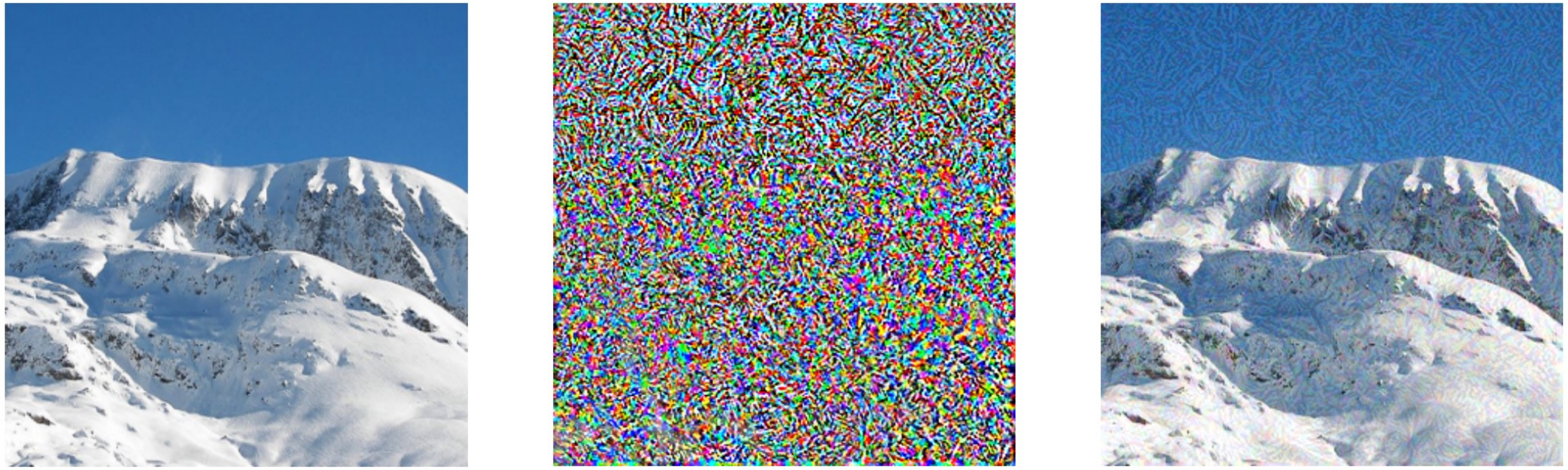
AutoDA:
Automated Decision-based Iterative Adversarial Attacks

Qi-An Fu, Yinpeng Dong,
Hang Su, Jun Zhu, Chao Zhang

Tsinghua University

Adversarial Examples

- DNNs have been integrated into security-critical applications.
  - e.g., autonomous driving, healthcare, and finance.
- DNN classifiers are vulnerable to adversarial examples.
  - Small adversarial perturbations can fool DNNs.



Alps: 94.39% Dog: 99.99%

Dong et al. 2018

Adversarial Attack & Defense

- Threat models
  - Distance metrics: l_2 or l_∞.
  - Attacker's goal: *targeted* or *untargeted.*
  - Attacker's knowledge about the target model: *white-box* or *black-box.*
- *Black-box* attacks
  - Scored-based.
  - Transfer-based.
  - Decision-based.
- Defense
  - Adversarial training.

Attack with less knowledge about the target model is usually more challenging and practical!

Automated Attacks?

- Developing adaptive attacks is necessary to evaluate defenses.
  - Designed by expert case by case.
  - Requiring lots of manual trial-and-error efforts.
- Decision-based black-box attack.
  - Jacobian-based attacks.
  - Boundary attack.
  - Evolutionary attack.
  - HSJ attack.
  - Sign-OPT attack.

