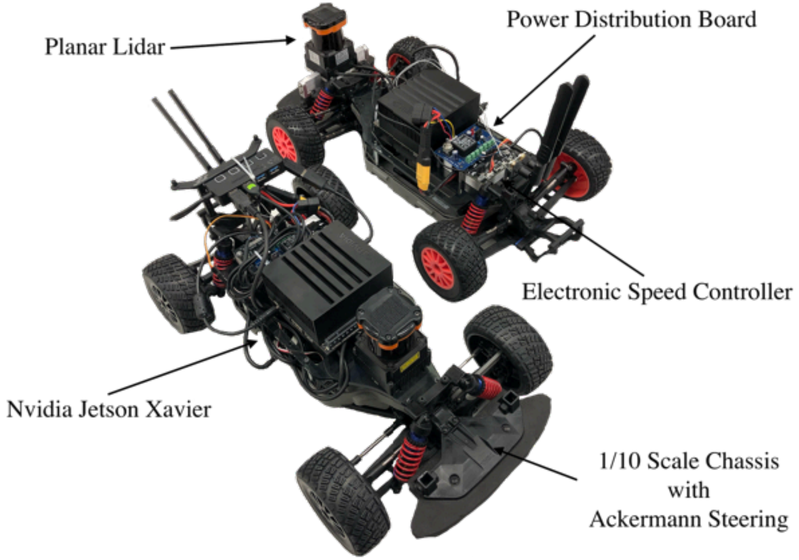


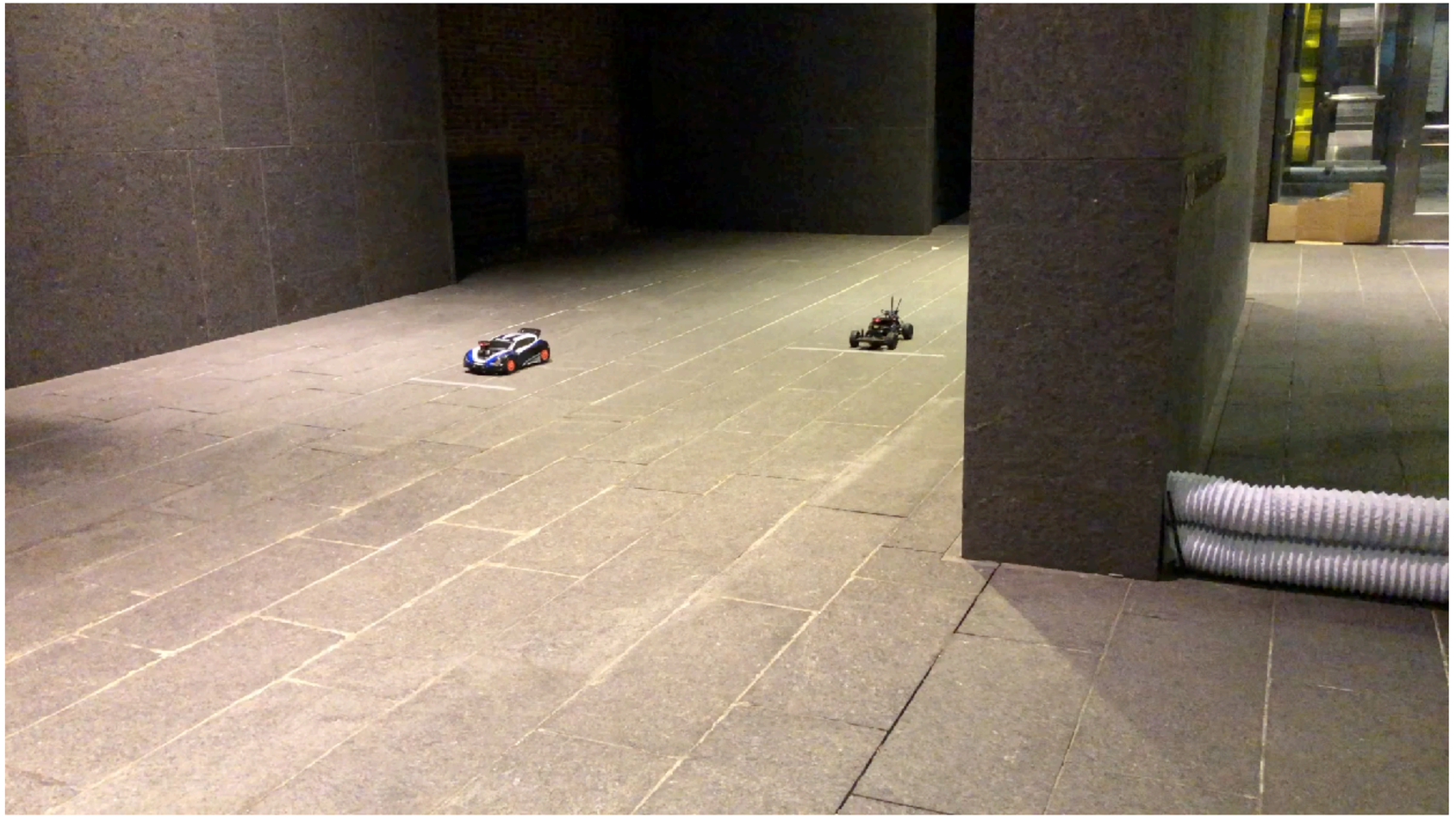
Belief updates (adaptivity)

- Update beliefs using the observed history of the opponent
  - Modified version of EXP3 [Auer et al. 2002]



Real-world experiments





Balancing safety and performance

Larger uncertainty sets (larger ρ) increase safety but decrease performance

Agent	% of iTTC values < 0.5s
$\rho/N_w = 0.001$	$\mathbf{7.86 \pm 0.90}$
$\rho/N_w = 0.025$	6.46 ± 0.78
$\rho/N_w = 0.2$	4.75 ± 0.65
$\rho/N_w = 0.4$	5.41 ± 0.74
$\rho/N_w = 0.75$	5.50 ± 0.82
$\rho/N_w = 1.0$	$\mathbf{5.76 \pm 0.84}$

Increased safety with ρ

Agent	Win-rate Non-adaptive
$\rho/N_w = 0.001$	$\mathbf{0.593 \pm 0.025}$
$\rho/N_w = 0.025$	0.593 ± 0.025
$\rho/N_w = 0.2$	0.538 ± 0.025
$\rho/N_w = 0.4$	0.503 ± 0.025
$\rho/N_w = 0.75$	0.513 ± 0.025
$\rho/N_w = 1.0$	0.498 ± 0.025

Decreased win-rate with ρ

Balancing safety and performance

Online adaptation regains the performance of aggressive strategies:
Safe when uncertain, aggressive once the opponent is identified

Agent	Win-rate Non-adaptive	Win-rate Adaptive	p-value
$\rho/N_w = 0.001$	$\mathbf{0.593 \pm 0.025}$	0.588 ± 0.025	0.84
$\rho/N_w = 0.025$	0.593 ± 0.025	0.600 ± 0.024	0.77
$\rho/N_w = 0.2$	0.538 ± 0.025	0.588 ± 0.025	0.045
$\rho/N_w = 0.4$	0.503 ± 0.025	0.573 ± 0.025	0.0098
$\rho/N_w = 0.75$	0.513 ± 0.025	0.593 ± 0.025	0.0013
$\rho/N_w = 1.0$	0.498 ± 0.025	$\mathbf{0.590 \pm 0.025}$	0.00024

Key ideas of this talk

Robustness

Build models with guaranteed performance over uncertainty sets

$$\min_{\theta \in \Theta} \max_{Q \in \mathcal{P}} \mathbb{E}_Q[f(\theta; X)]$$

- With small $\mathcal{P}$, can we solve this quickly?
- What about when $\mathcal{P}$ is large/unknown?

Risk

Find failure modes and quantify the probability of failure

$$\mathbb{P}_0(f(X) < \gamma)$$

- Why is this the right problem?
- How do we solve it quickly?

The risk-based framework

O’Kelly*, Sinha*, Namkoong*, Duchi, Tedrake. NeurIPS 2018.
O’Kelly*, Sinha*, Norden*, Namkoong*. NeurIPS ML4H 2018.





The risk-based framework

Tesla Autopilot crashing in highway scenarios

The risk-based framework

- Certify a level of reliability
- Work with a blackbox algorithm

Related work

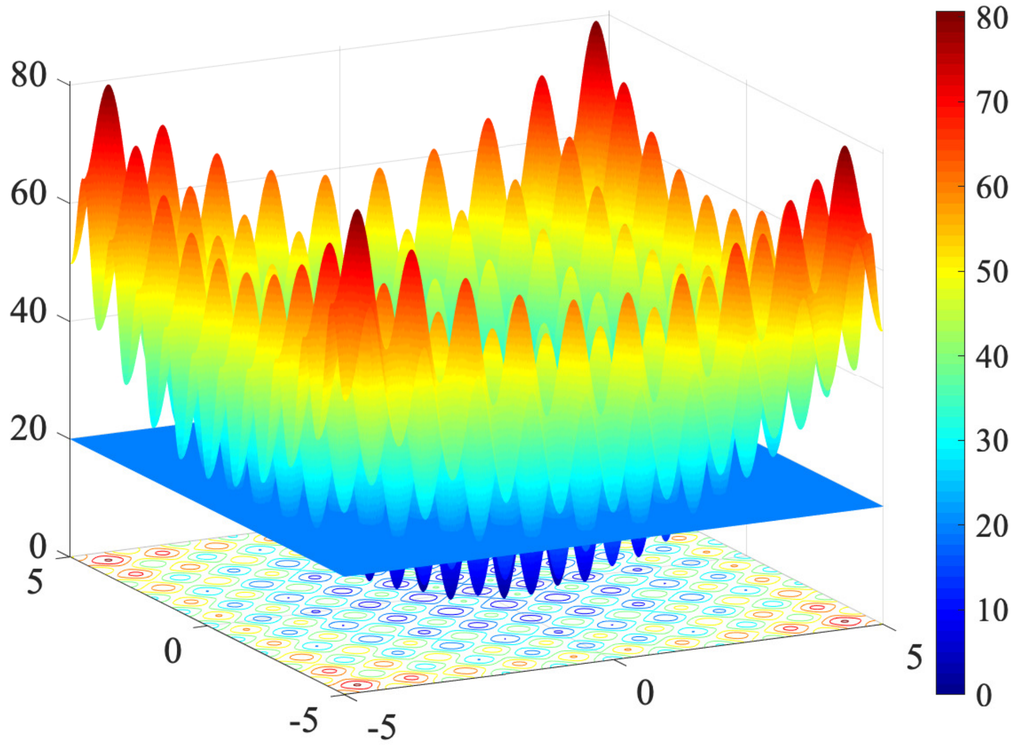
- Formal verification [Kwiatkowska et al. 2011, Althoff and Dolan 2014, Seshia et al. 2015, O'Kelly et al. 2016]
- Falsification [Abbas and Fainekos 2011, Tuncali et al. 2016, DeCastro et al. 2018]
- Probabilistic falsification/Adaptive stress testing [Koren et al. 2018, Lee et al. 2018]

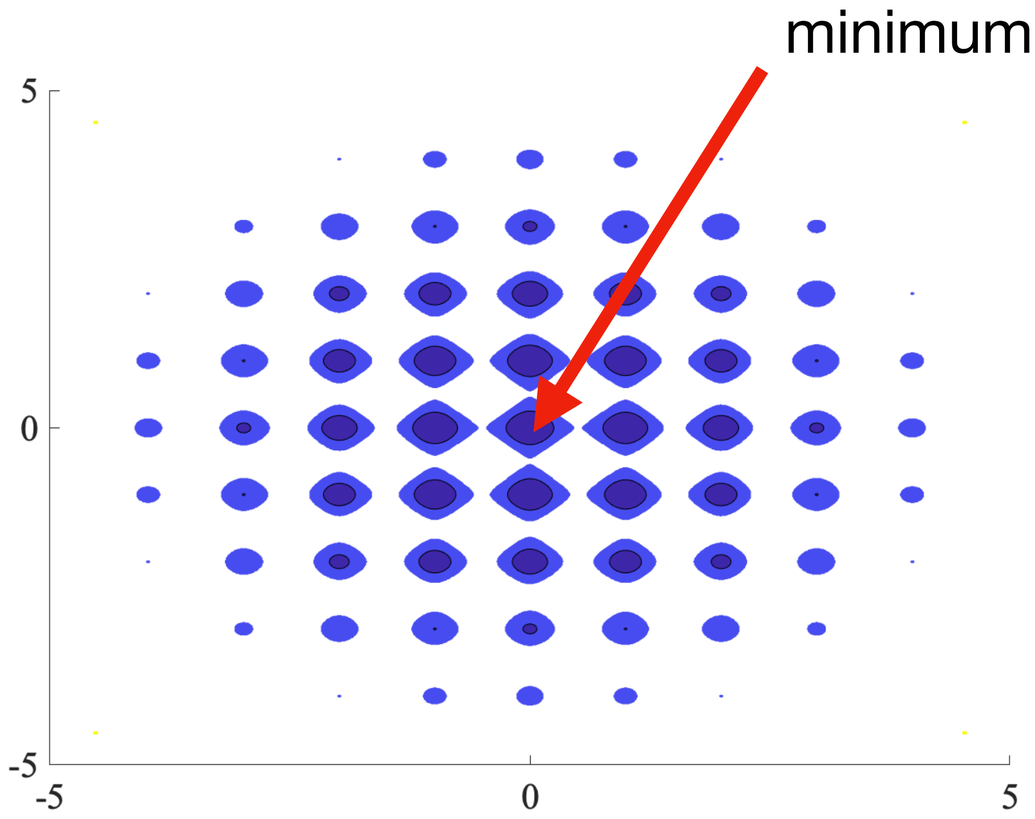
Problems with verification

- A “correctness” specification is subjective
- Intractable
- Requires white-box model

Problems with falsification

- Not designed for coverage



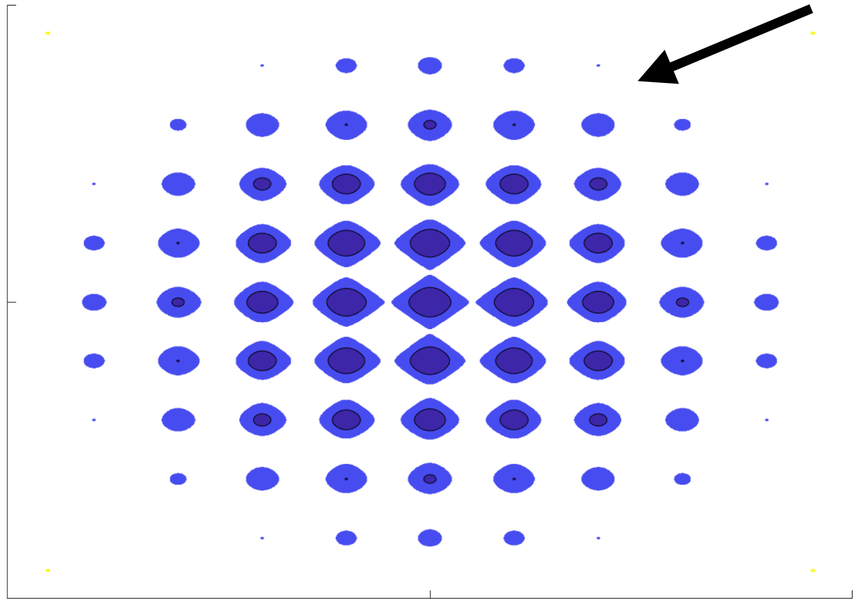


The risk-based framework

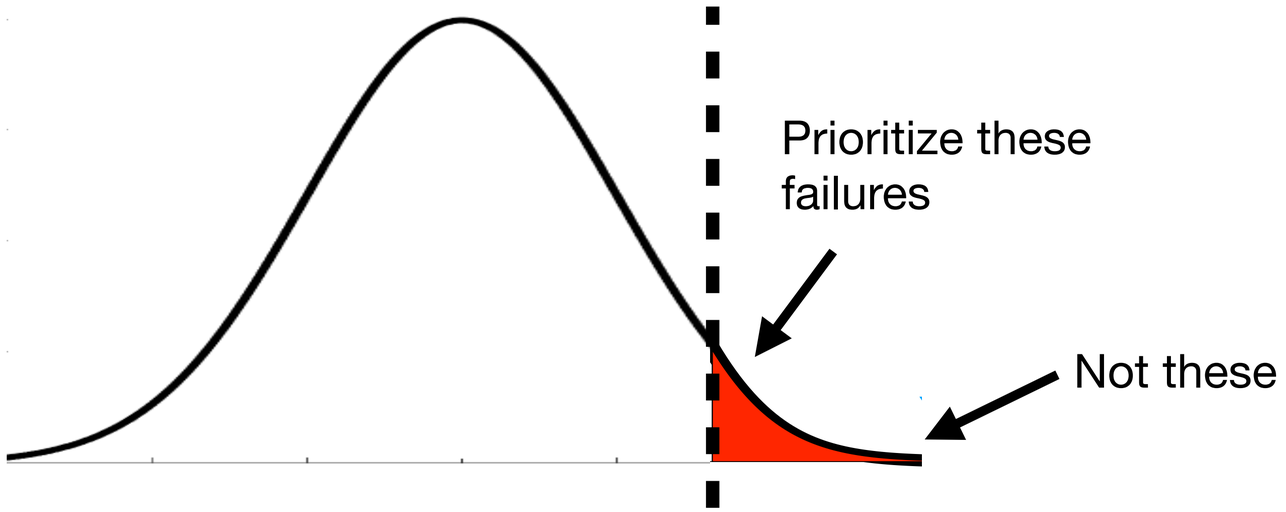
- Goal: probability of dangerous event $p_\gamma := \mathbb{P}_0(f(X) < \gamma)$
- Base distribution of behavior $X \sim P_0$
- Objective function (i.e. safety metric) $f : \mathcal{X} \to \mathbb{R}$

