SYSTEMS SECURITY RESEARCH LAB

KING'S College LONDON

Talk based on the IEEE S&P '20 paper with Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro

Intriguing Properties of Adversarial ML Attacks in the Problem Space

Fabio Pierazzi
Assistant Professor at King's College London, UK
<fabio.pierazzi@kcl.ac.uk>
https://fabio.pierazzi.com

SoSySec Seminar
IRISA, Rennes
June 19, 2020

A Dystopian Future…

Pandas are forbidden!

Guilty of being too cute!

Luckily, pandas are fluent in math…

Intriguing properties of neural networks

Christian Szegedy
Google Inc.

Wojciech Zaremba
New York University

Ilya Sutskever
Google Inc.

Joan Bruna
New York University

Dumitru Erhan
Google Inc.

Ian Goodfellow
University of Montreal

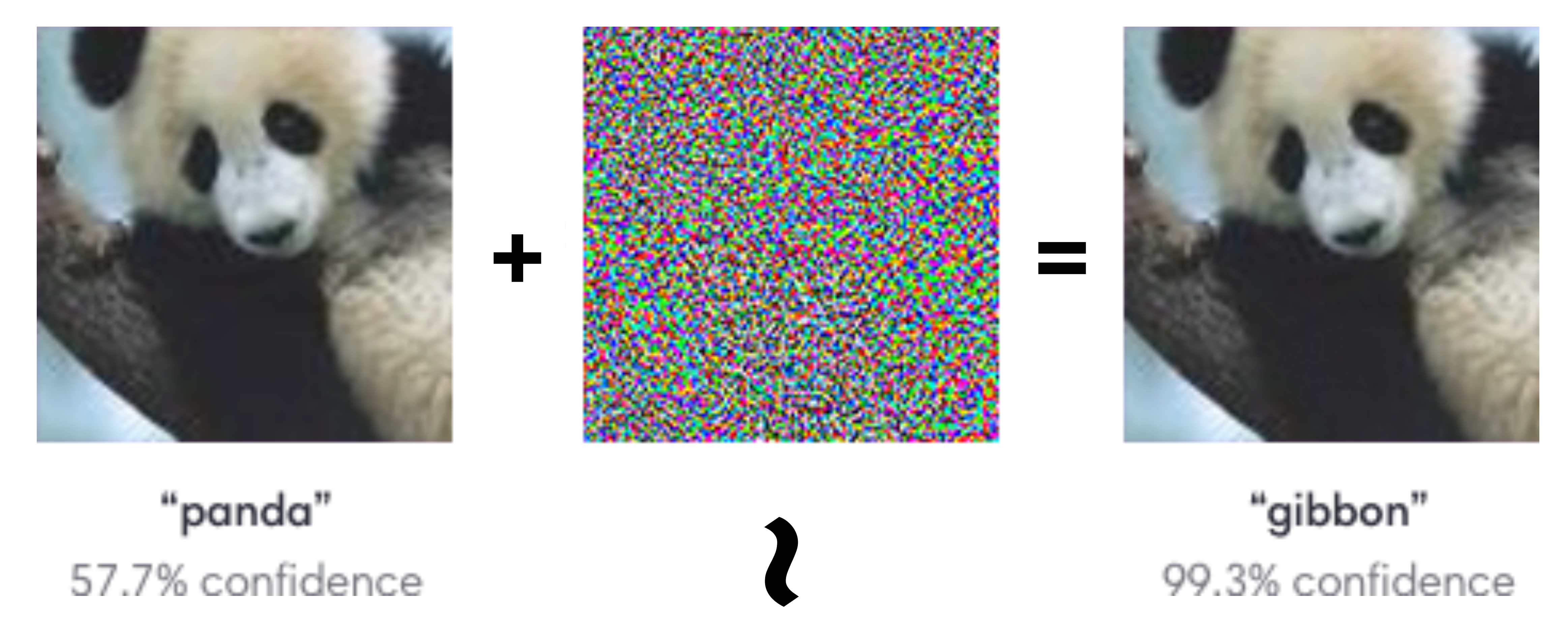
Rob Fergus
New York University
Facebook Inc.

[cs.CV] 19 Feb 2014

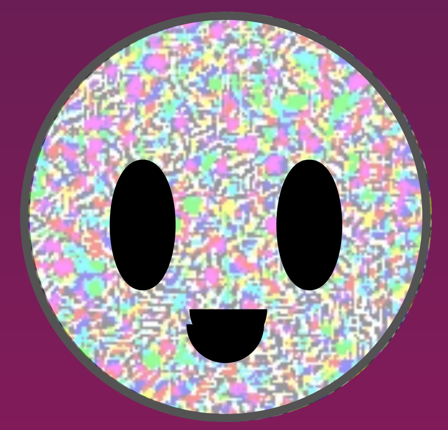
Abstract

Deep neural networks are highly expressive models that have recently achieved state of the art performance on speech and visual recognition tasks. While their expressiveness is the reason they succeed, it also causes them to learn uninterpretable solutions that could have counter-intuitive properties. In this paper we report two such properties.
First, we find that there is no distinction between individual high level units and random linear combinations of high level units, according to various methods of unit analysis. It suggests that it is the space, rather than the individual units, that contains the semantic information in the high layers of neural networks.
Second, we find that deep neural networks learn input-output mappings that are [illegible] to a significant extent. We can cause the network to misclassify [illegible] hardly perceptible perturbation, which is found [illegible] In addition, the specific nature of [illegible] perturbation can