based on heuristics

based on zeroth-order optimization

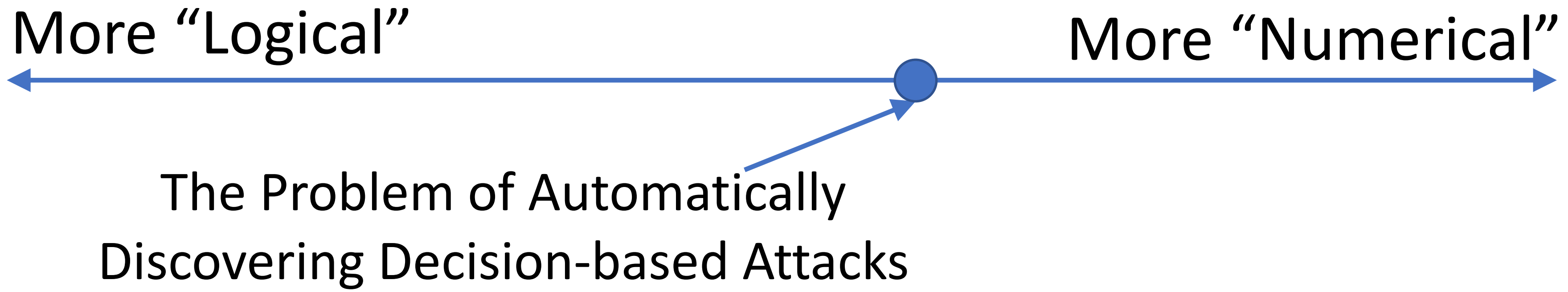
Program Synthesis & AutoML

Program Synthesis

- Objective: find programs satisfying some specifications/constraints.
- Search space: programs.
- Use solvers:
  - e.g., SAT solvers, SMT solvers.

Neural Architecture Search (NAS)

- Objective: find neural network architectures achieving higher accuracy.
- Search space: constructed from expert-designed layers.
- Use advanced search method:
  - e.g., reinforcement learning, gradient-based methods.



AutoDA

- **Auto**mated **D**ecision-based Iterative Adversarial **A**ttacks.
- For simplicity, focus on untargeted attacks.
- Intuition: Boundary attack & Evolutionary attack.
  - Their implementations share a quite similar control flow.
  - Their main difference lies in a loop-free code segment.
  - This code segment use only a dozen of mathematical operations.

Fix the control flow using an algorithm template

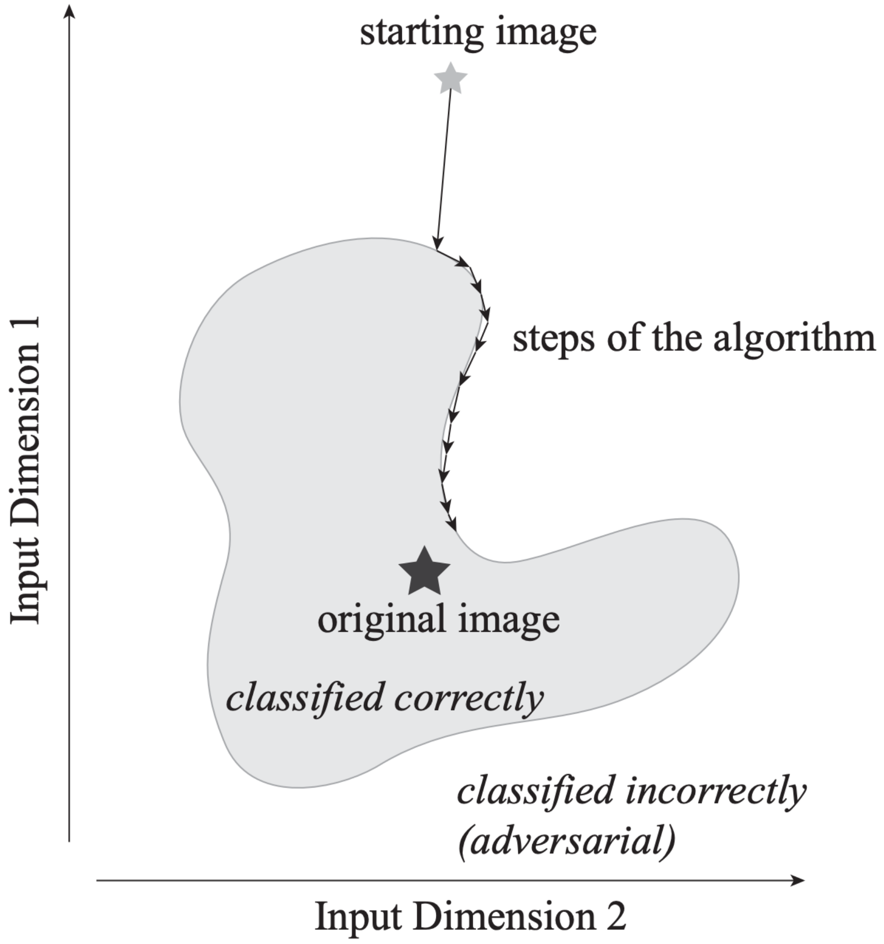
Search for the loop-free code segment

Define Search Space → Define Search Method → Define Evaluation Method

Random-walk Framework for $\boldsymbol{l_2}$ Decision-based Attacks

- Proposed in the Boundary attack.
- Used by many later decision-based attacks.