The risk-based framework

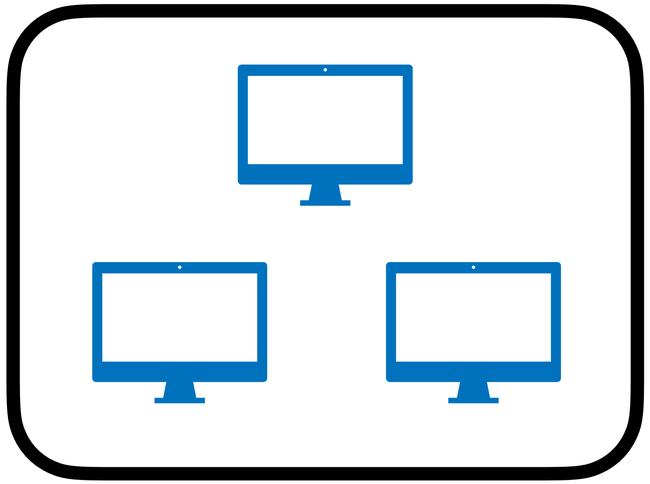
- Coverage of failure modes



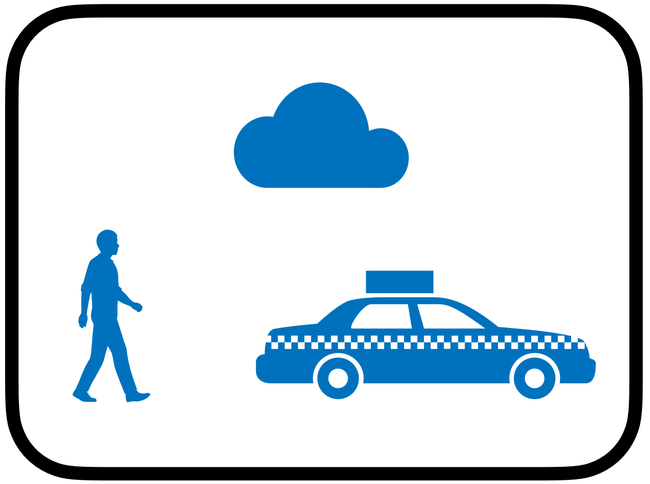
- Prioritization by likelihood



Components of the risk-based framework



Simulation



Generative models

Search algorithm

Search

$$p_\gamma := \mathbb{P}_0(f(X) < \gamma)$$

- Random search (aka naive Monte Carlo)

$$\hat{p}_\gamma = \frac{1}{N}\sum_{i=1}^{N} I\{f(x_i) < \gamma\}$$

Estimate

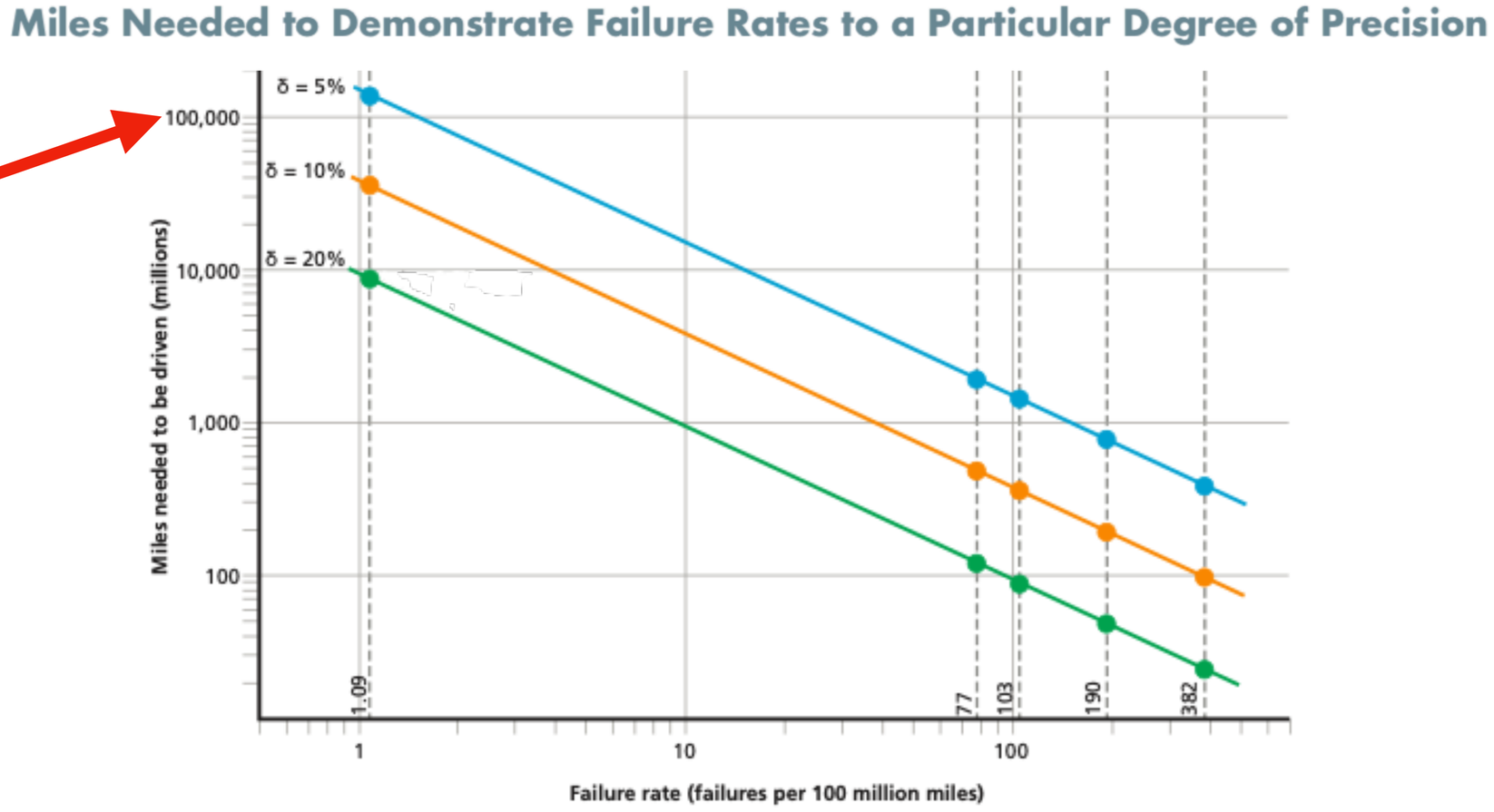
$$\mathbb{E}[(\hat{p}_\gamma / p_\gamma - 1)^2] = \frac{1 - p_\gamma}{N p_\gamma}$$

Error of estimate

- Rule of thumb: need at least $100/p_\gamma$ samples for accurate estimate (error bars < 10%)

Search

100 billion miles needed per code release!



[Kalra & Paddock 2016]

Key ideas of this talk

Robustness

Build models with guaranteed performance over uncertainty sets

$$\min_{\theta \in \Theta} \max_{Q \in \mathcal{P}} \mathbb{E}_Q[f(\theta; X)]$$

- With small $\mathcal{P}$, can we solve this quickly?
- What about when $\mathcal{P}$ is large/unknown?

Risk

Find failure modes and quantify the probability of failure

$$\mathbb{P}_0(f(X) < \gamma)$$

- Why is this the right problem?
- How do we solve it quickly?

Techniques for rare-event simulation

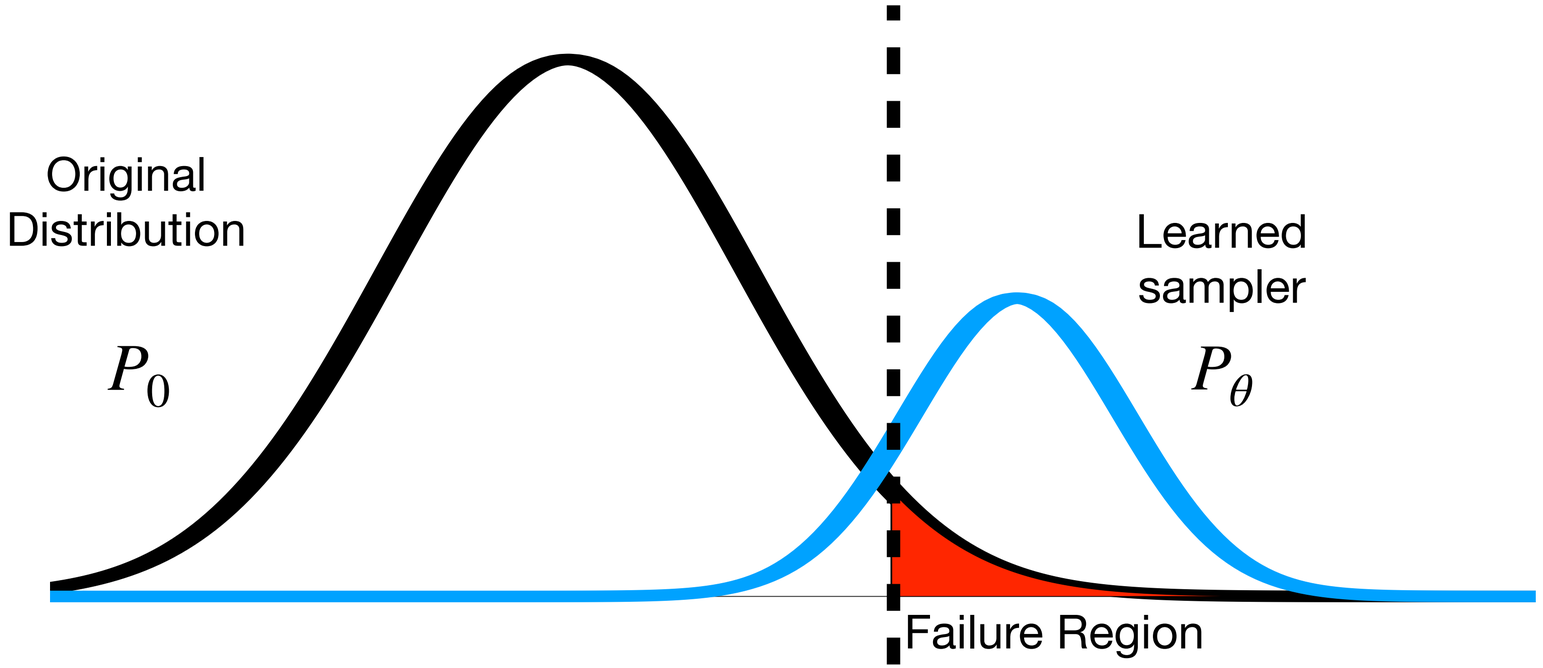
O’Kelly*, Sinha*, Namkoong*, Duchi, Tedrake. NeurIPS 2018.
Sinha*, O’Kelly*, Duchi, Tedrake. Under review.




Warm up: cross-entropy method

Goal: find a good parametric importance-sampling distribution

- Explore: draw samples from current sampling distribution
- Exploit: update sampling distribution



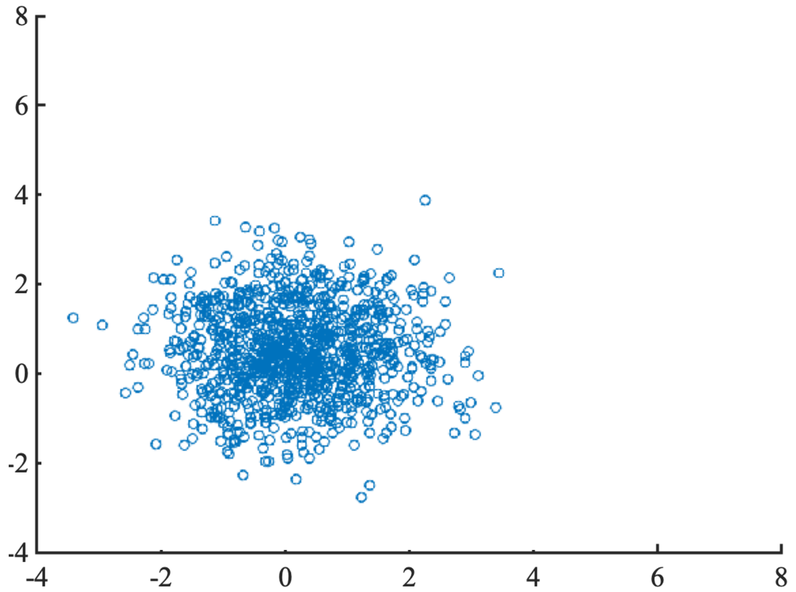
[Rubinstein & Kroese 2004, Asmussen & Glynn 2007]

Warm up: cross-entropy method

Goal: find a good parametric importance-sampling distribution

- Explore: draw samples from current sampling distribution
- Exploit: update sampling distribution

$$P_0 = \mathcal{N}(0, I)$$



Warm up: cross-entropy method

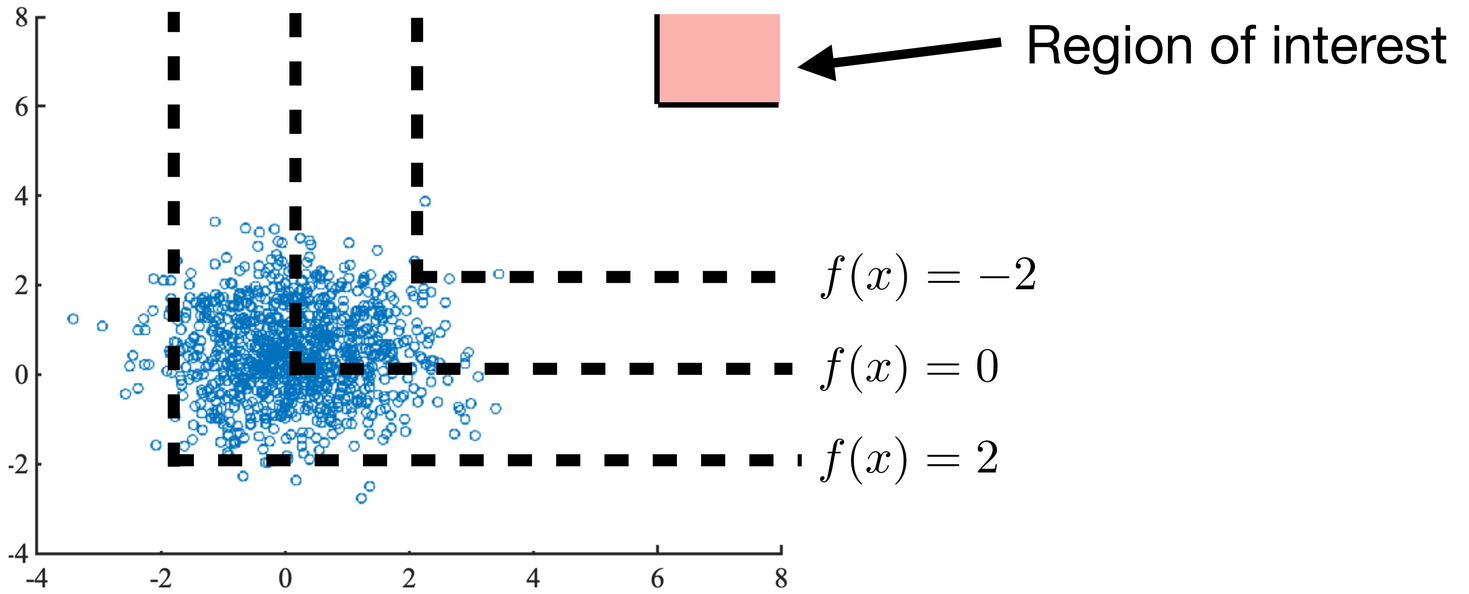
Goal: find a good parametric importance-sampling distribution

- Explore: draw samples from current sampling distribution
- Exploit: update sampling distribution

$$P_0 = \mathcal{N}(0, I)$$

$$f(x) = -\min(x_{[i]})$$

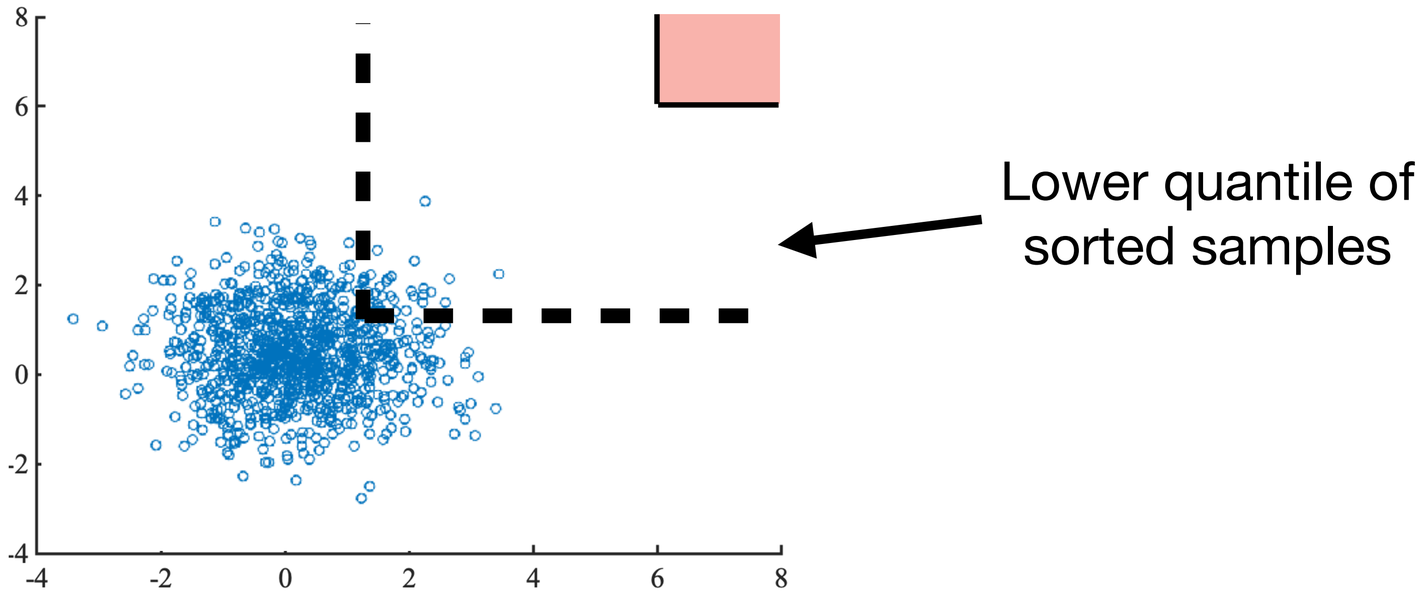
$$\gamma = -6$$



Warm up: cross-entropy method

Goal: find a good parametric importance-sampling distribution

- Explore: draw samples from current sampling distribution
- Exploit: update sampling distribution