Luckily, pandas are fluent in math…



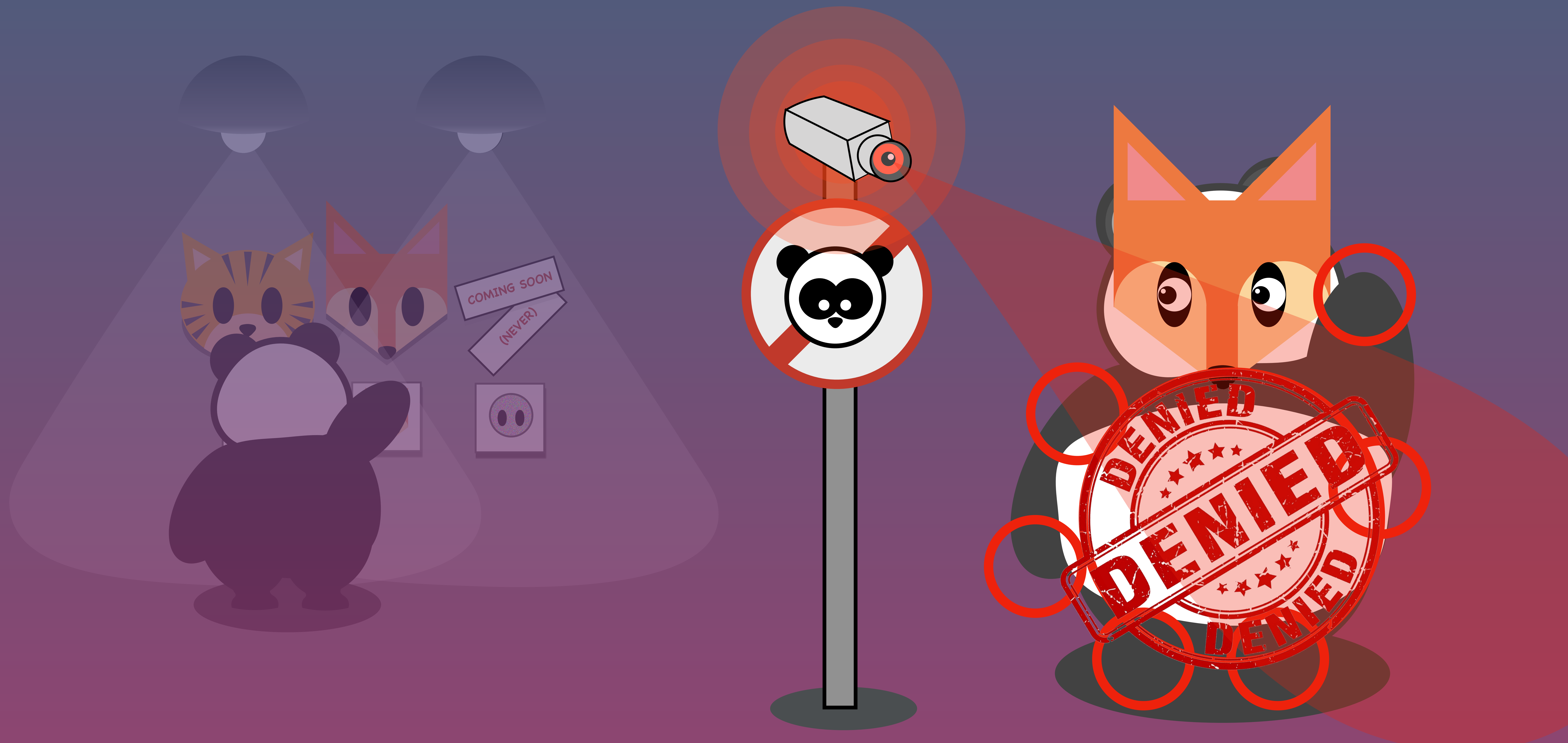
Feature-space
noise mask



What happens in the **problem-space**, i.e., the real world?

What happens in the **problem-space**, i.e., the real world?

What happens in the **problem-space**, i.e., the real world?