Data: original example $\mathbf{x}_0$, adversarial starting point $\mathbf{x}_1$;
Output: adversarial example $\mathbf{x}$ such that the ℓ_2 distortion $\|\mathbf{x}-\mathbf{x}_0\|_2$ is minimized;
Initialization: $\mathbf{x} \leftarrow \mathbf{x}_1$; $d_{\min} \leftarrow \|\mathbf{x}-\mathbf{x}_0\|_2$;
while query budget is not reached **do**
  $\mathbf{x}' \leftarrow \boxed{\texttt{generate}(\mathbf{x}, \mathbf{x}_0)}$;
  if $\mathbf{x}'$ is adversarial **and** $\|\mathbf{x}'-\mathbf{x}_0\|_2 < d_{\min}$ **then**
    $\mathbf{x} \leftarrow \mathbf{x}'$; $d_{\min} \leftarrow \|\mathbf{x}-\mathbf{x}_0\|_2$;
  end if
  Update the success rate of whether $\mathbf{x}'$ is adversarial;
  Adjust hyperparameters according to the success rate;
end while



Brendel et al. 2018

Search Space

- Only search for the `generate()` function.
- Define the search space as **programs** expressed in a DSL.
  - 10 basic scalar and vector mathematical operations.
  - Loop-free, SSA form programs.
  - Accept 3 arguments x, x_0, n.
- Adequate *expressiveness*:
  - Enough to express the Boundary attack's `generate()` function.
- Affordable *complexity*.

ID	Notation	Description
1	ADD.SS	scalar-scalar addition
2	SUB.SS	scalar-scalar subtraction
3	MUL.SS	scalar-scalar multiplication
4	DIV.SS	scalar-scalar division
5	ADD.VV	vector-vector element-wise addition
6	SUB.VV	vector-vector element-wise subtraction
7	MUL.VS	vector-scalar broadcast multiplication
8	DIV.VS	vector-scalar broadcast division
9	DOT.VV	vector-vector dot product
10	NORM.V	vector ℓ_2 norm