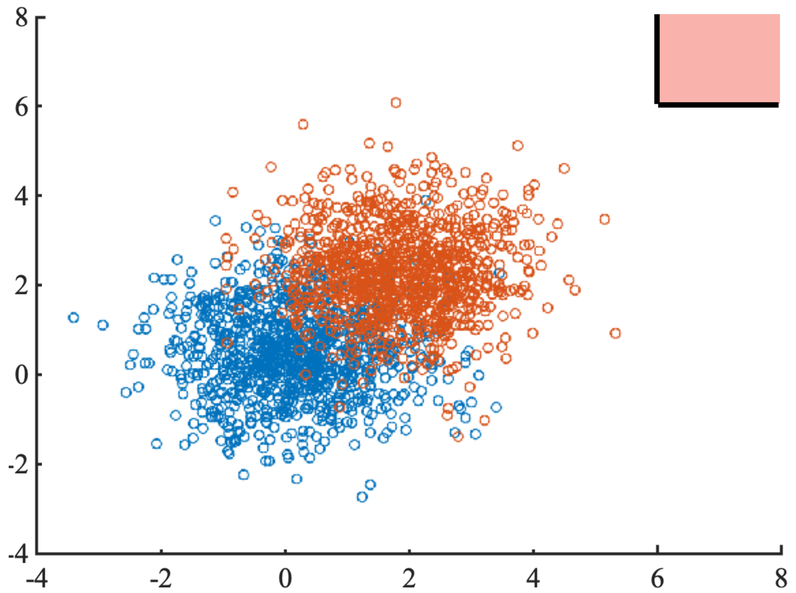


Warm up: cross-entropy method

Goal: find a good parametric importance-sampling distribution

- Explore: draw samples from current sampling distribution
- Exploit: update sampling distribution

Sampling family: $\mathcal{N}(\theta, 1)$

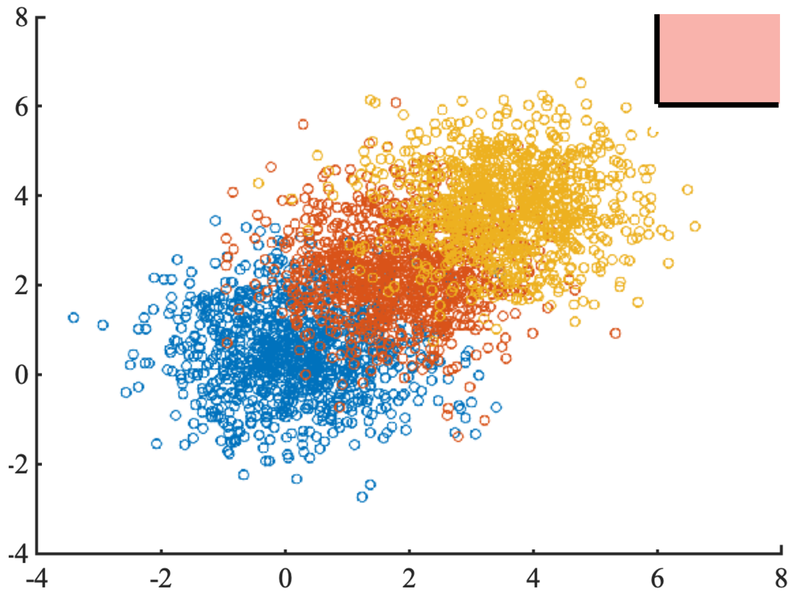


Warm up: cross-entropy method

Goal: find a good parametric importance-sampling distribution

- Explore: draw samples from current sampling distribution
- Exploit: update sampling distribution

Sampling family: $\mathcal{N}(\theta, 1)$

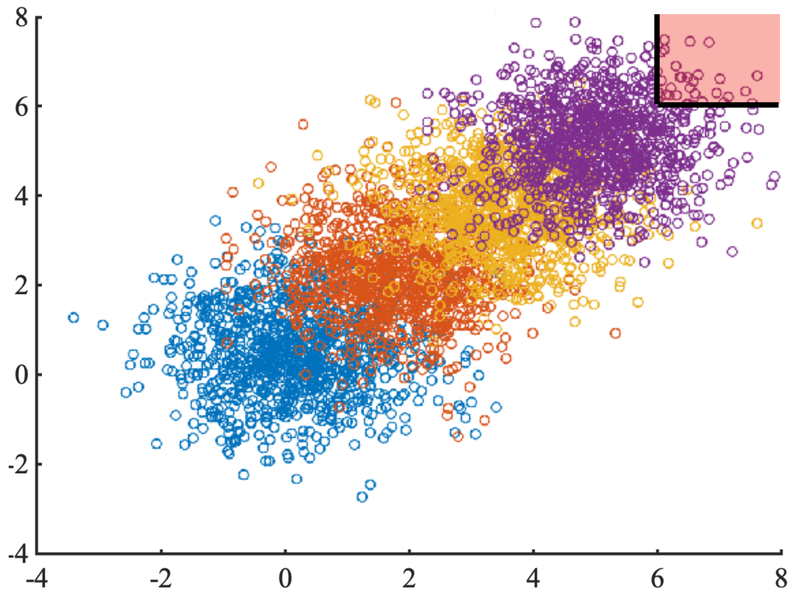


Warm up: cross-entropy method

Goal: find a good parametric importance-sampling distribution

- Explore: draw samples from current sampling distribution
- Exploit: update sampling distribution

Sampling family: $\mathcal{N}(\theta, 1)$

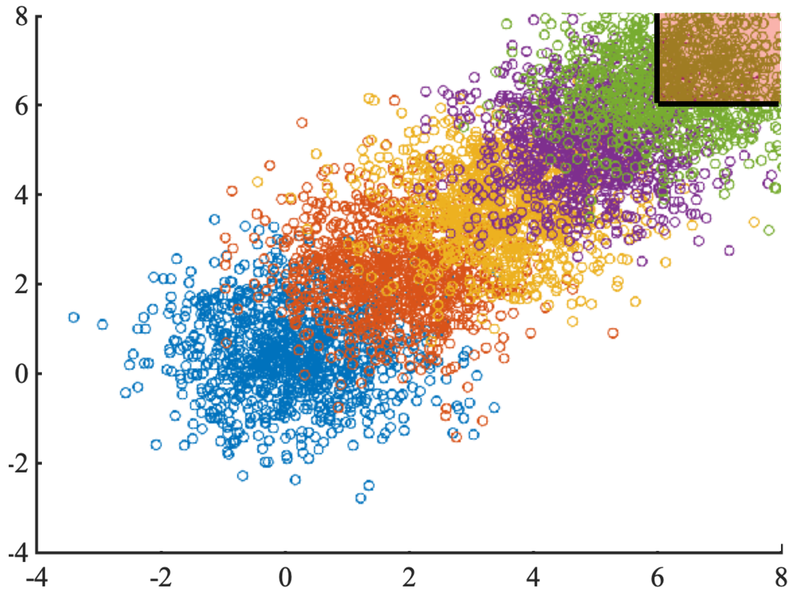


Warm up: cross-entropy method

Goal: find a good parametric importance-sampling distribution

- Explore: draw samples from current sampling distribution
- Exploit: update sampling distribution

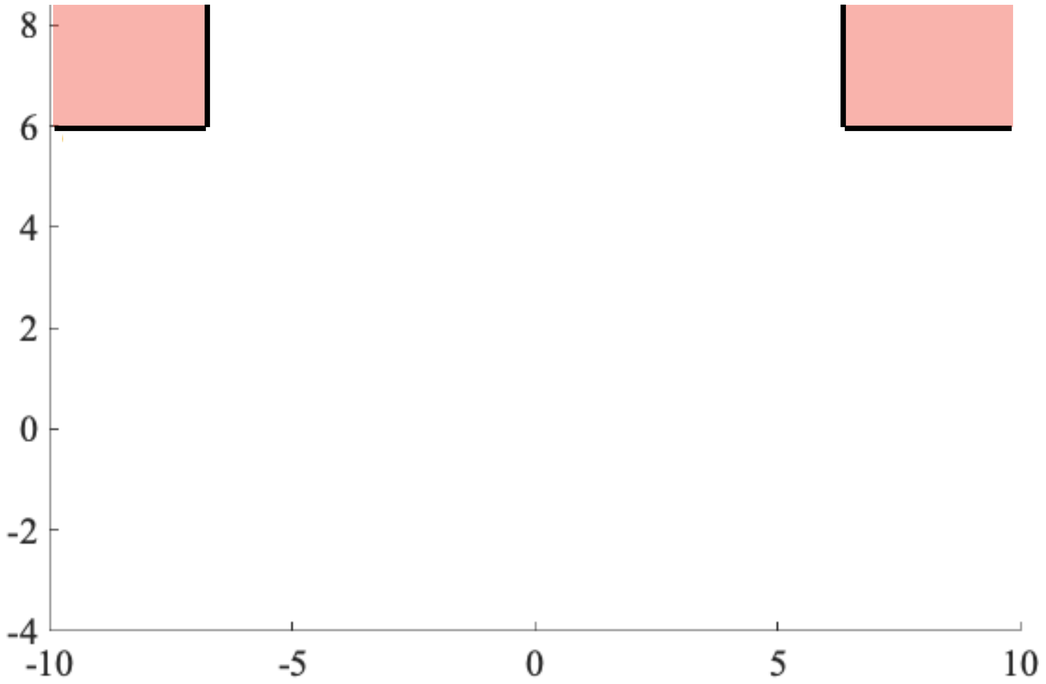
Sampling family: $\mathcal{N}(\theta, 1)$



Warm up: cross-entropy method

Downside: need to know a good parametrization of samplers

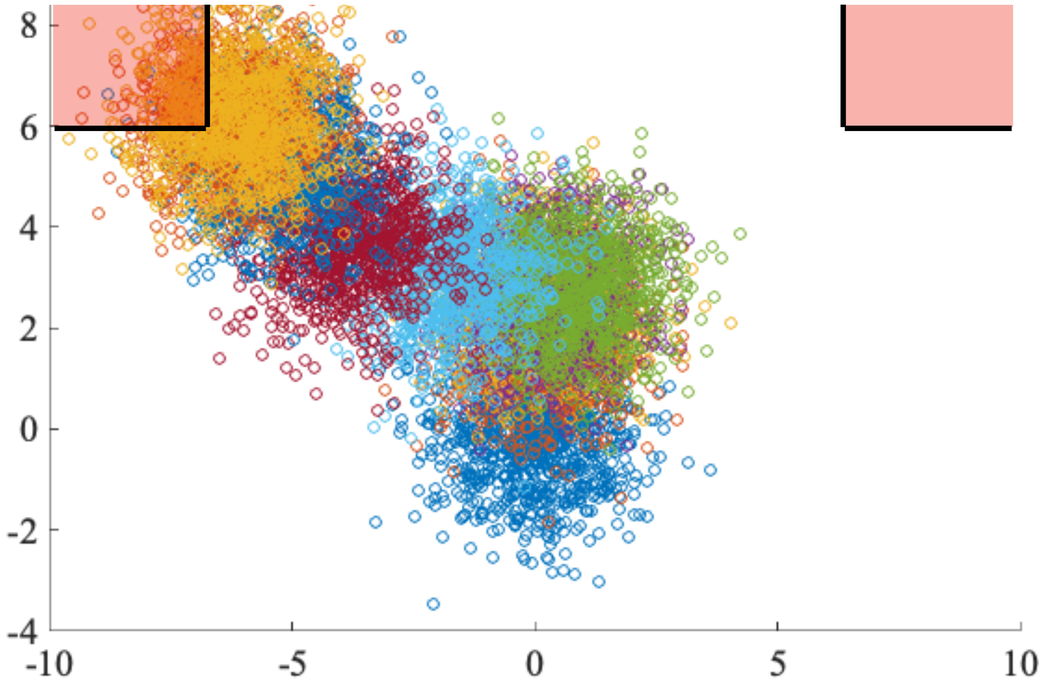
$$f(x) = -\min(|x_{[1]}|, x_{[2]})$$



Warm up: cross-entropy method

Downside: need to know a good parametrization of samplers

$$f(x) = -\min(|x_{[1]}|, x_{[2]})$$



$\mathcal{N}(\theta, 1)$ family fails catastrophically!

Warm up 2: adaptive multilevel splitting (AMS)

Goal: decompose rare probability into a ladder of non-rare probabilities

- Exploit: throw away worst samples and rejuvenate from the remaining good ones
- Explore: MCMC to sample from updated level

$$\mathbb{P}_0(f(X) < \gamma) = \prod_{k=1}^{K} \mathbb{P}(f(X) < L_k | f(X) < L_{k-1})$$

$$\infty =: L_0 > L_1 \ldots > L_K := \gamma$$

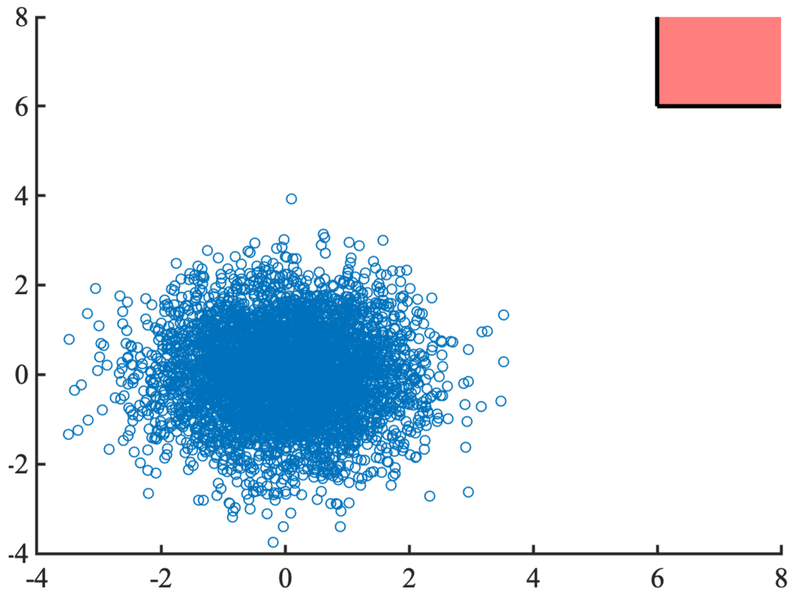
[Guyader et al. 2011, Bréhier et al. 2015, Web et al. 2018, Norden et al. 2020]

Warm up 2: adaptive multilevel splitting (AMS)

Goal: decompose rare probability into a ladder of non-rare probabilities

- Exploit: throw away worst samples and rejuvenate from the remaining good ones
- Explore: MCMC to sample from updated level

$$\mathbb{P}_0(f(X) < \gamma) = \prod_{k=1}^{K} \mathbb{P}(f(X) < L_k | f(X) < L_{k-1})$$



Warm up 2: adaptive multilevel splitting (AMS)

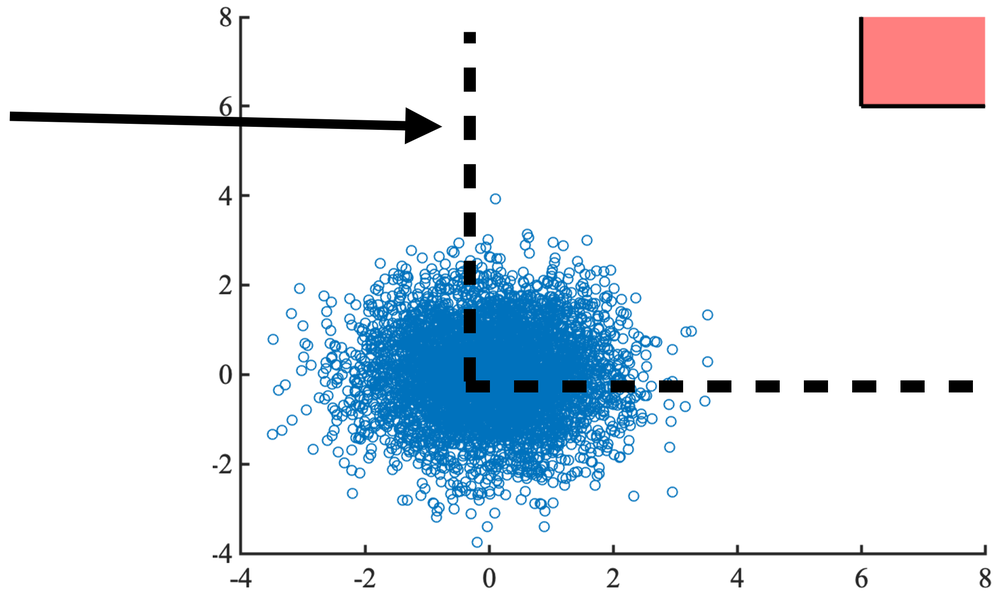
Goal: decompose rare probability into a ladder of non-rare probabilities

- Exploit: throw away worst samples and rejuvenate from the remaining good ones
- Explore: MCMC to sample from updated level

$$\mathbb{P}_0(f(X) < \gamma) = \prod_{k=1}^{K} \mathbb{P}(f(X) < L_k | f(X) < L_{k-1})$$