Let's Analyze What Happened

SYSTEMS SECURITY RESEARCH LAB
KING'S College LONDON

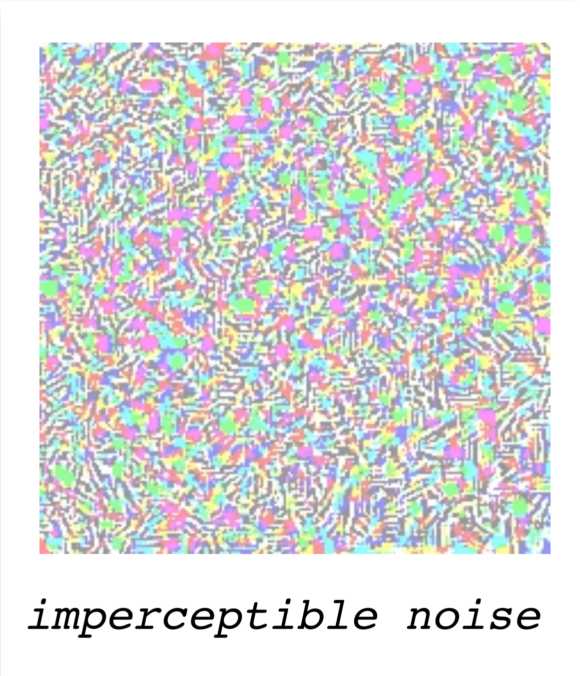
Feature-Space Attacks

Original Image

x

+

Perturbation



δ

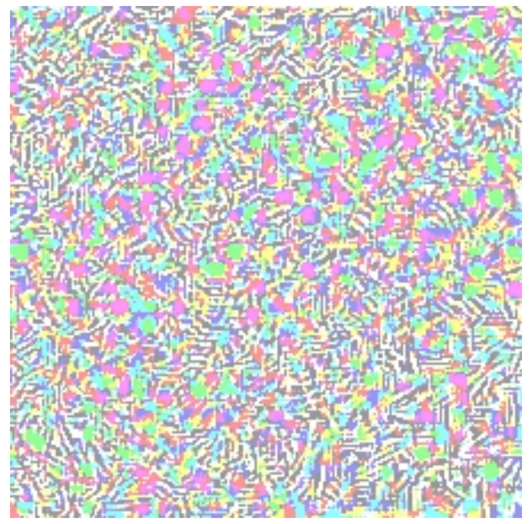
=

Adv. Image

$x + \delta$

Let’s Analyze What Happened

Feature-Space Attacks

Original Image		Perturbation		Adv. Image
	+		=	
“panda” 57.7%		*imperceptible noise*		“gibbon” 99.3%
x		δ		$x + \delta$

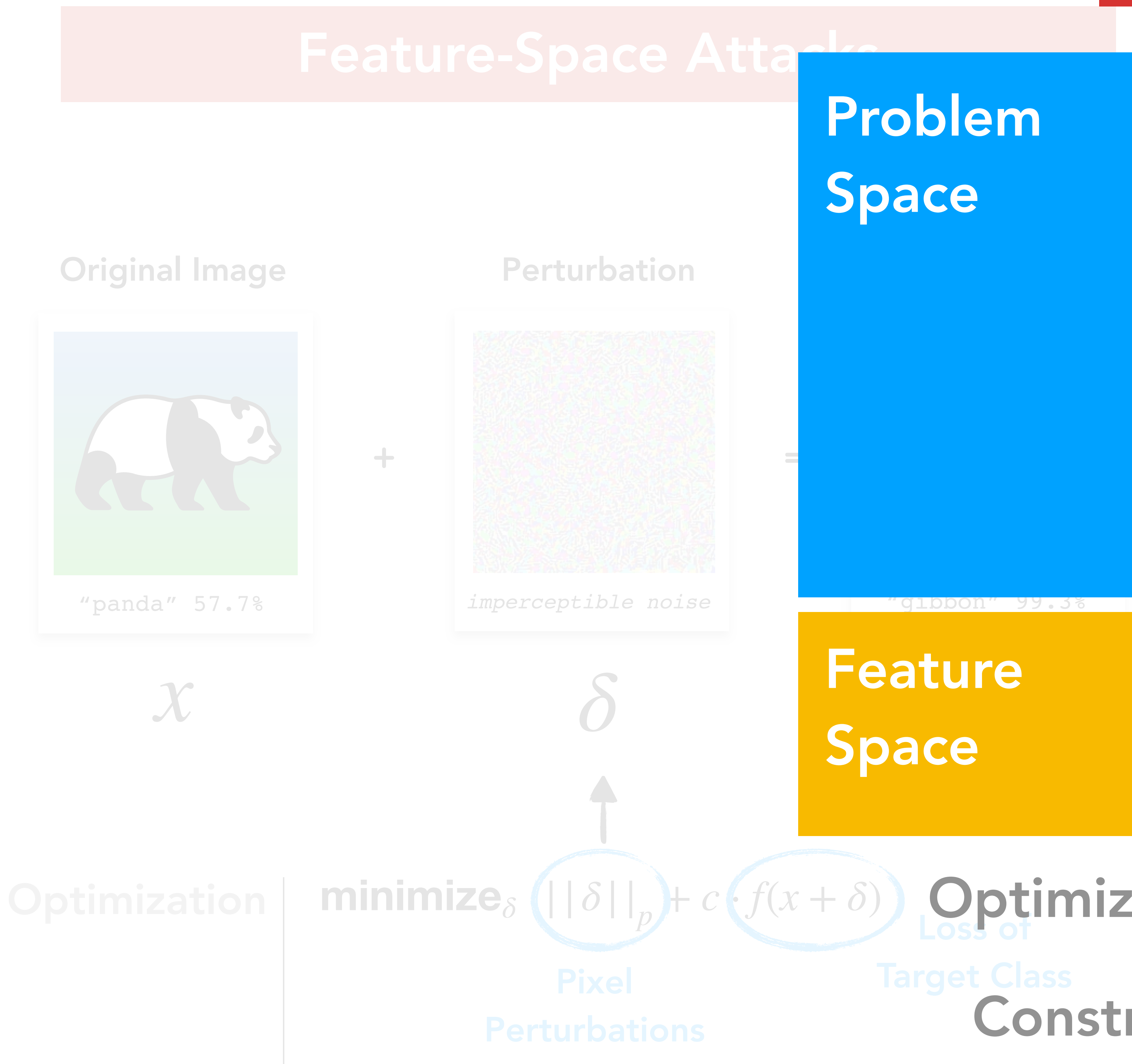
Optimization | **minimize**$_\delta$ $||\delta||_p + c \cdot f(x+\delta)$