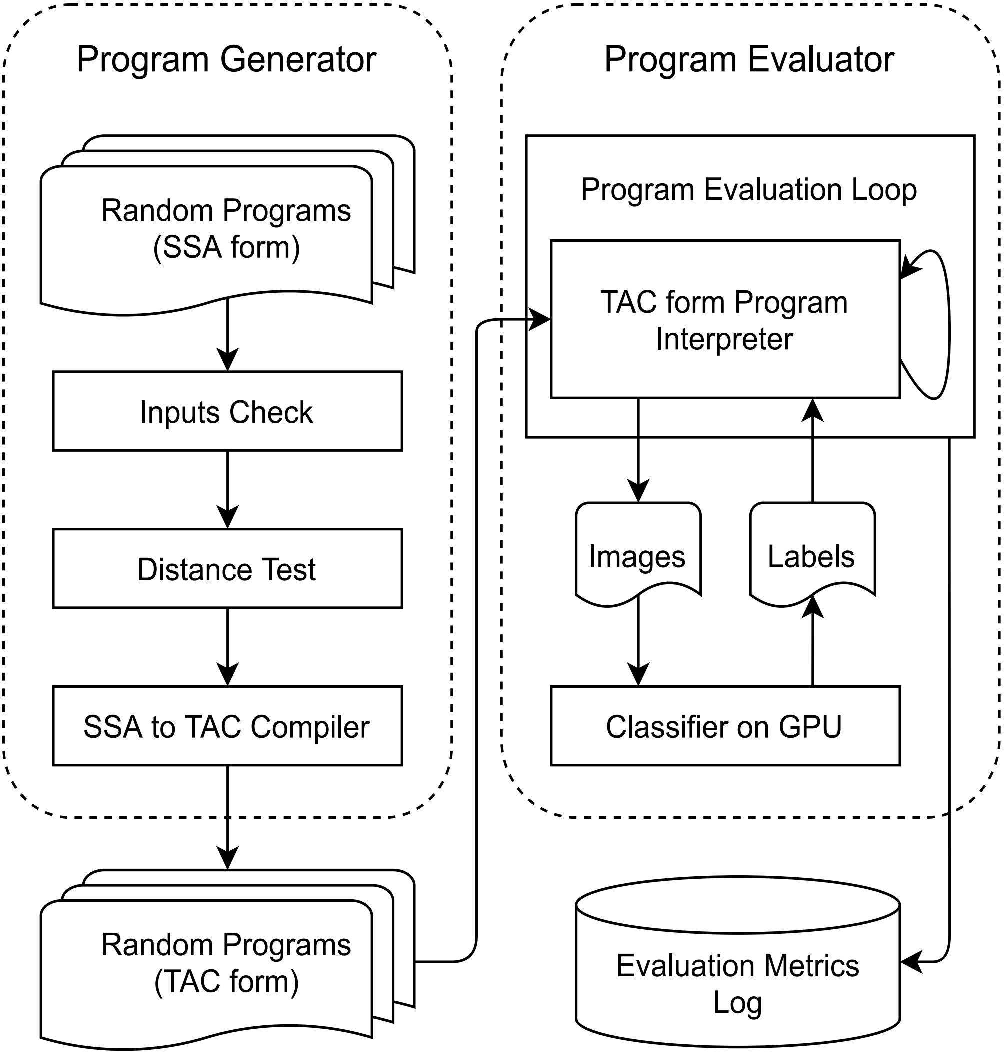
Search Method

- Random search combined with two pruning techniques and two priors.
- Pruning techniques:
  - *Inputs check*: meaningful attacks should use all 3 inputs arguments.
  - *Distance test*: `generate()` should reduce the distance between adversarial example x and original example x_0.
- Priors:
  - *Compact program*: generate less unused statements.
  - *Predefined statements*: the distance d and the angle u between x and x_0.

Program Evaluation Method

- Use a small and fast EfficientNet classifier on class 0 & 1 from CIFAR-10.
  - Can process more than 60,000 images/second on a single GTX 1080 Ti GPU.
- Evaluate programs on a handful of examples to save GPU time.
- l_2 *distortion ratio.* $\frac{||x-x_0||_2}{||x_1-x_0||_2}$
  - The extra $||x_1 - x_0||_2$ is for reducing the impact of the starting points.
- Two rounds evaluation:
  - 1st round: evaluate programs with 100 steps, only keep the best program in each batch.
  - 2nd round: evaluate programs with 10,000 steps.

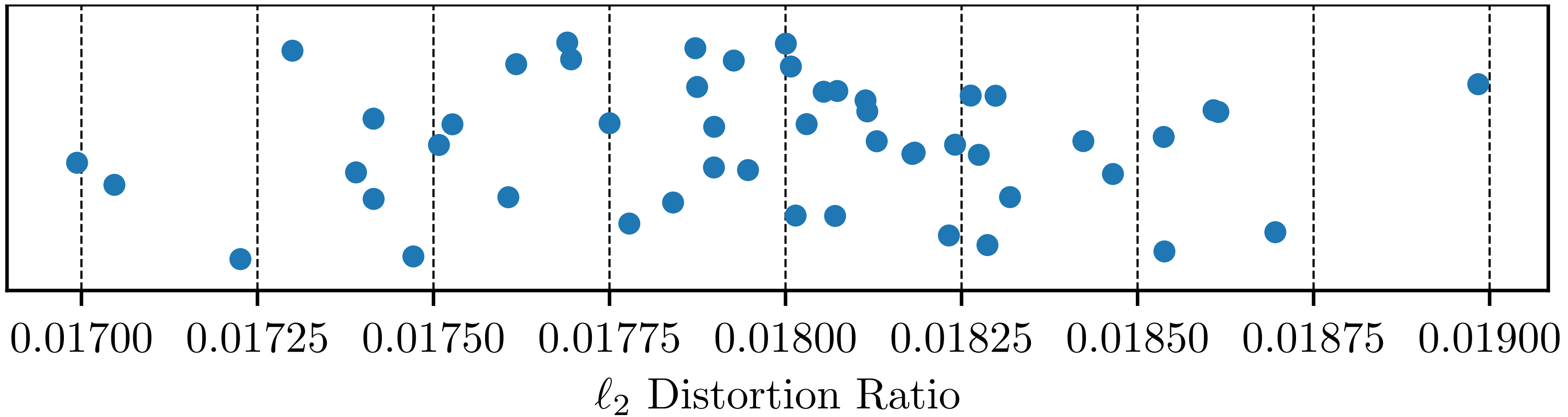
Implementation



- We implemented a prototype of AutoDA from scratch.
  - About 4,000 lines of C++.
  - About 2,000 lines of Python.
- Program generator generates programs with the two priors, and filters bad programs.
- Program evaluator evaluates programs against the classifier on GPU.
- Communications between CPU and GPU tasks are done asynchronously in large batches.