Empirical $(1-\delta)$-quantile

This is level L_1

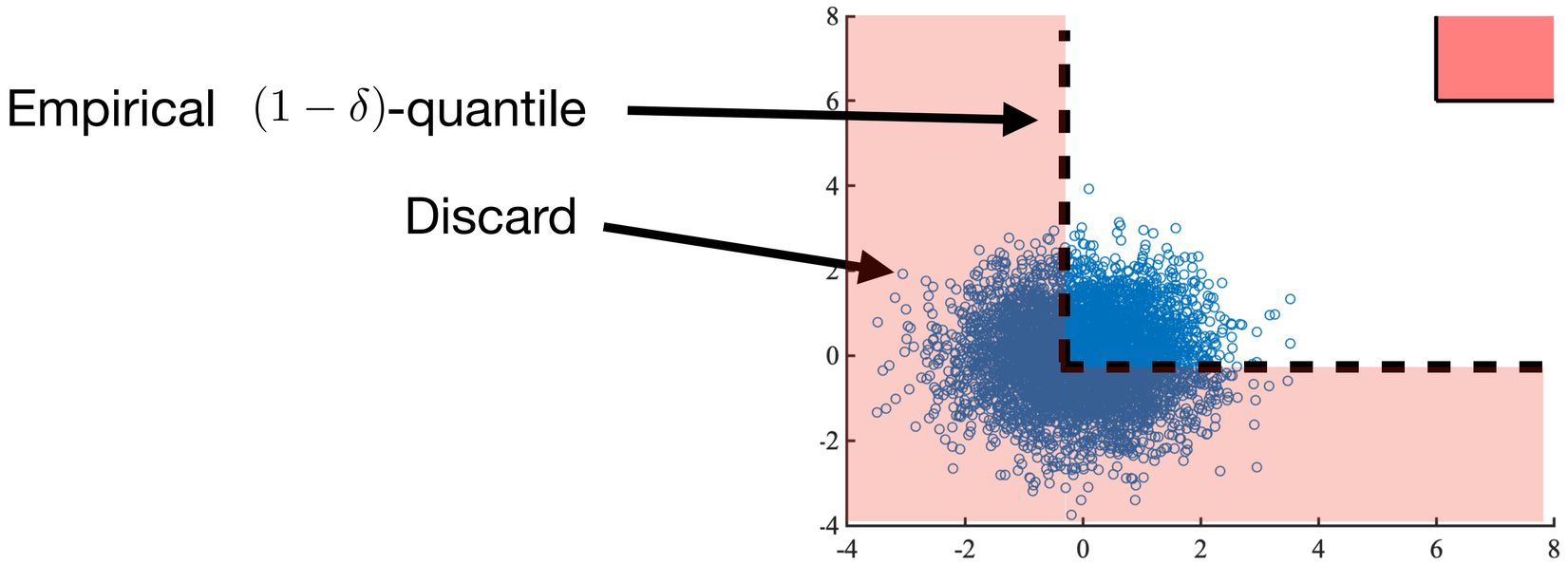


Warm up 2: adaptive multilevel splitting (AMS)

Goal: decompose rare probability into a ladder of non-rare probabilities

- Exploit: throw away worst samples and rejuvenate from the remaining good ones
- Explore: MCMC to sample from updated level

$$\mathbb{P}_0(f(X) < \gamma) = \prod_{k=1}^{K} \mathbb{P}(f(X) < L_k | f(X) < L_{k-1})$$

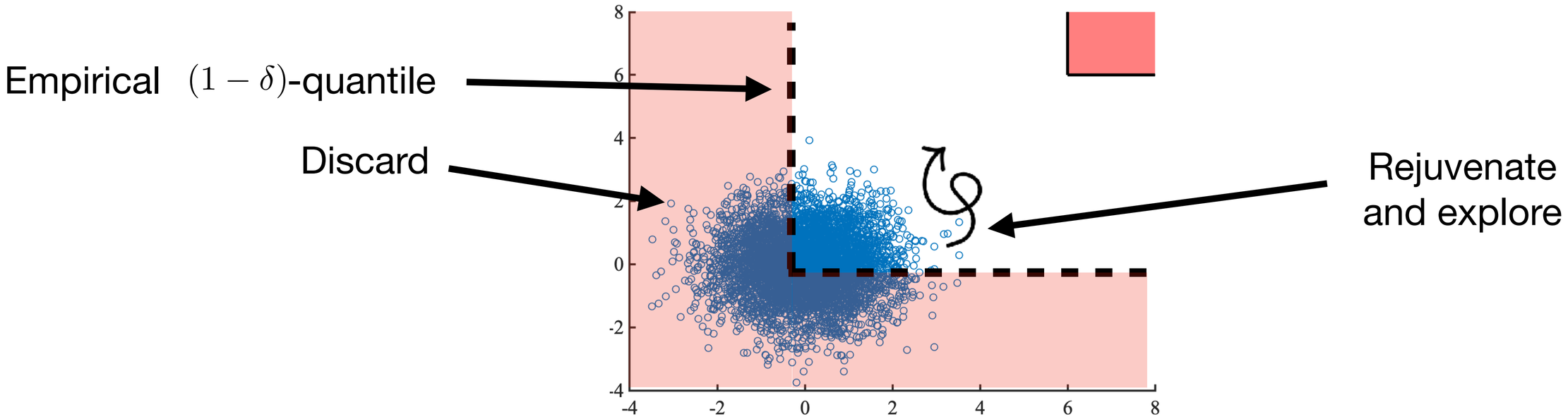


Warm up 2: adaptive multilevel splitting (AMS)

Goal: decompose rare probability into a ladder of non-rare probabilities

- Exploit: throw away worst samples and rejuvenate from the remaining good ones
- Explore: MCMC to sample from updated level

$$\mathbb{P}_0(f(X) < \gamma) = \prod_{k=1}^{K} \mathbb{P}(f(X) < L_k | f(X) < L_{k-1})$$

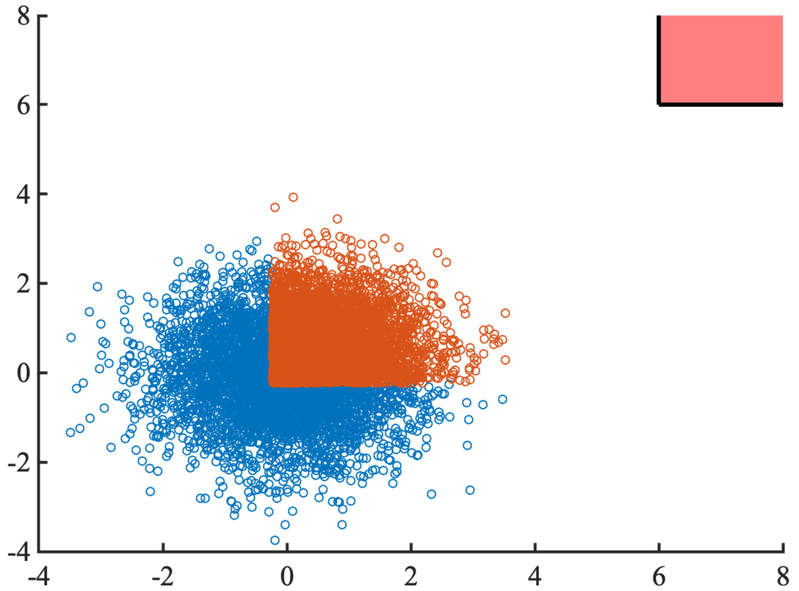


Warm up 2: adaptive multilevel splitting (AMS)

Goal: decompose rare probability into a ladder of non-rare probabilities

- Exploit: throw away worst samples and rejuvenate from the remaining good ones
- Explore: MCMC to sample from updated level

$$\mathbb{P}_0(f(X) < \gamma) = \prod_{k=1}^{K} \mathbb{P}(f(X) < L_k | f(X) < L_{k-1})$$

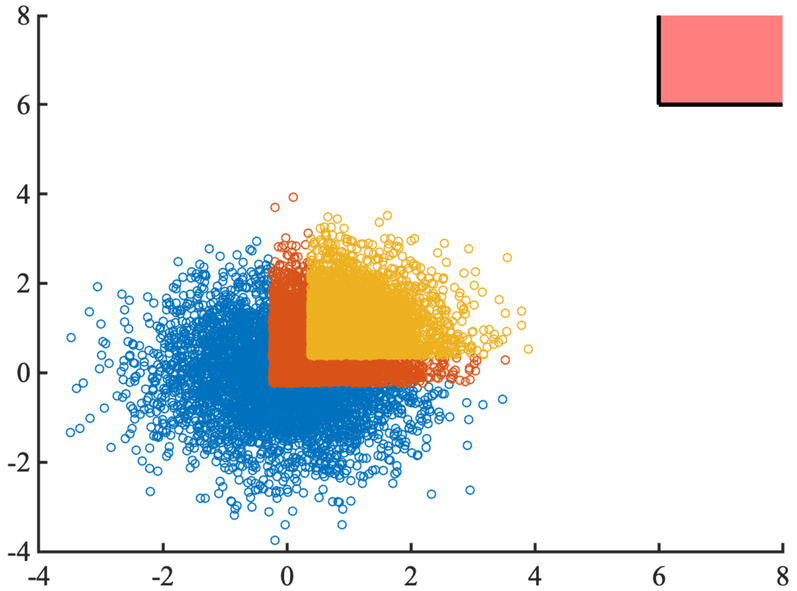


Warm up 2: adaptive multilevel splitting (AMS)

Goal: decompose rare probability into a ladder of non-rare probabilities

- Exploit: throw away worst samples and rejuvenate from the remaining good ones
- Explore: MCMC to sample from updated level

$$\mathbb{P}_0(f(X) < \gamma) = \prod_{k=1}^{K} \mathbb{P}(f(X) < L_k | f(X) < L_{k-1})$$

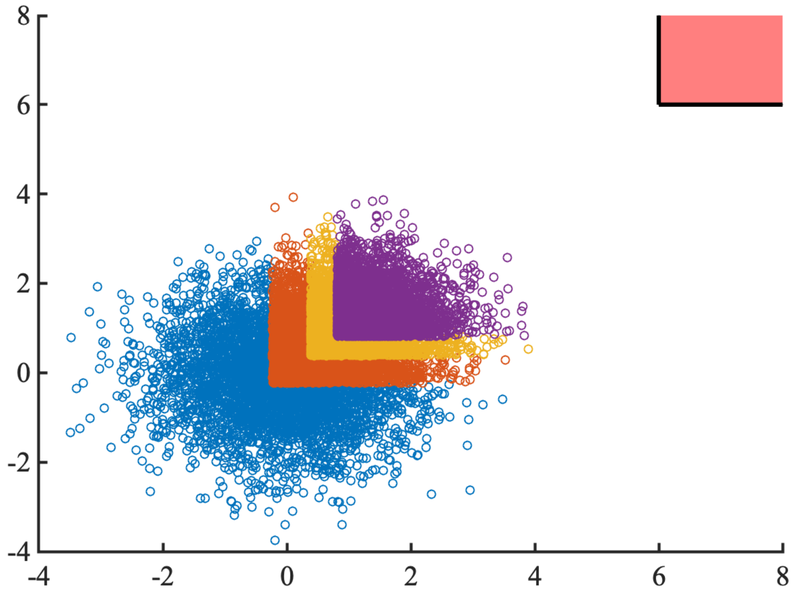


Warm up 2: adaptive multilevel splitting (AMS)

Goal: decompose rare probability into a ladder of non-rare probabilities

- Exploit: throw away worst samples and rejuvenate from the remaining good ones
- Explore: MCMC to sample from updated level

$$\mathbb{P}_0(f(X) < \gamma) = \prod_{k=1}^{K} \mathbb{P}(f(X) < L_k | f(X) < L_{k-1})$$



Warm up 2: adaptive multilevel splitting (AMS)

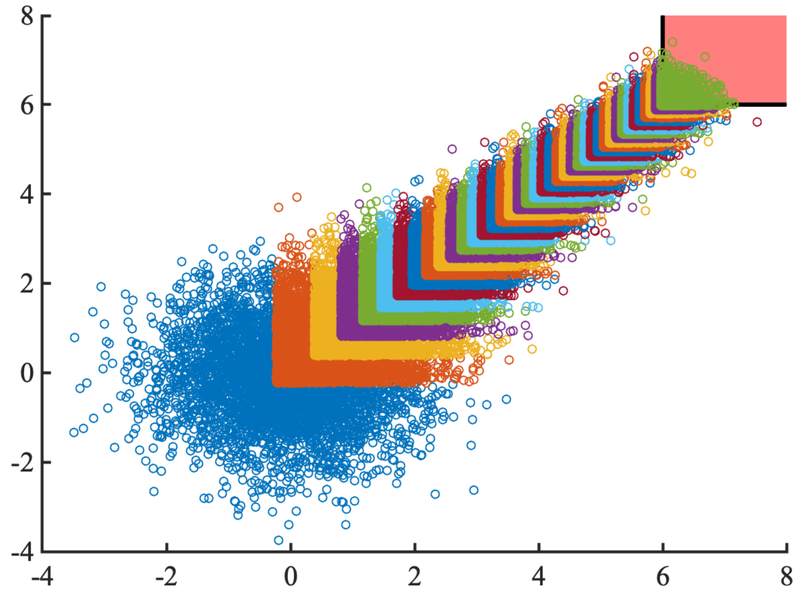
Goal: decompose rare probability into a ladder of non-rare probabilities

- Exploit: throw away worst samples and rejuvenate from the remaining good ones
- Explore: MCMC to sample from updated level

$$\mathbb{P}_0(f(X) < \gamma) = \prod_{k=1}^{K} \mathbb{P}(f(X) < L_k | f(X) < L_{k-1})$$

Random stopping time K

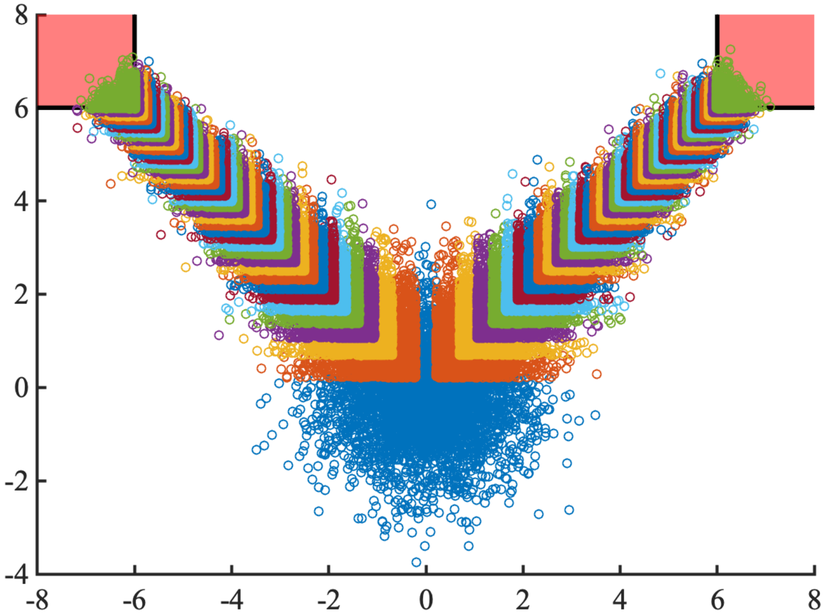
$$\hat{p}_\gamma = (1-\delta)^K$$



Warm up 2: adaptive multilevel splitting (AMS)