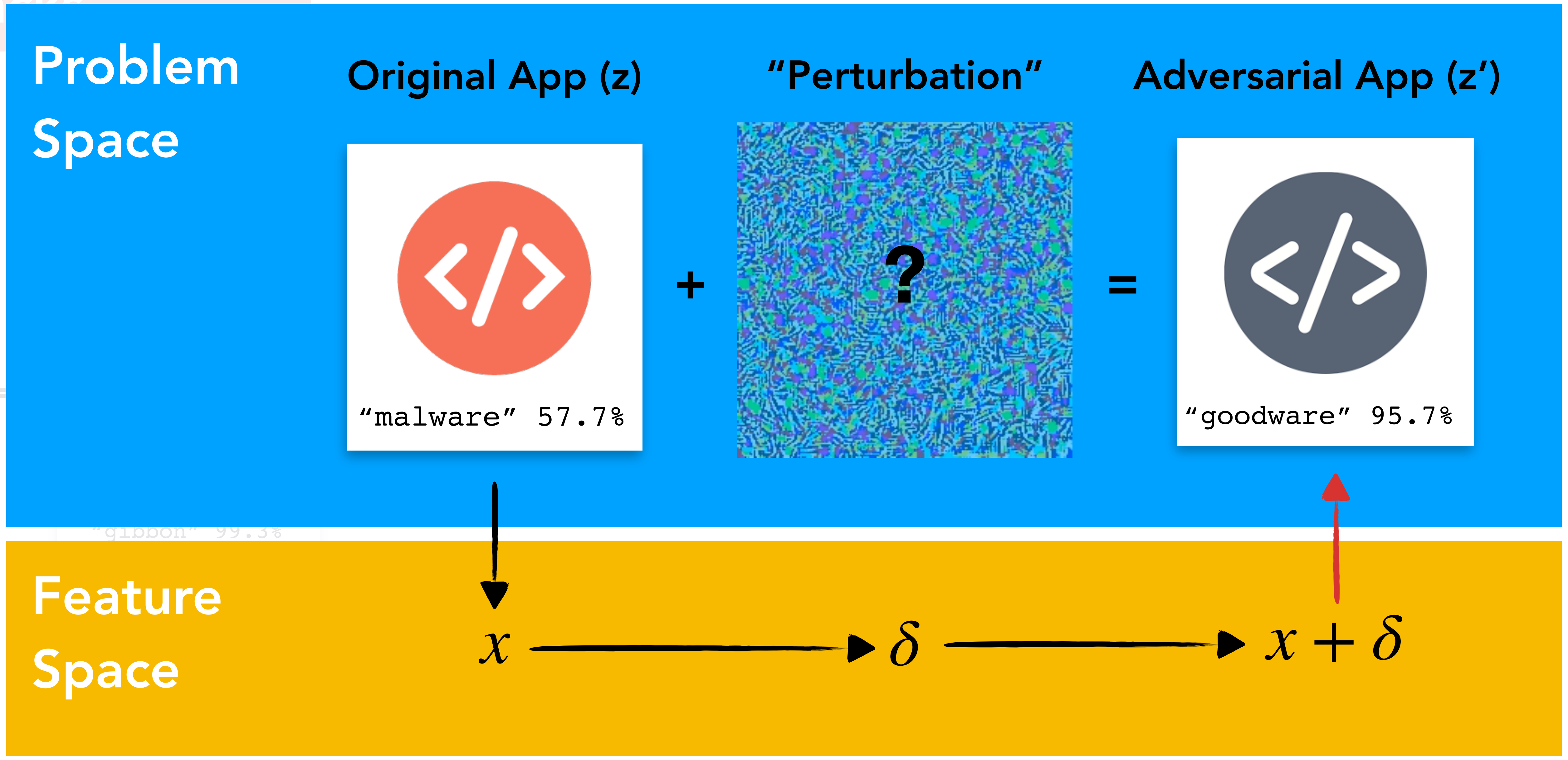
Pixel Perturbations

Loss of Target Class

Let’s Analyze What Happened

Problem-Space Attacks





? Inverse Feature-Mapping Problem

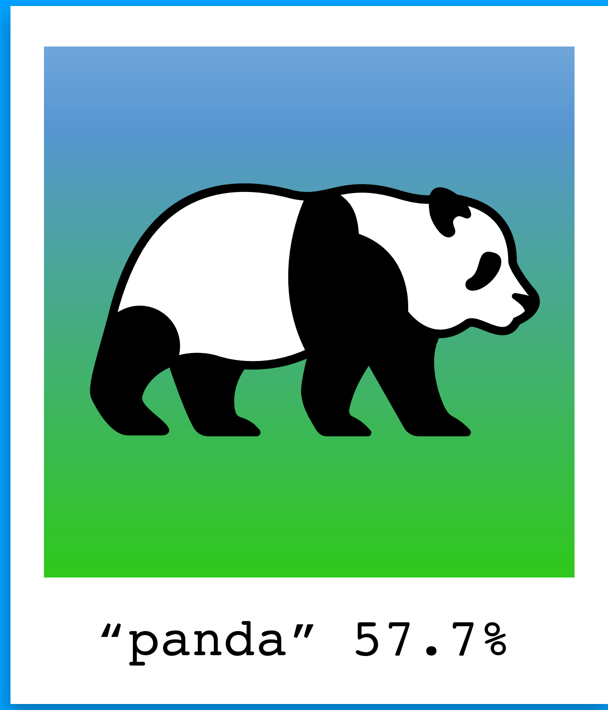
Optimization | **minimize**$_\delta$ $\cancel{||\delta||_p} + c \cdot f(x+\delta)$

Constraints

- Is it realistic/plausible?
- Does it crash?
- Can it be detected by signatures?
- Does it preserve malicious functionality?
- … are there **"general" constraints**?