Searching for Programs Experiments

- 50 runs. Each run allows 500 million queries to the classifier.
- About 125 billion random programs are generated.
  - 45.475% of them failed in the *inputs check.*
  - 54.497% of them failed in the *distance test.*
  - Only 0.028% of them survived both.
- Distribution of the lowest l_2 distortion ratios found in each of the 50 runs: average at 0.01797 with a standard deviation of 0.00043.



AutoDA 1st & 2nd: The top-2 programs with lowest $\boldsymbol{l_2}$ distortion ratios

Adversarial example x

Original example x_0

Random noise n

Hyperparameter

10 statements

```
def AutoDA_1st(s0, v1, v2, v3):
    v4 = SUB.VV(v1, v2)
    s5 = NORM.V(v4)
    v6 = DIV.VS(v4, s5)
    v8 = MUL.VS(v3, s0)
    v11 = MUL.VS(v8, s5)
    v17 = ADD.VV(v8, v6)
    s18 = DOT.VV(v17, v8)
    v21 = MUL.VS(v4, s18)
    v22 = ADD.VV(v21, v2)
    v23 = SUB.VV(v22, v11)
    return v23
```

Next point to walk to x'

```
def AutoDA_2nd(s0, v1, v2, v3):
    v4 = SUB.VV(v1, v2)
    s5 = NORM.V(v4)
    v6 = DIV.VS(v4, s5)
    v7 = MUL.VS(v3, s0)
    v8 = ADD.VV(v7, v6)
    s9 = NORM.V(v2)
    v10 = MUL.VS(v8, s5)
    s11 = DOT.VV(v10, v7)
    v12 = DIV.VS(v6, s9)
    v17 = ADD.VV(v6, v12)
    v20 = MUL.VS(v17, s11)
    v21 = ADD.VV(v1, v20)
    v23 = SUB.VV(v21, v10)
    return v23
```