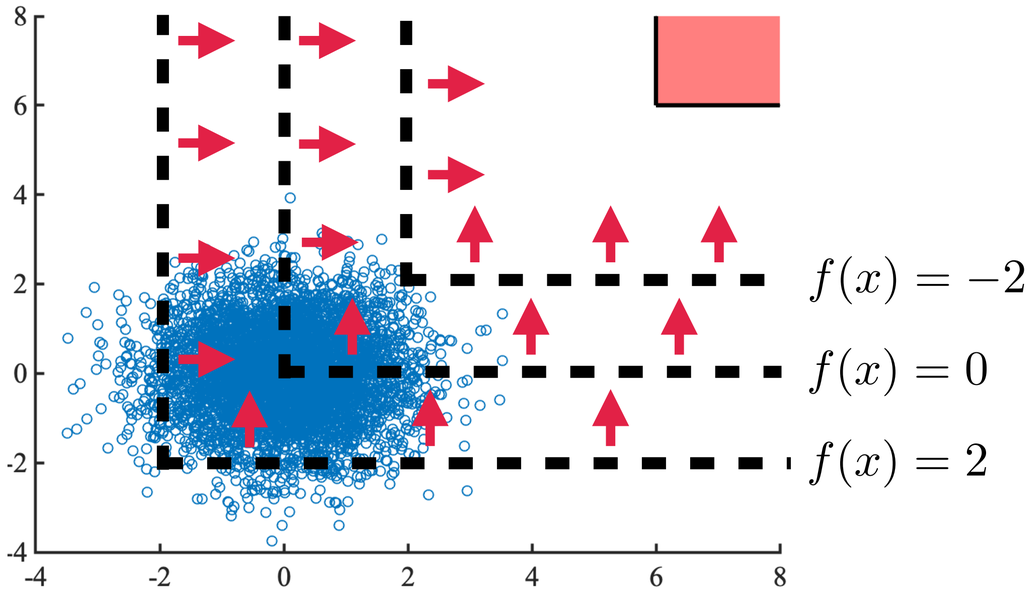
Upside

No need to come up with parametric distributions



Downside

Doesn’t use gradient information $\nabla f(x)$



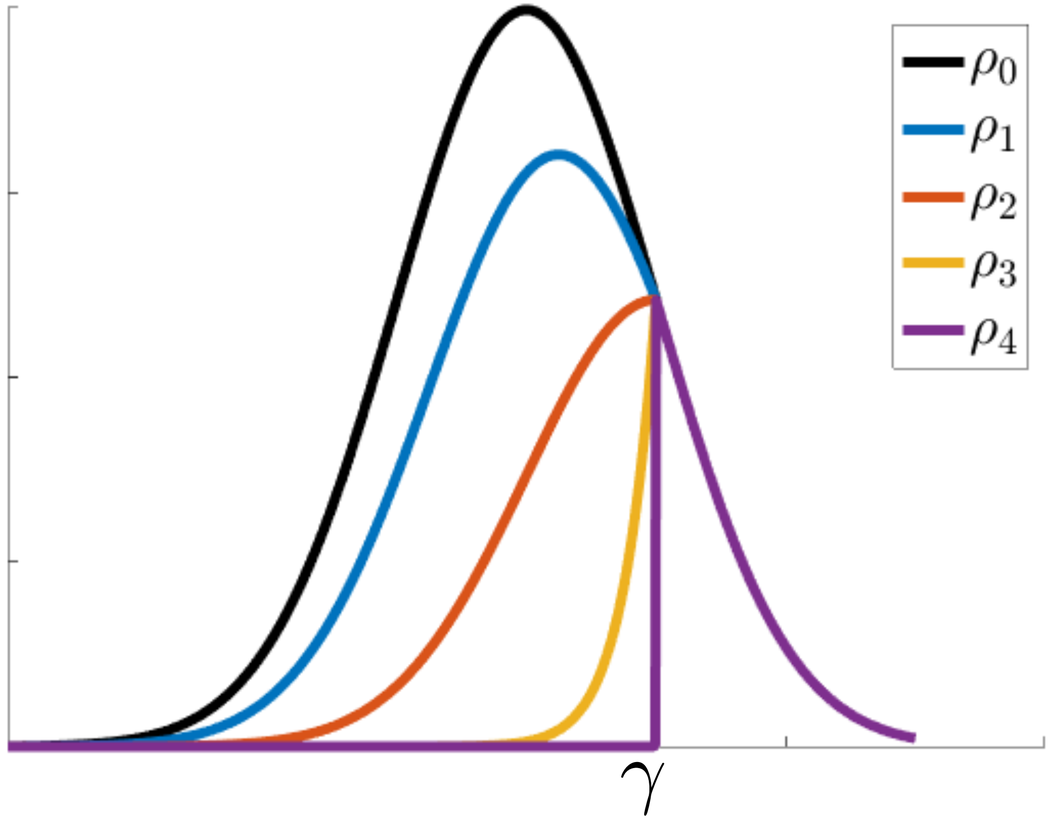
Challenge: intermediate distributions don’t depend smoothly on f

Our approach

- A smoother ladder towards failure

$$\rho_k(x) := \rho_0(x) \underbrace{\exp\left(-\beta_k \left[f(x) - \gamma\right]_+\right)}_{\text{exponential barrier}}$$

$$Z_k := \int_{\mathcal{X}} \rho_k(x) dx$$



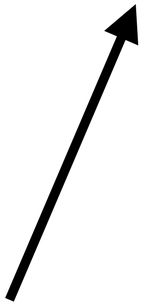
- Contrast with AMS $\rho_k(x) := \rho_0(x) \underbrace{I\{f(x) < L_k\}}_{\text{hard barrier}}$

Our approach

- A smoother ladder towards failure

$$\rho_k(x) := \rho_0(x) \underbrace{\exp\left(-\beta_k \left[f(x) - \gamma\right]_+\right)}_{\text{exponential barrier}}, \quad Z_k := \int_{\mathcal{X}} \rho_k(x) dx$$

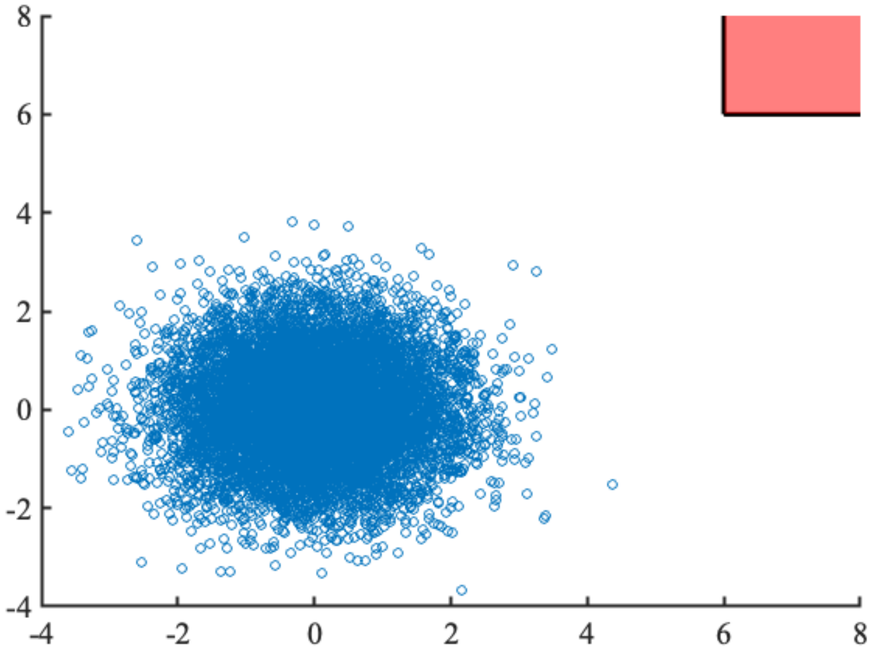
$$\mathbb{P}_0(f(X) < \gamma) = \mathbb{E}_{P_K}\left[\frac{Z_K}{Z_0}\frac{\rho_0(X)}{\rho_K(X)} I\{f(X) < \gamma\}\right], \qquad \frac{Z_K}{Z_0} = \prod_{k=1}^{K} \frac{Z_k}{Z_{k-1}}$$



We will estimate these

Our approach

- A smoother ladder towards failure
- Exploit: determine the next β using current samples (k^{th} distribution)
- Explore + optimize: utilize gradient-based MCMC to sample from $(k+1)^{st}$ distribution
- Estimate: compute Z_{k+1}/Z_k via bridge sampling



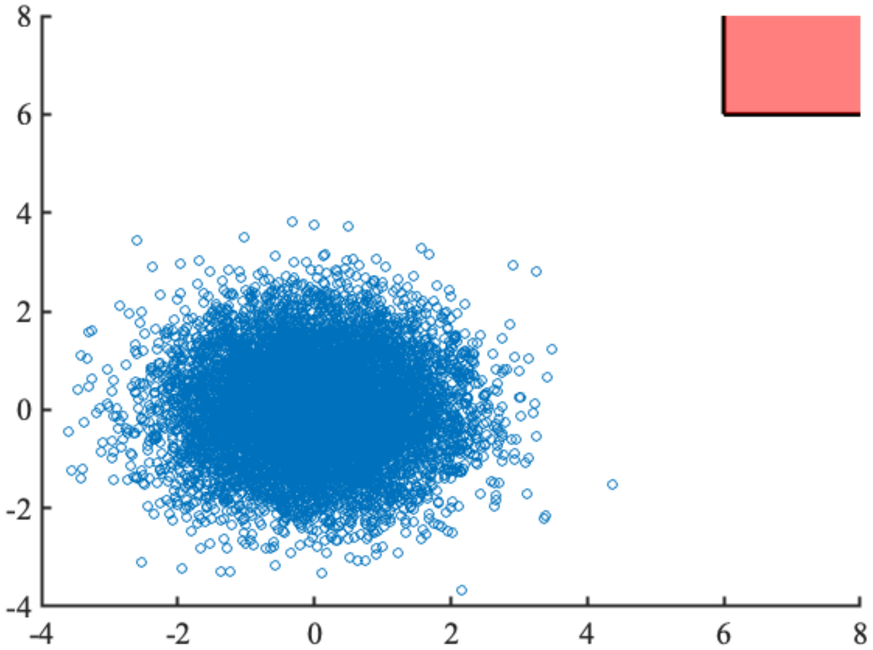
Our approach

- A smoother ladder towards failure
- Exploit: determine the next β using current samples (k^{th} distribution)
- Explore + optimize: utilize gradient-based MCMC to sample from $(k+1)^{st}$ distribution
- Estimate: compute Z_{k+1}/Z_k via bridge sampling

Choose β_{k+1} such that

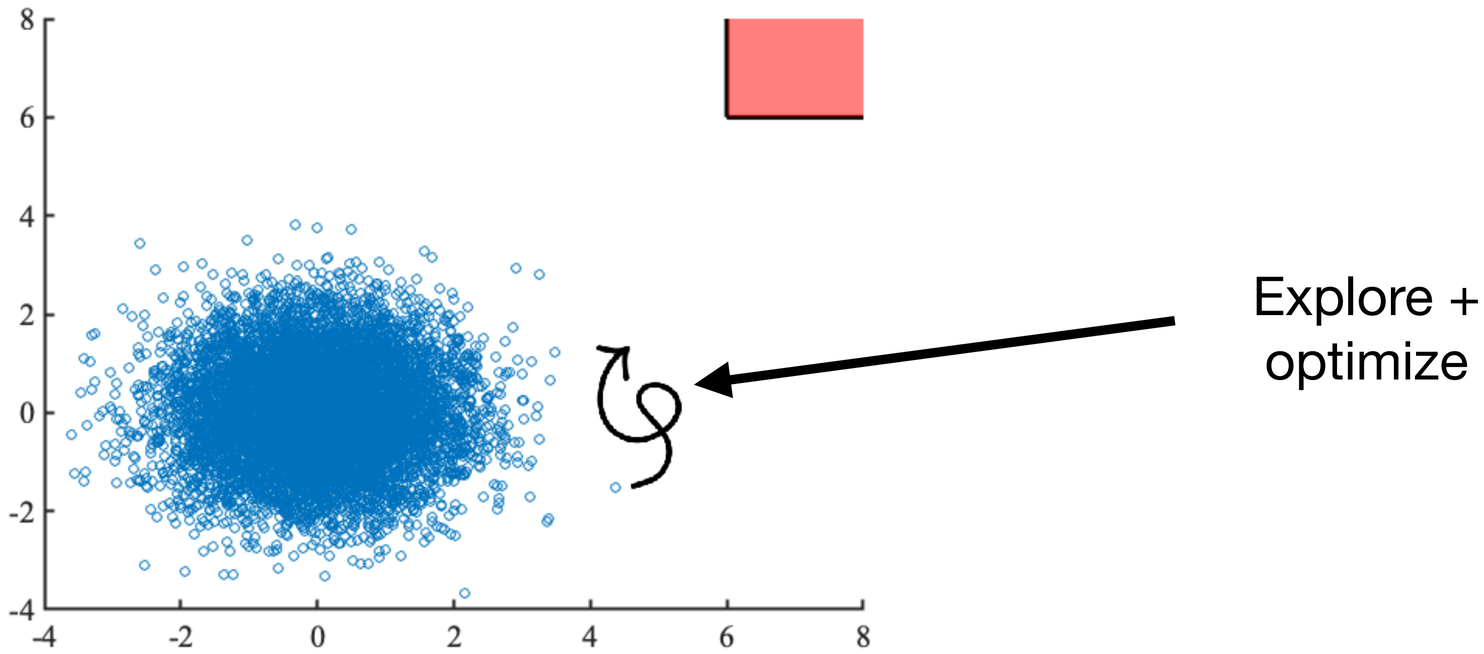
$$\frac{Z_{k+1}}{Z_k} \approx \alpha$$

(quasiconvex problem)



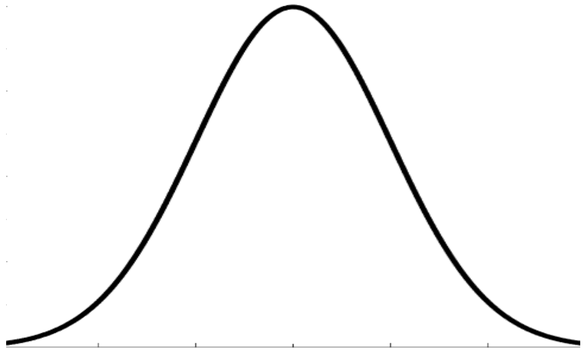
Our approach

- A smoother ladder towards failure
- Exploit: determine the next β using current samples (k^{th} distribution)
- Explore + optimize: utilize gradient-based MCMC to sample from $(k+1)^{st}$ distribution
- Estimate: compute Z_{k+1}/Z_k via bridge sampling

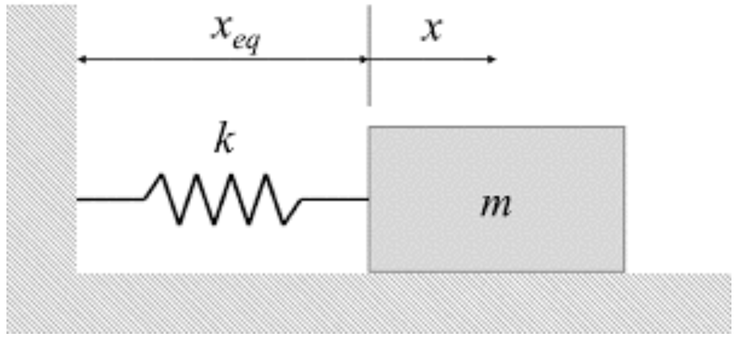


Hamiltonian Monte Carlo (HMC)

- Treat $-\log \rho_k$ as a physical energy potential and simulate the dynamics (ODEs)



Gaussian distribution



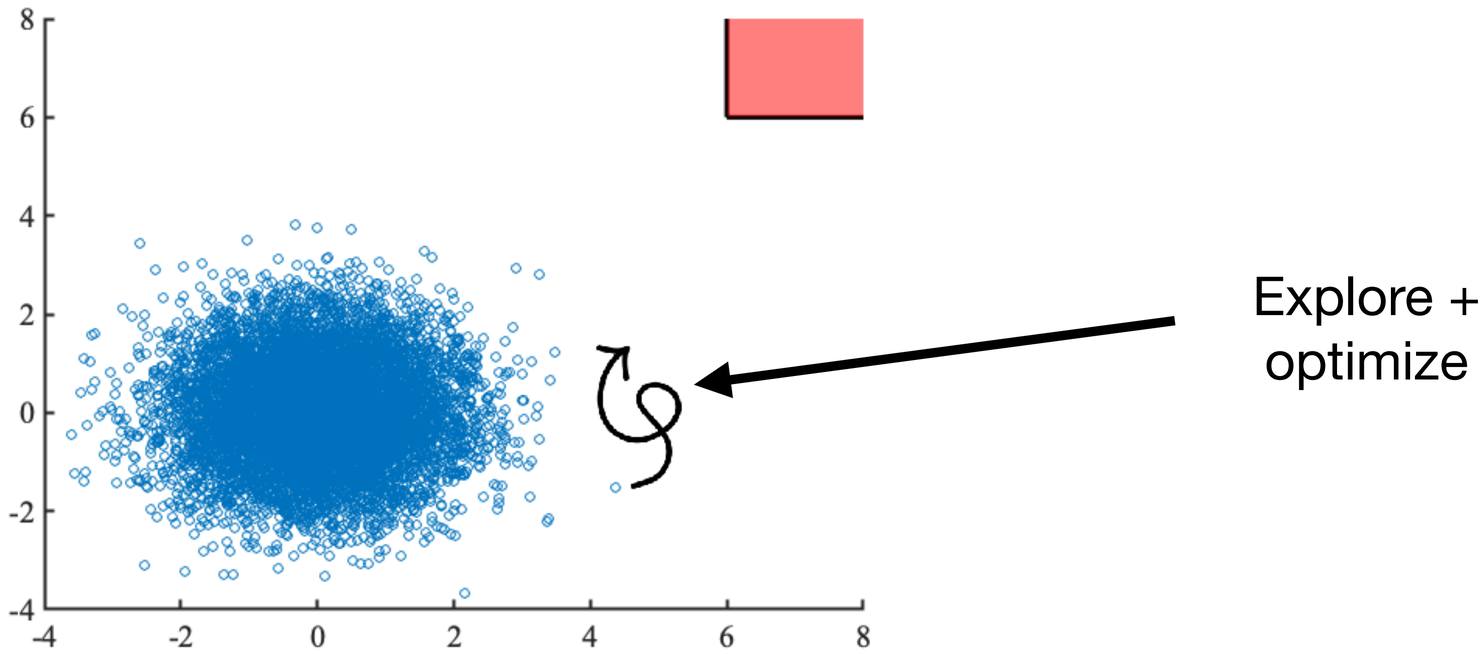
Spring-mass system

- Automatic tradeoff between exploration and optimization

$$\nabla \log \rho_k(x) = \underbrace{\nabla \log \rho_0(x)}_{\text{exploration}} - \beta_k \underbrace{\nabla f(x) I\{f(x) > \gamma\}}_{\text{optimization}}$$

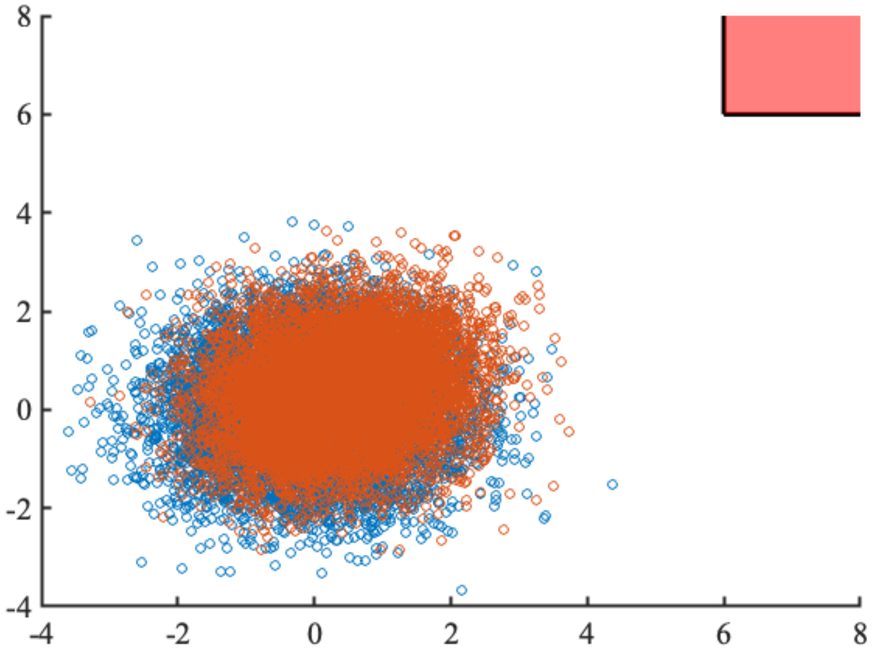
Our approach

- A smoother ladder towards failure
- Exploit: determine the next β using current samples (k^{th} distribution)
- Explore + optimize: utilize gradient-based MCMC to sample from $(k+1)^{st}$ distribution
- Estimate: compute Z_{k+1}/Z_k via bridge sampling