Inverse Feature-Mapping Problem

Problem Space
Images



Feature Space
Vectors

φ → x

1	0	0.5	1	…	1

FS attack

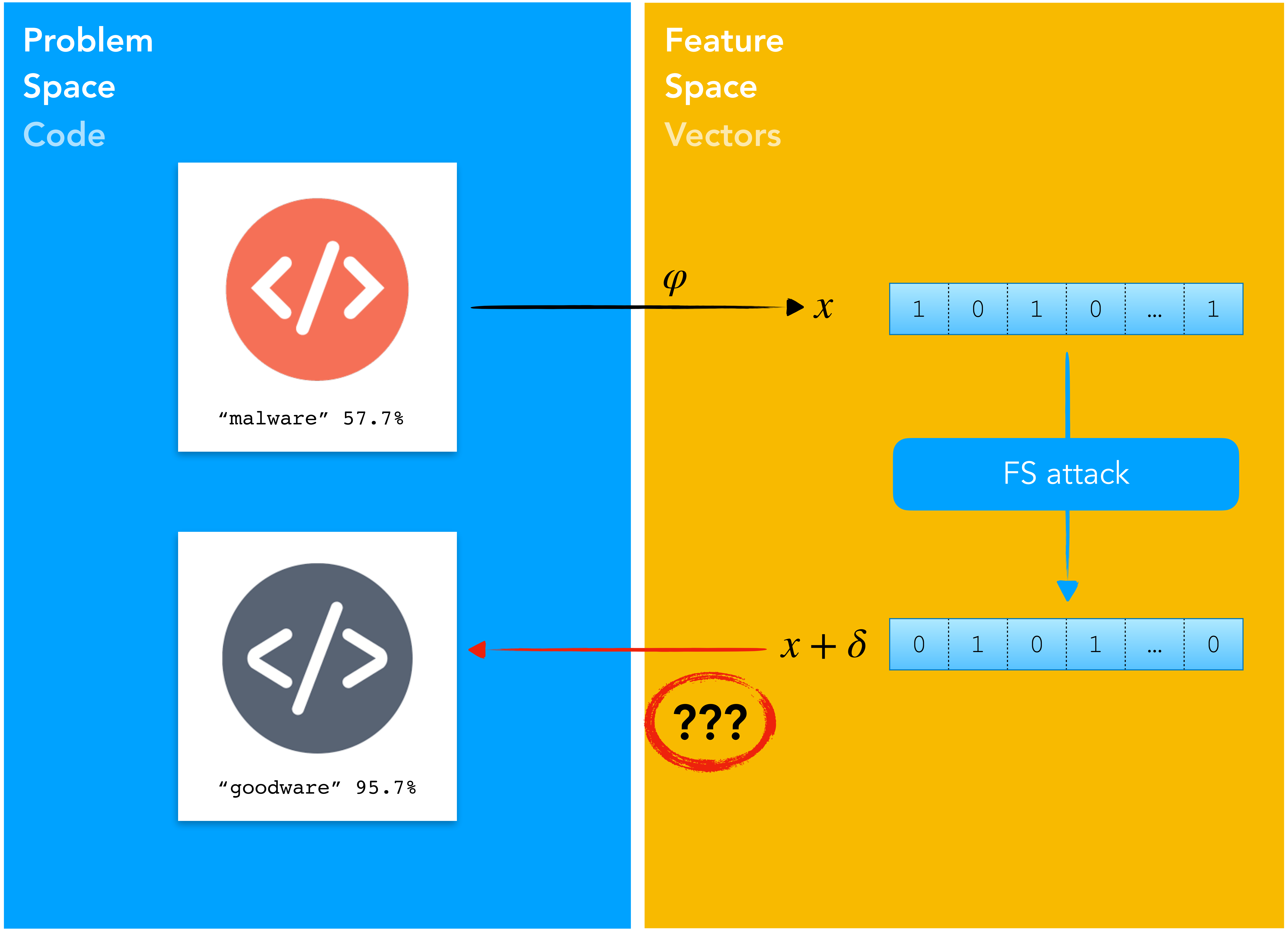
$x + \delta$

0.2	0.1	0	0	…	0.3

$\approx \varphi^{-1}$

The feature mapping φ is <u>differentiable</u>
— you can backpropagate to input

Inverse Feature-Mapping Problem



In the software domain,
the feature mapping φ is
neither <u>invertible</u> nor <u>differentiable</u>
— how to get back to the problem space?

Many Problem-Space Attack Papers

Android Malware
[TDSC'17, ESORICS'17, ACSAC'19]

Windows Malware
[RAID'18, EUSIPCO'18]

PDF Malware
[ECML-PKDD'13, NDSS'16]

Network Traffic
[NCA'18, NCA'19]

What is the State of the Art?
How to compare them?

Outline

Formalization

- Problem-space attacks
- Relationships
- Actionable points

Android Problem-Space Attack

- End-to-end adversarial malware generation at scale
- Feasible to evade feature-space defenses

Outline

Formalization

- Problem-space attacks
- Relationships
- Actionable points

Android Problem-Space Attack

- End-to-end adversarial malware generation at scale
- Feasible to evade feature-space defenses

Evasion Attacks

Running example: Code

Formalization

Background: Feature-Space Attacks

SYSTEMS SECURITY RESEARCH LAB

KING'S College LONDON

Test-time evasion

Optimal FS Perturbation

$$\delta^* = \arg\min_{\delta \in \mathbb{R}^n} f_t(x+\delta)$$

Loss of target class

$$\text{subject to: } \delta \models \Omega .$$

Feature-space Constraints

Background: Feature-Space Attacks

Test-time evasion

Optimal FS Perturbation

$$\delta^* = \arg\min_{\delta \in \mathbb{R}^n} f_t(x + \delta)$$

Loss of target class

$$\text{subject to:} \quad \delta \models \Omega \, .$$

Feature-space Constraints

Threat Models (Attacker Knowledge):

- **White box**: Feature Space, Algorithm, Training Data [1]
- **Gray box**
- **Black box**: None - perhaps type of Feature Space