13 statements

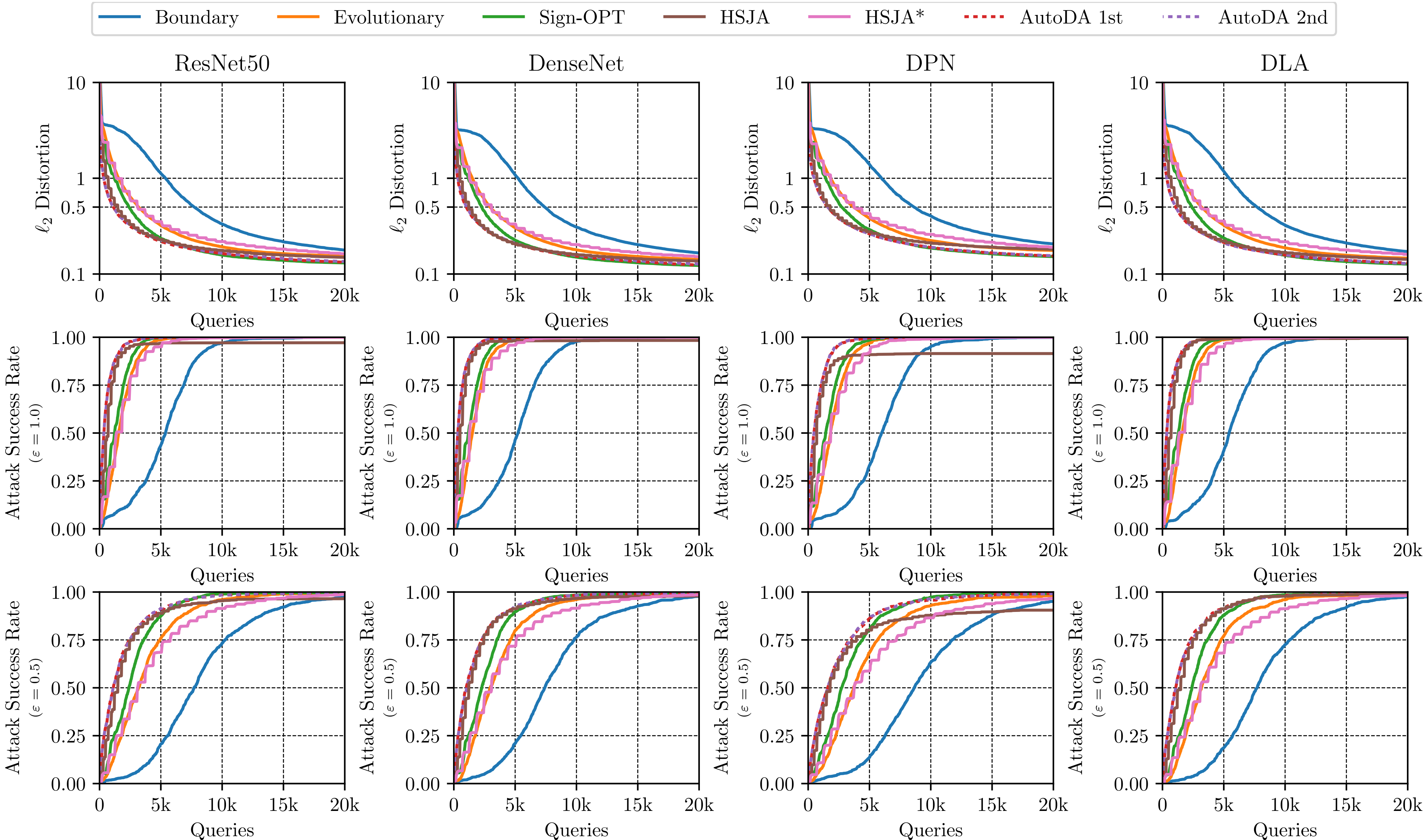
Benchmark Experiments

- Expert-designed baselines
  - Boundary attack.
  - Evolutionary attack.

  Random-walk based. Inspired our method.
  - HopSkipJump attack (HSJA). (S&P 2020)
    - HSJA (default) & HSJA* (grid search).
  - Sign-OPT attack. (ICLR 2020)

  SOTA
- Benchmark metrics
  - *Median l_2 distortion* vs. *queries* curve.
  - *Attack success rate* vs. *queries* curve.

Benchmark Experiments on CIFAR-10 models



Benchmark Experiments on CIFAR-10 models

Model	ResNet50			DenseNet		
Queries	2,000	4,000	20,000	2,000	4,000	20,000
Boundary	10.7%	28.4%	100.0%	10.6%	28.5%	100.0%
Evolutionary	64.9%	96.3%	100.0%	66.9%	95.8%	100.0%
Sign-OPT	76.1%	98.8%	100.0%	77.8%	98.9%	100.0%
HSJA	91.9%	96.6%	97.1%	94.2%	97.9%	98.3%
HSJA*	67.4%	92.6%	100.0%	71.7%	92.9%	100.0%
AutoDA 1st	**95.9%**	**99.7%**	100.0%	96.4%	99.5%	100.0%
AutoDA 2nd	95.6%	99.5%	100.0%	**96.5%**	**99.7%**	100.0%