Our approach

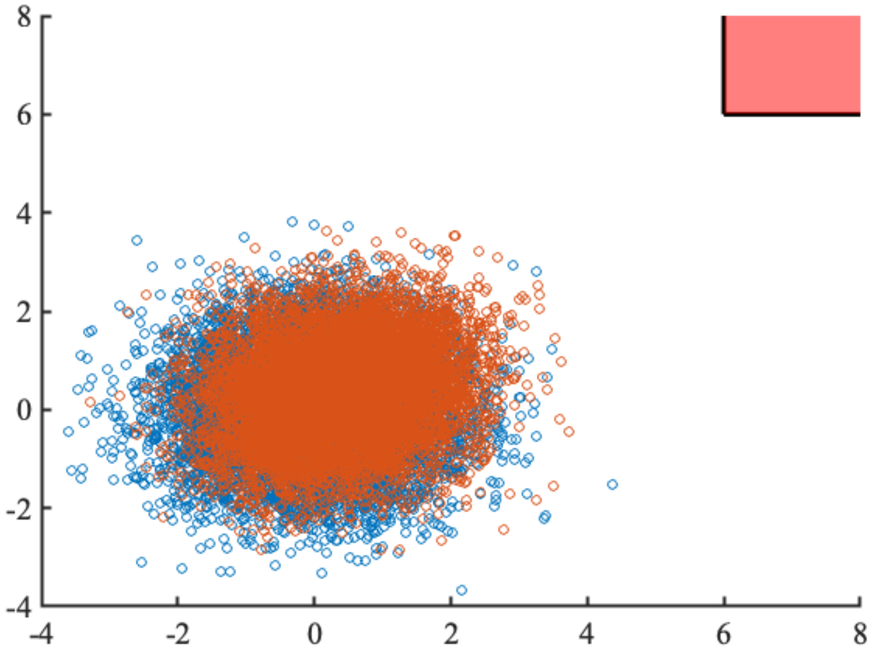
- A smoother ladder towards failure
- Exploit: determine the next β using current samples (k^{th} distribution)
- Explore + optimize: utilize gradient-based MCMC to sample from $(k+1)^{st}$ distribution
- Estimate: compute Z_{k+1}/Z_k via bridge sampling



Our approach

- A smoother ladder towards failure
- Exploit: determine the next β using current samples (k^{th} distribution)
- Explore + optimize: utilize gradient-based MCMC to sample from $(k+1)^{st}$ distribution
- Estimate: compute Z_{k+1}/Z_k via bridge sampling

Use both sets of samples to compute an accurate estimate of Z_{k+1}/Z_k

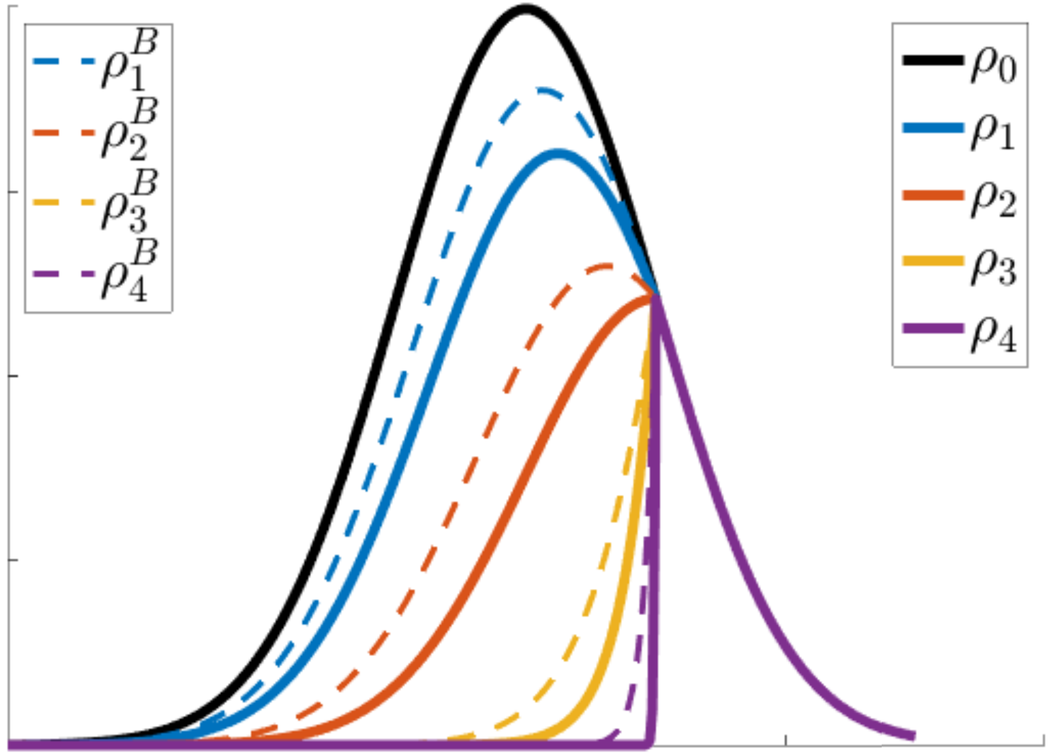


Bridge sampling

- Use samples from neighboring distributions to estimate their ratio Z_{k+1}/Z_k

$$\frac{Z_{k+1}}{Z_k} = \frac{Z_k^B/Z_k}{Z_k^B/Z_{k+1}}$$

Use an auxiliary “bridge” distribution



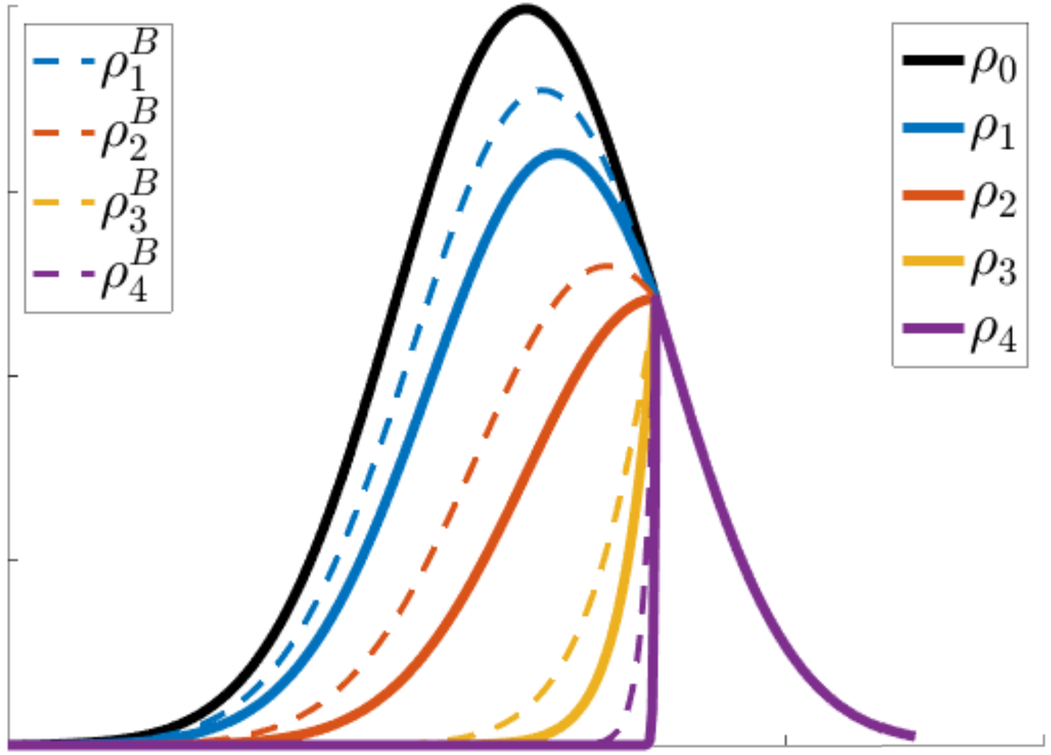
[Bennett 1976, Meng & Wong 1996]

Bridge sampling

- Use samples from neighboring distributions to estimate their ratio Z_{k+1}/Z_k

$$\frac{Z_{k+1}}{Z_k} = \frac{Z_k^B/Z_k}{Z_k^B/Z_{k+1}}$$

Use an auxiliary “bridge” distribution

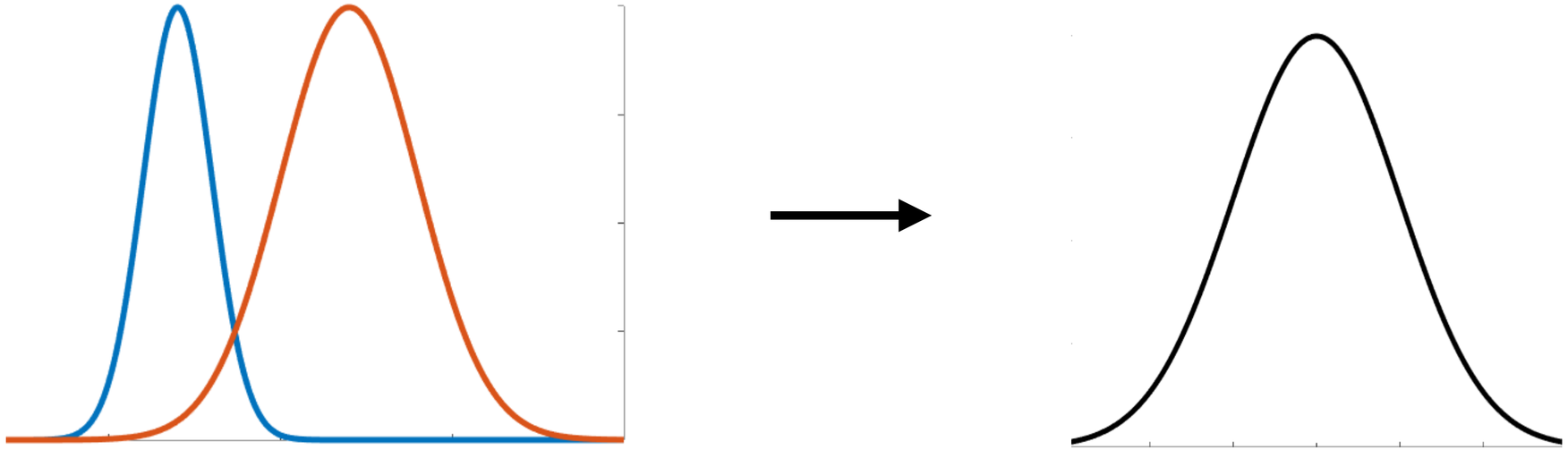


Problem:
Error depends on distance between distributions

[Bennett 1976, Meng & Wong 1996]

Neural warping

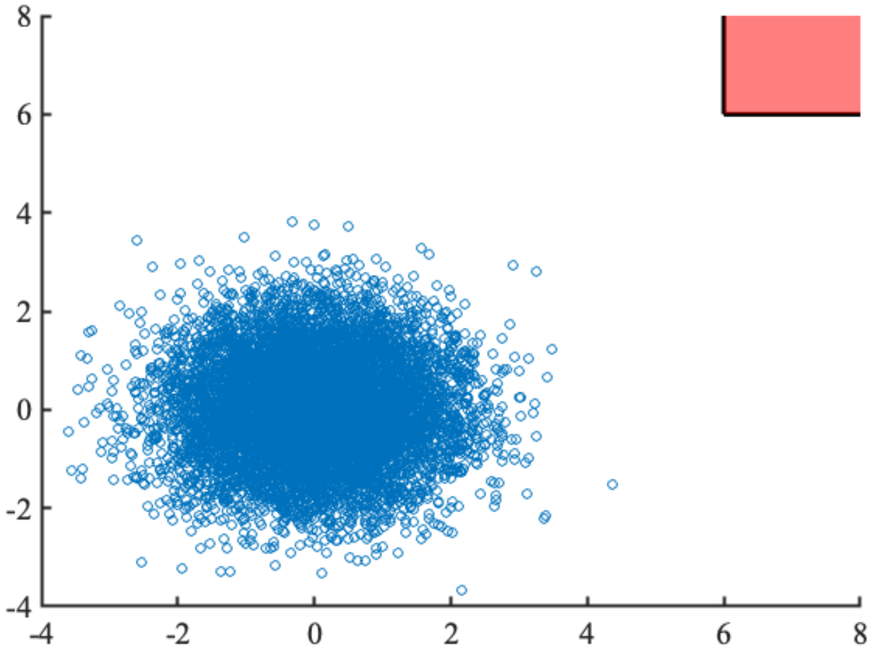
- Error of bridge-sampling estimate depends on the distance between distributions
- Transform the space so they are closer (“warp” the space between them)



- Classical techniques: mean shift, affine scaling [Voter 1985, Meng & Schilling 2002]
- Modern ML toolbox: normalizing flows [Papamakarios et al. 2019]
- Bonus: warping helps HMC [Girolami & Calderhead 2011, Hoffman et al. 2019]

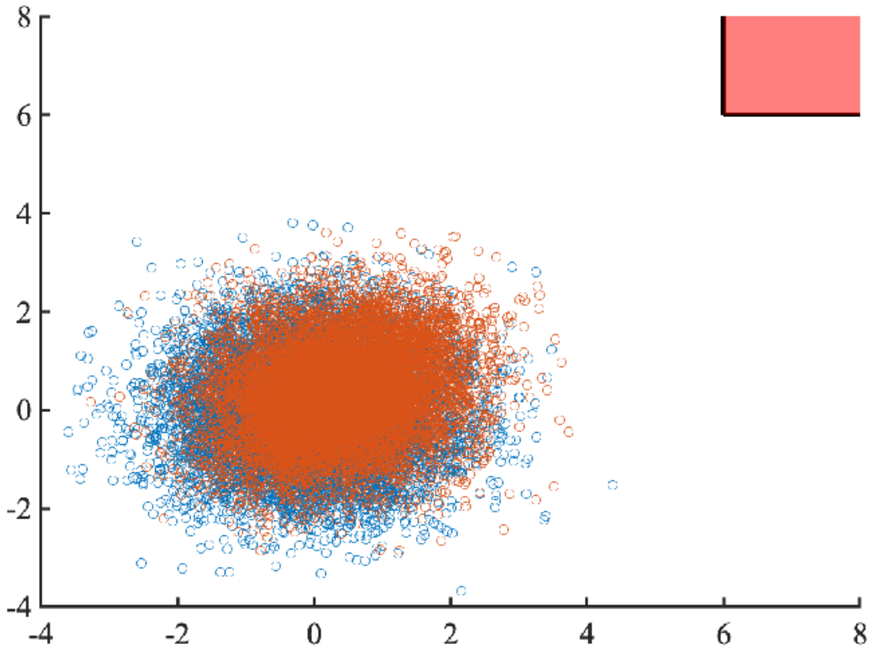
Our approach

- A smoother ladder towards failure
- Exploit: determine the next β using current samples (k^{th} distribution)
- Explore + optimize: utilize gradient-based MCMC to sample from $(k+1)^{st}$ distribution
- Estimate: compute Z_{k+1}/Z_k via bridge sampling



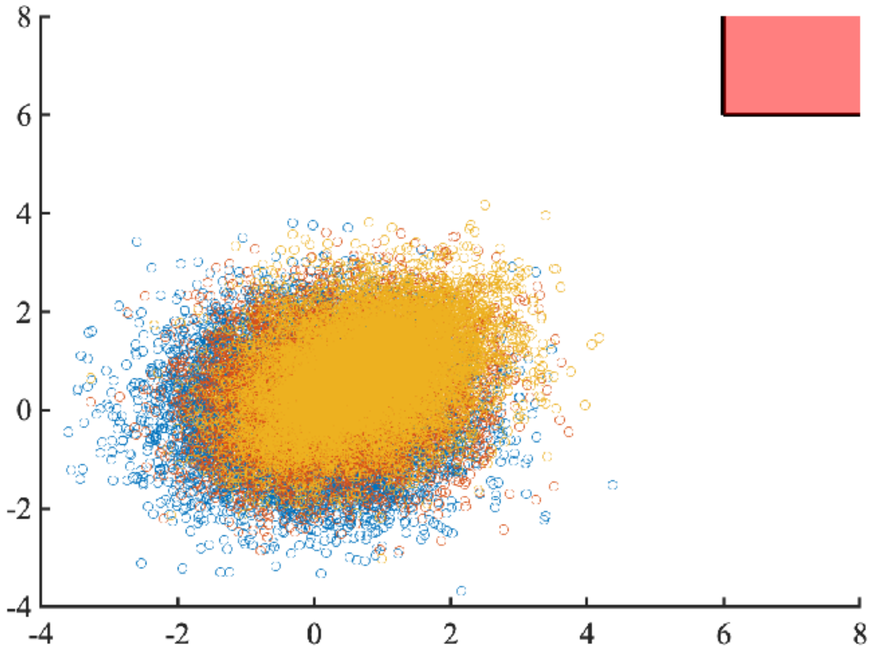
Our approach

- A smoother ladder towards failure
- Exploit: determine the next β using current samples (k^{th} distribution)
- Explore + optimize: utilize gradient-based MCMC to sample from $(k+1)^{st}$ distribution
- Estimate: compute Z_{k+1}/Z_k via bridge sampling