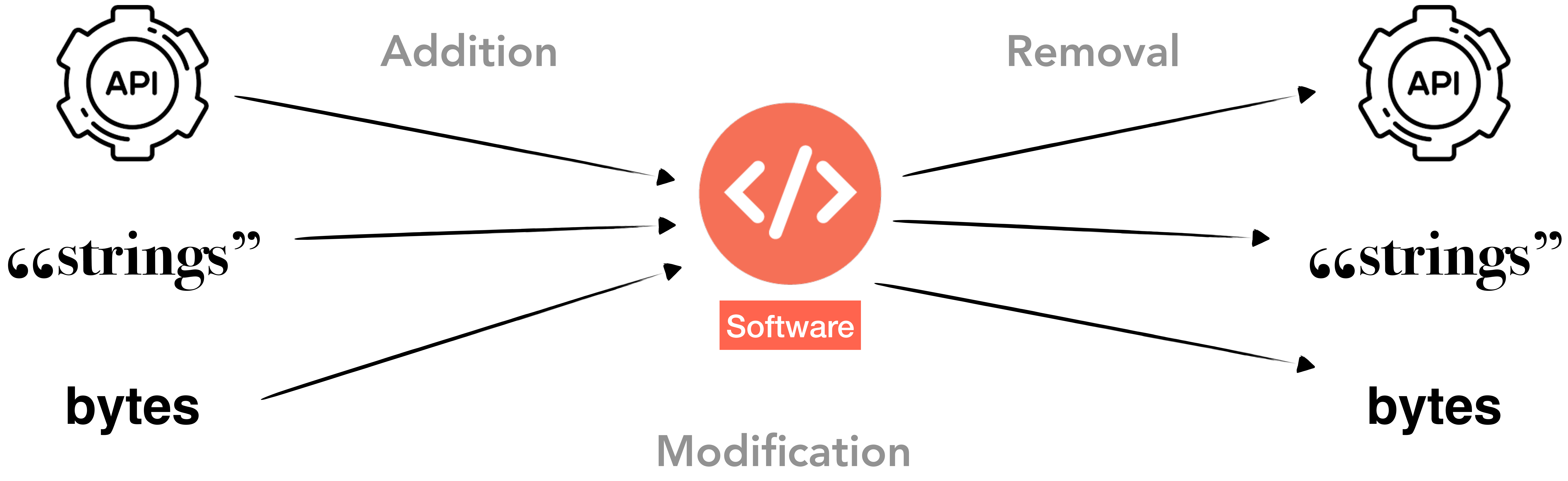
[1] Carlini et al, *"On Evaluating Adversarial Robustness", 2019*

Problem-Space Constraints

Problem-Space Constraints

Available Transformations

How can you alter problem-space objects?