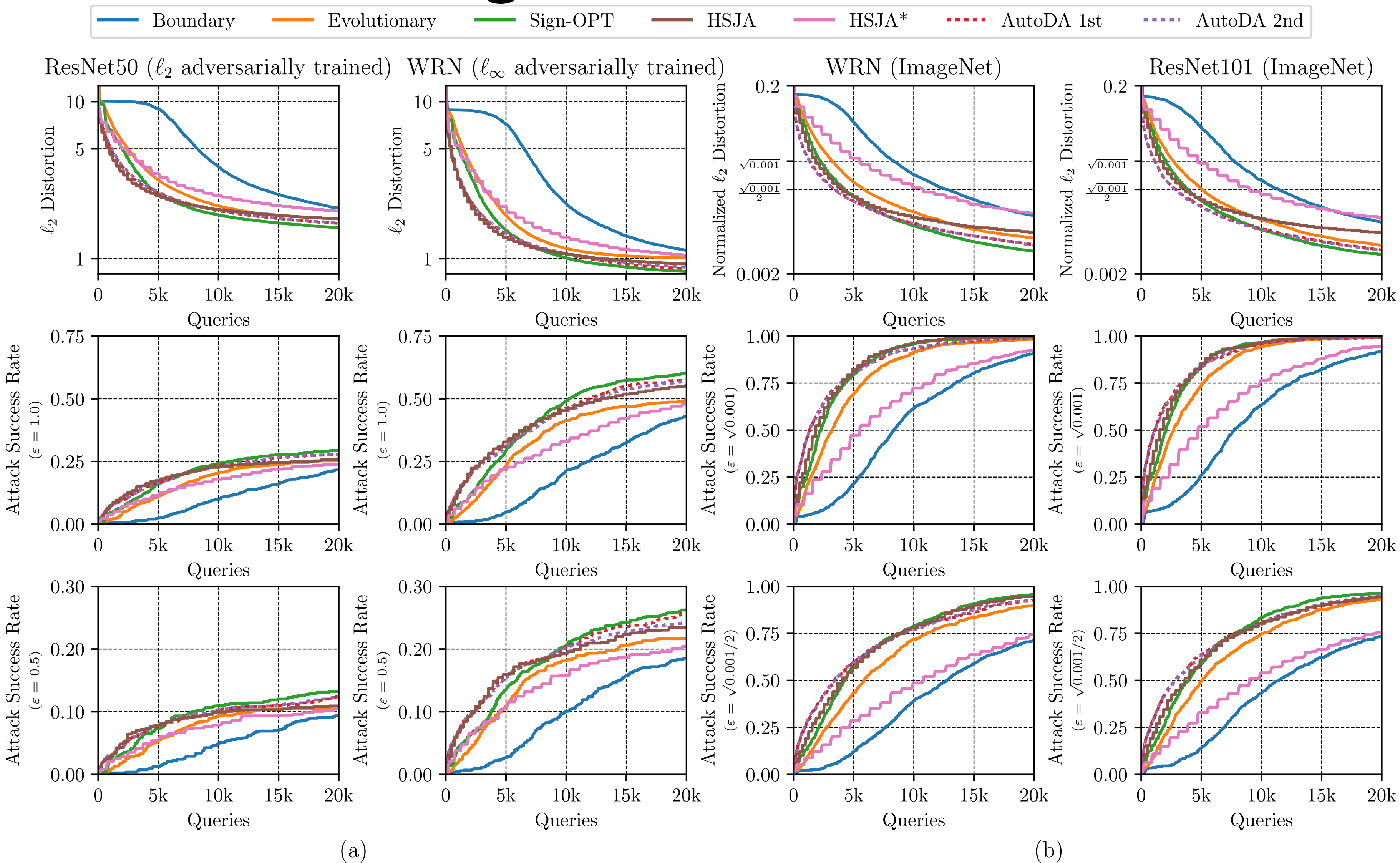
Attack success rate (ε = 1.0) vs. *queries*

Model	ResNet50			DenseNet		
Queries	2,000	4,000	20,000	2,000	4,000	20,000
Boundary	3.000	1.636	0.178	2.847	1.579	0.166
Evolutionary	0.793	0.399	0.154	0.754	0.378	0.142
Sign-OPT	0.611	0.288	**0.131**	0.586	0.273	**0.123**
HSJA	0.399	0.252	0.149	0.361	0.228	0.137
HSJA*	0.732	0.402	0.162	0.680	0.376	0.152
AutoDA 1st	**0.356**	**0.245**	0.133	**0.338**	**0.231**	0.124
AutoDA 2nd	0.364	0.254	0.135	0.344	0.236	0.127

Median l_2 distortion vs. *queries*

- Though our search space is based on the Boundary attack, AutoDA 1st & 2nd are much stronger than it.
- AutoDA 1st & 2nd converge faster before ~7k queries, while converge to slightly worse adversarial examples than Sign-OPT.

Benchmark Experiments on Adv. Trained & ImageNet models



Conclusion

- A novel solution to automatically discover decision-based iterative adversarial attacks.
- A way to construct a search space of decision-based iterative attacks.
- An effective random search algorithm to efficiently explore the search space.
- A prototype of AutoDA
  - The discovered attacks are simple yet powerful;
  - They show comparable performance than SOTA expert-designed attacks;
  - Suggesting these expert-designed attacks are near optimal in our search space.

Thanks for listening!
Q&A

Contact: Qi-An Fu, fugoes.qa@gmail.com