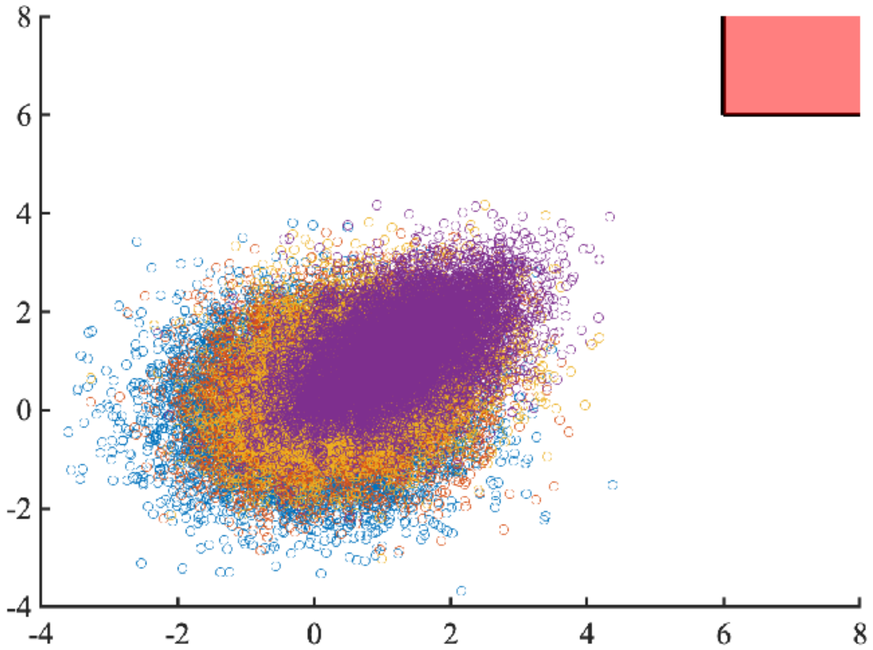
Our approach

- A smoother ladder towards failure
- Exploit: determine the next β using current samples (k^{th} distribution)
- Explore + optimize: utilize gradient-based MCMC to sample from $(k+1)^{st}$ distribution
- Estimate: compute Z_{k+1}/Z_k via bridge sampling



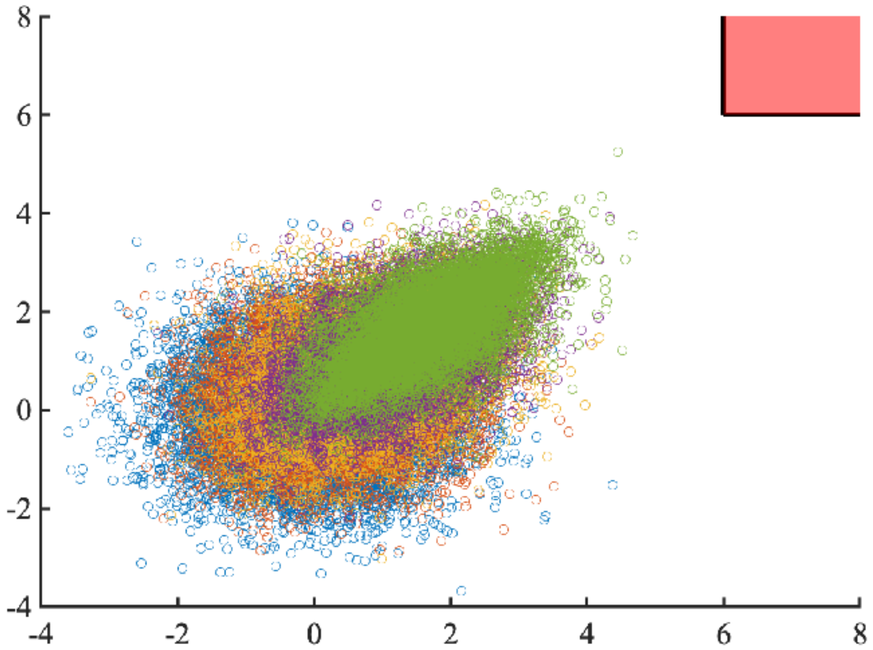
Our approach

- A smoother ladder towards failure
- Exploit: determine the next β using current samples (k^{th} distribution)
- Explore + optimize: utilize gradient-based MCMC to sample from $(k+1)^{st}$ distribution
- Estimate: compute Z_{k+1}/Z_k via bridge sampling



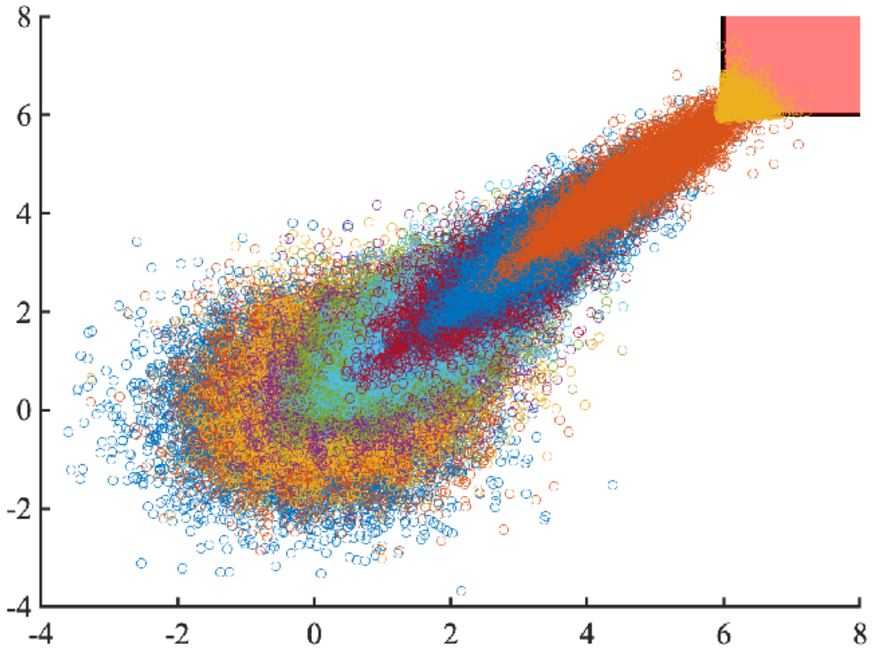
Our approach

- A smoother ladder towards failure
- Exploit: determine the next β using current samples (k^{th} distribution)
- Explore + optimize: utilize gradient-based MCMC to sample from $(k+1)^{st}$ distribution
- Estimate: compute Z_{k+1}/Z_k via bridge sampling



Our approach

- A smoother ladder towards failure
- Exploit: determine the next β using current samples (k^{th} distribution)
- Explore + optimize: utilize gradient-based MCMC to sample from $(k+1)^{st}$ distribution
- Estimate: compute Z_{k+1}/Z_k via bridge sampling

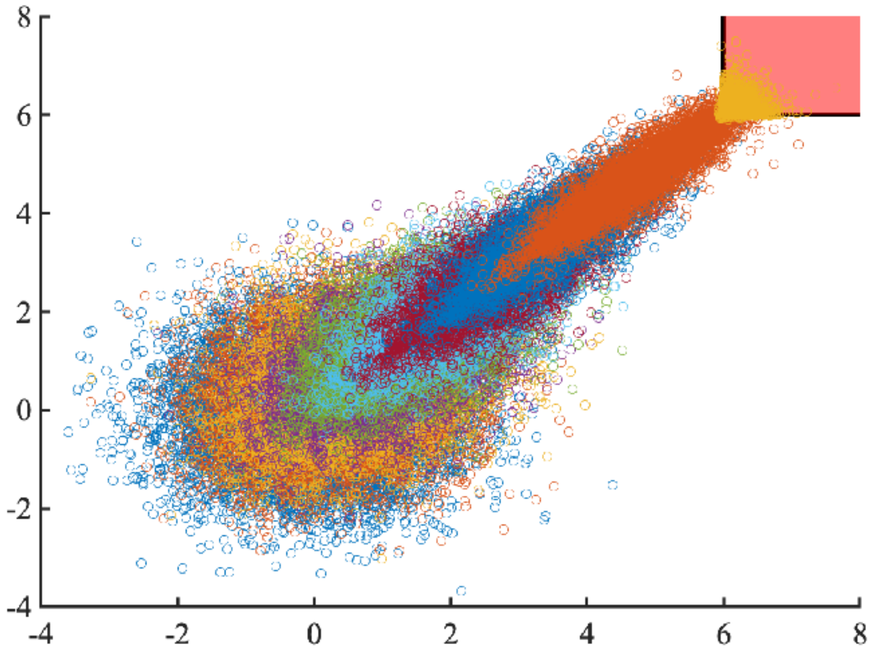


Our approach

- A smoother ladder towards failure
- Exploit: determine the next β using current samples (k^{th} distribution)
- Explore + optimize: utilize gradient-based MCMC to sample from $(k+1)^{st}$ distribution
- Estimate: compute Z_{k+1}/Z_k via bridge sampling

Random stopping time K

$\hat{p}_\gamma \approx \alpha^K$



Performance guarantees

- Theorem: Number of iterations $K \to \log p_\gamma / \log \alpha$ and $\mathbb{E}[(\hat{p}_\gamma / p_\gamma - 1)^2] \leq 2KD/N$
  - D depends on (warped) distance between consecutive distributions

- Computational cost is $O(KN)$ simulations

	Cost	Error
Neural bridge sampling	$N \log(1/p_\gamma)$	$\dfrac{\log(1/p_\gamma)}{N}$
Monte Carlo	N	$\dfrac{1}{p_\gamma N}$

Performance guarantees

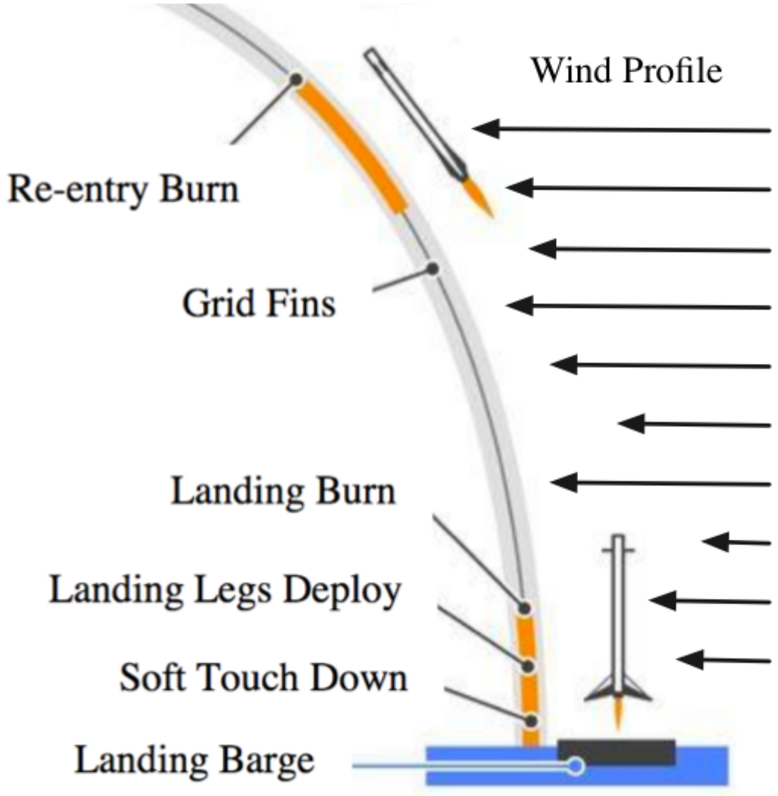
- Overall efficiency gain of $O\left(\frac{1}{p_\gamma \log(p_\gamma)^2}\right)$ over Monte Carlo

- Relative advantage scales with rarity

	Time	Error
Neural bridge sampling	$N \log(1/p_\gamma)$	$\frac{\log(1/p_\gamma)}{N}$
Monte Carlo	N	$\frac{1}{p_\gamma N}$

Experiments: rocket design

- Vertical landing of an orbital-class rocket (e.g. SpaceX Falcon 9)
- P_0 is the model of wind gusts during flight (100 dimensions)
- $f(x)$ is the distance from the center of the launchpad at landing



Rocket1: Boosters capable of 15% main thrust

Rocket2: Boosters capable of 10% main thrust

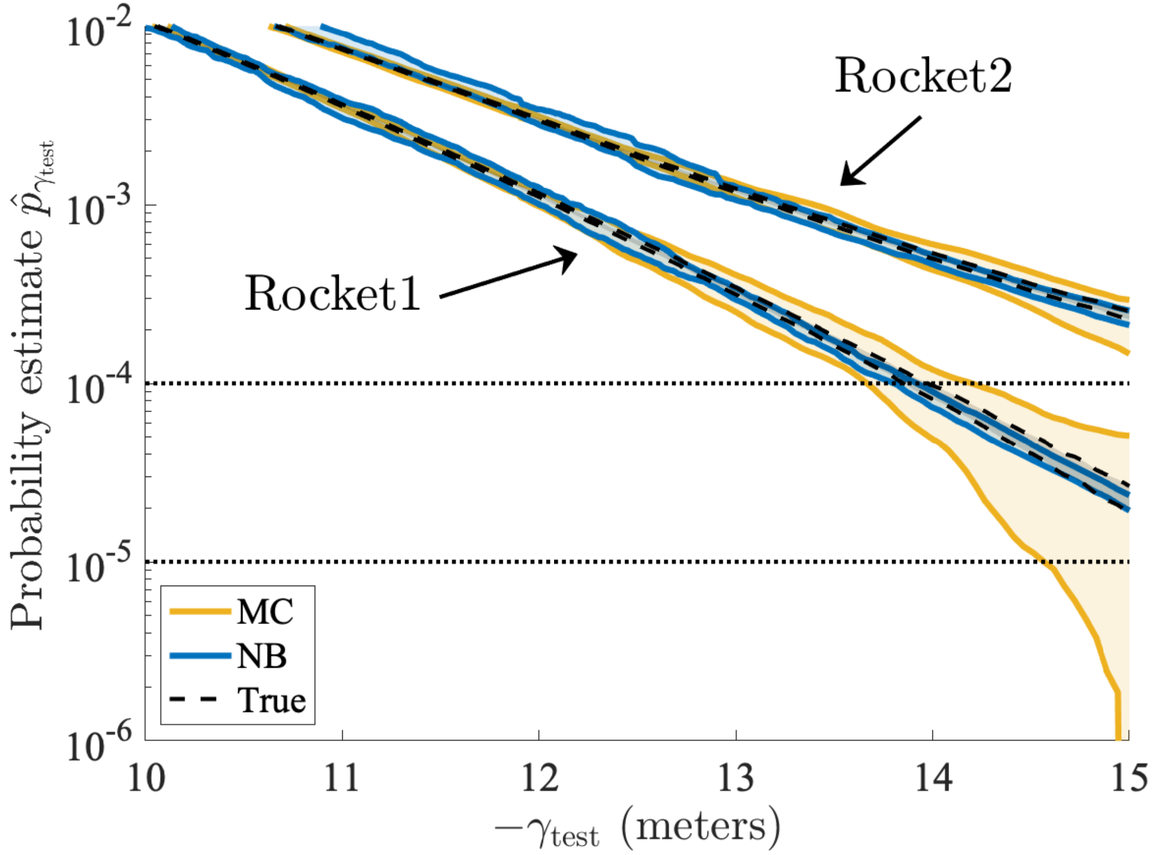
Bigger boosters are safer but mean smaller payloads

Experiments: rocket design

- Vertical landing of an orbital-class rocket (e.g. SpaceX Falcon 9)
- P_0 is the model of wind gusts during flight (100 dimensions)
- $f(x)$ is the distance from the center of the launchpad at landing