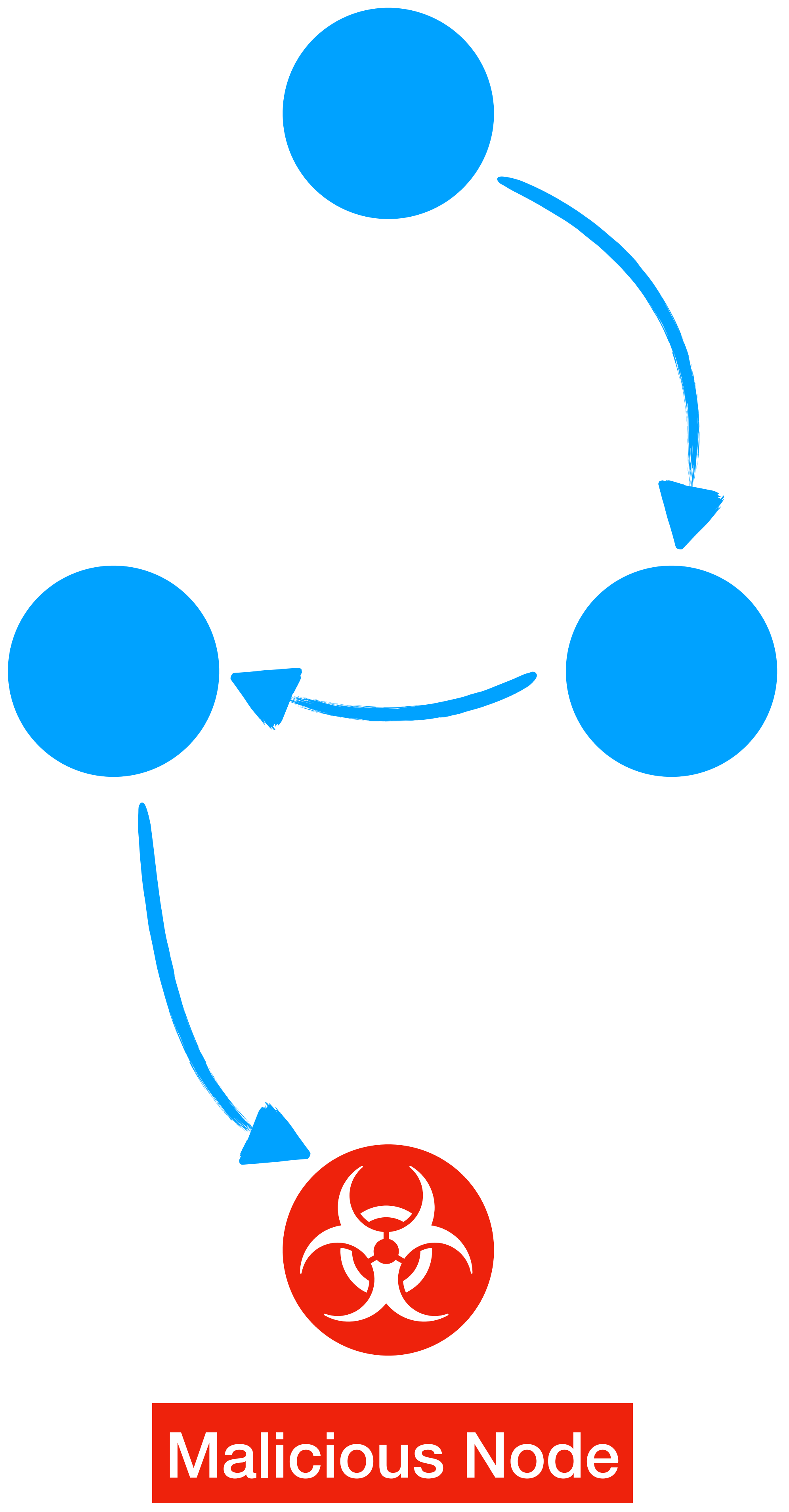
Problem-Space Constraints

Preserved Semantics

Available Transformations

Which semantics do you preserve? How?