MC/NB use 100,000 samples

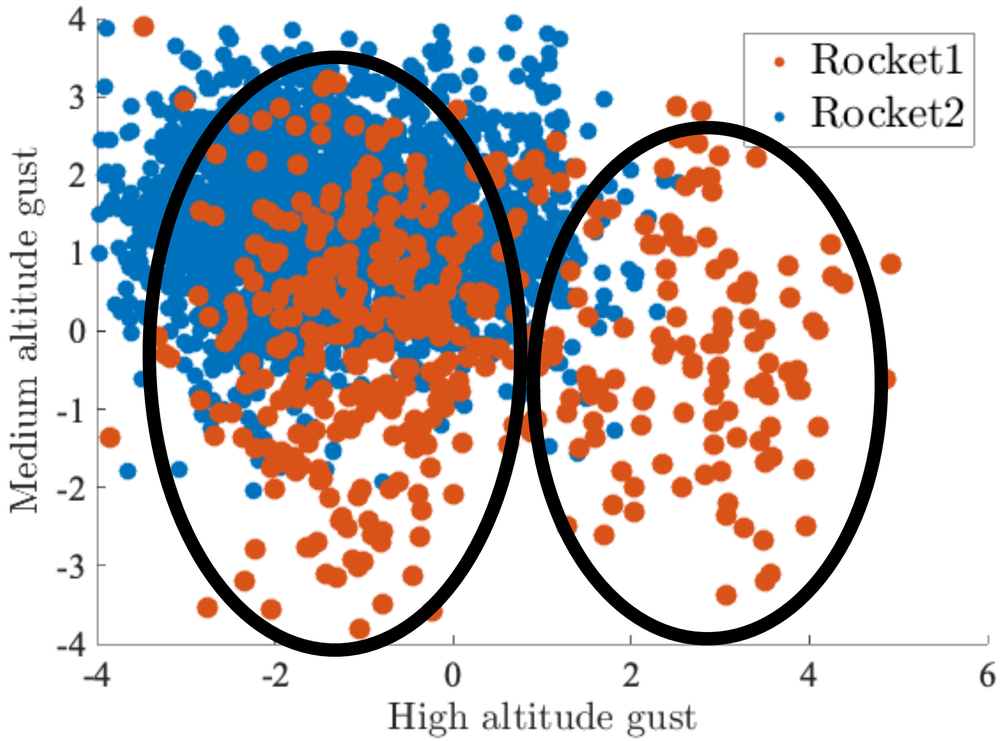
True is MC with 50M samples



Experiments: rocket design

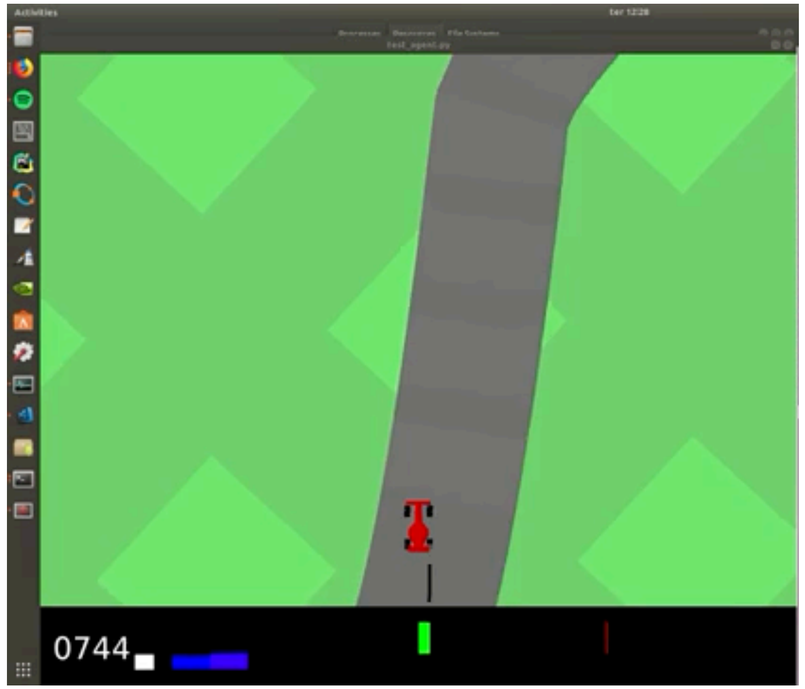
- Vertical landing of an orbital-class rocket (e.g. what SpaceX does)
- P_0 is the model of wind gusts during flight (100 dimensions)
- $f(x)$ is the distance from the center of the launchpad at landing

PCA visualization of failure modes



Experiments: OpenAI CarRacing

- Challenging environment (pixels to actions)
- P_0 is the model for track generation (24 dimensions)
- $f(x)$ is the score achieved

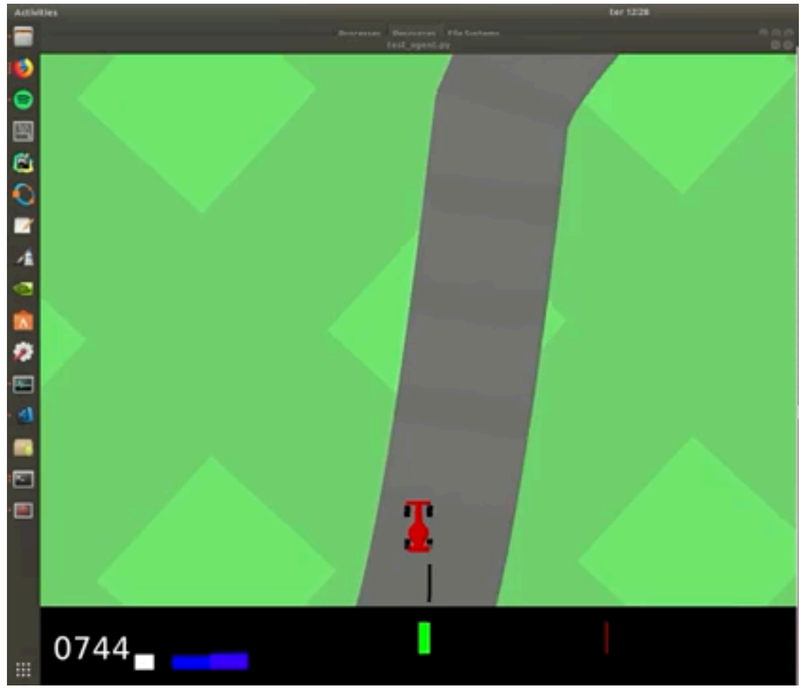


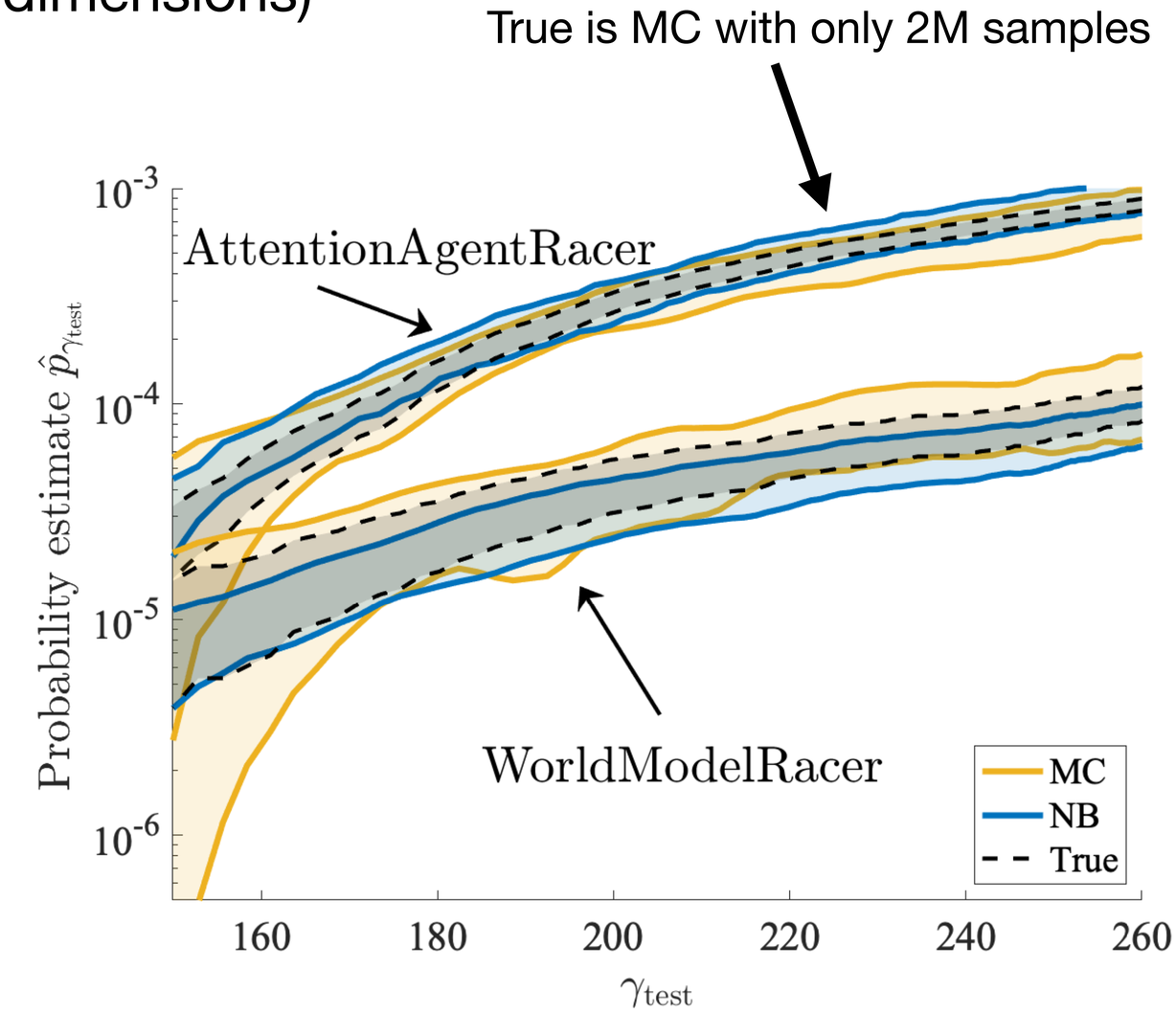
Compare 2 SOTA policies:

	Average Score (over 2M runs)
AttentionAgent [Tang et al. 2020]	903 ± 49
WorldModel [Ha & Schmidhuber 2018]	899 ± 46

Experiments: OpenAI CarRacing

- Challenging environment (pixels to actions)
- P_0 is the model for track generation (24 dimensions)
- $f(x)$ is the score achieved





Experiments

Relative mean-square error $\mathbb{E}[(\hat{p}_\gamma/p_\gamma - 1)^2]$ over 10 trials

	Synthetic	MountainCar	Rocket1	Rocket2	AttentionAgentRacer	WorldModelRacer
MC	1.1821	0.2410	1.1039	0.0865	1.0866	0.9508
AMS	0.0162	0.5424	0.0325	0.0151	1.0211	0.8177
B	0.0514	0.3856	0.0129	0.0323	0.9030	0.7837
NB	**0.0051**	**0.0945**	**0.0102**	**0.0078**	**0.2285**	**0.1218**
p_γ	$3.6 \cdot 10^{-6}$	$1.6 \cdot 10^{-5}$	$2.3 \cdot 10^{-5}$	$2.4 \cdot 10^{-4}$	$\approx 2.5 \cdot 10^{-5}$	$\approx 9.5 \cdot 10^{-6}$

Neural bridge sampling outperforms other methods

Key ideas of this talk

Robustness

Build models with guaranteed performance over uncertainty sets

$$\min_{\theta \in \Theta} \max_{Q \in \mathcal{P}} \mathbb{E}_Q[f(\theta; X)]$$

- With small $\mathcal{P}$, can we solve this quickly?
- What about when $\mathcal{P}$ is large/unknown?

Risk

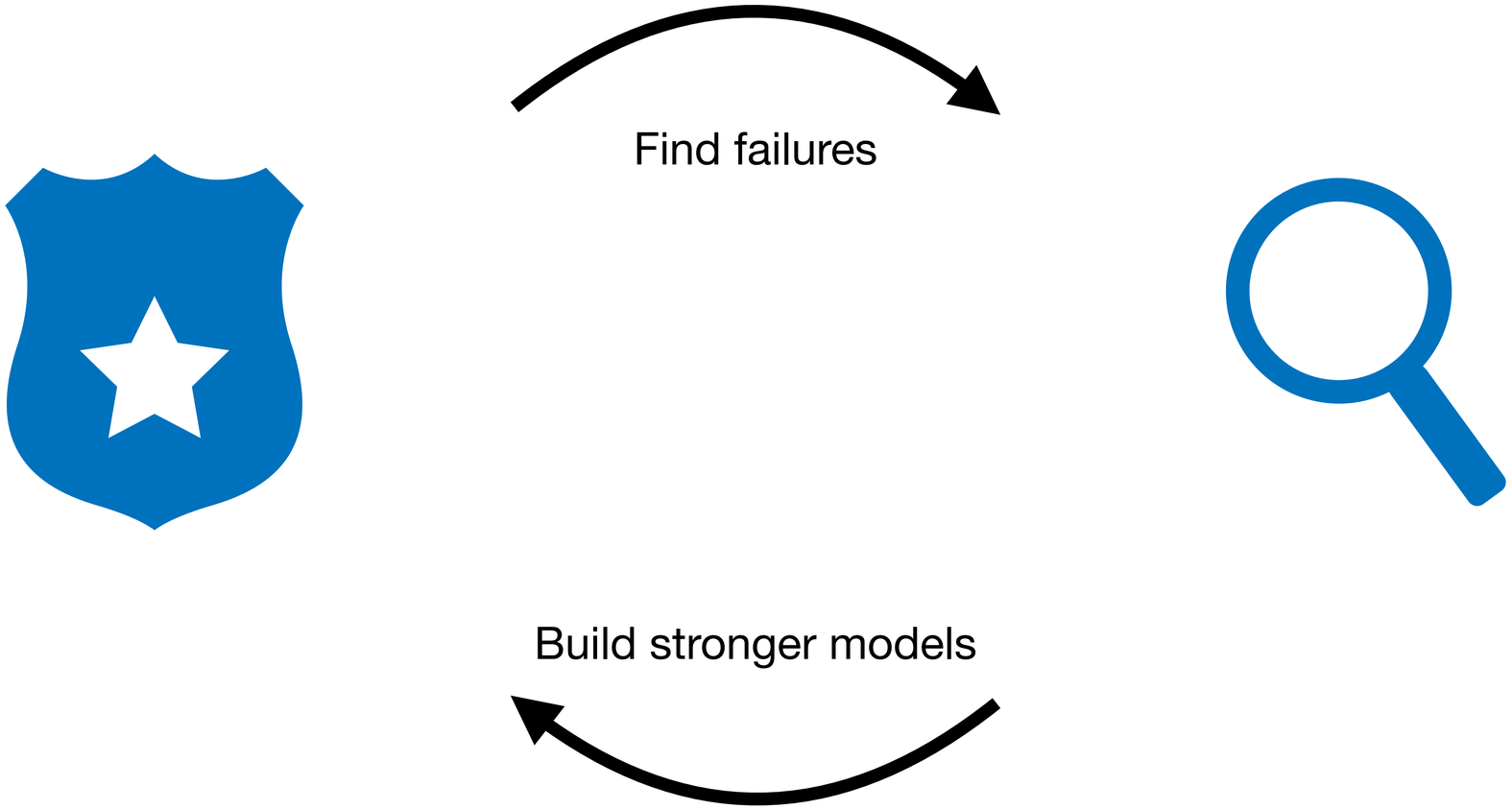
Find failure modes and quantify the probability of failure

$$\mathbb{P}_0(f(X) < \gamma)$$

- Why is this the right problem?
- How do we solve it quickly?

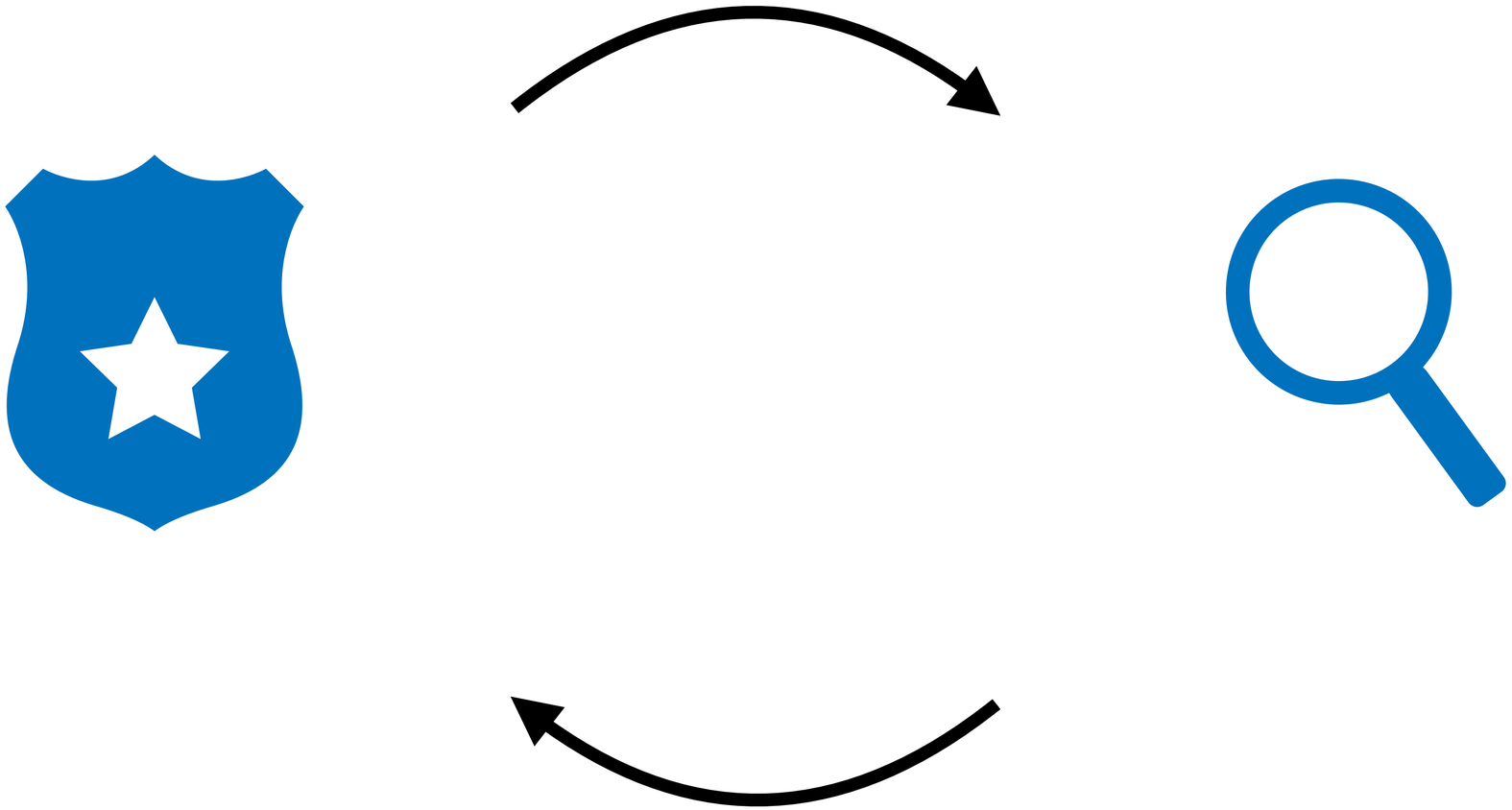
Future directions

Automating the development process



Future directions

Model governance more broadly



Future directions

Model governance more broadly