Which automatic tests can verify it?



Problem-Space Constraints

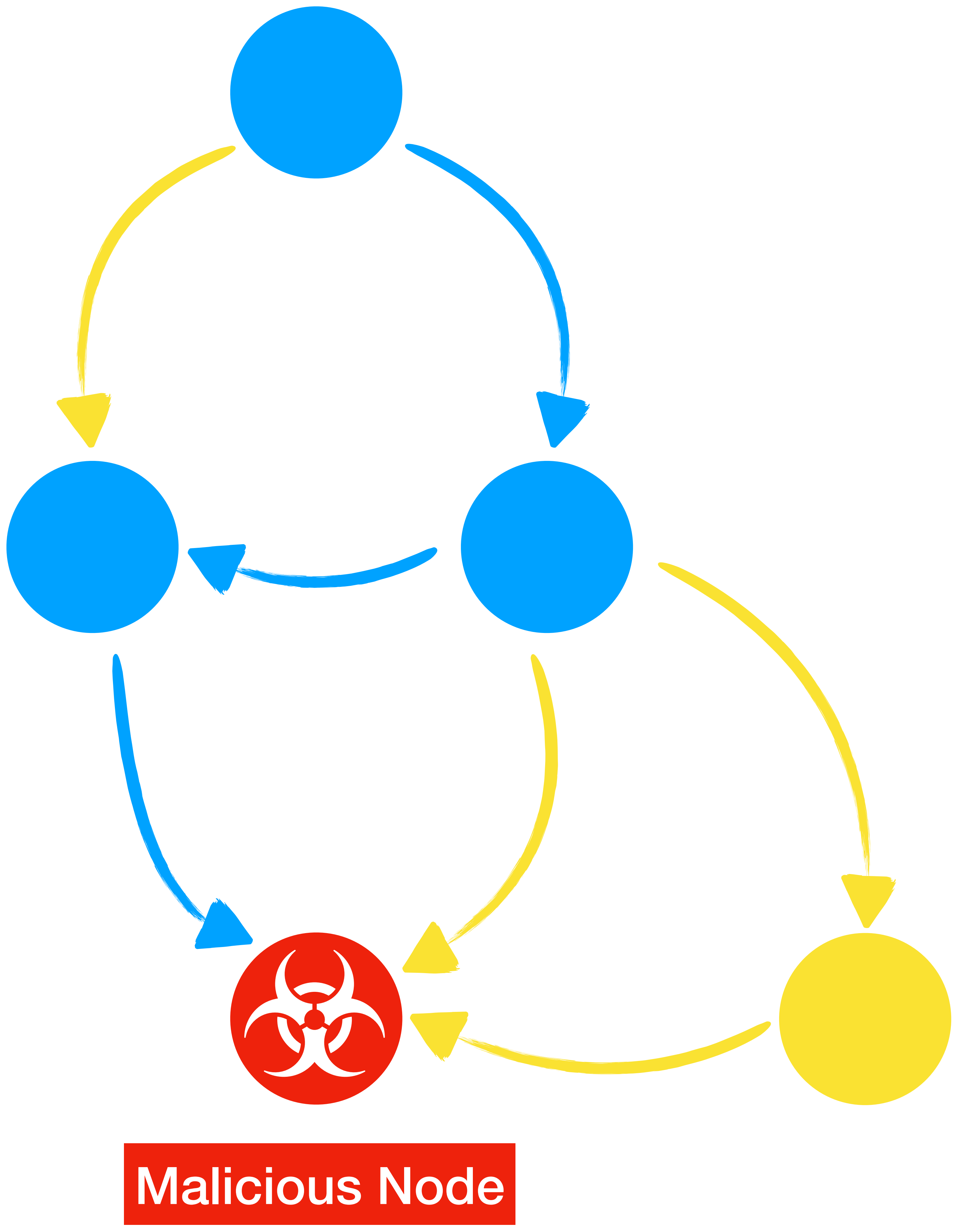
Preserved Semantics

Available Transformations

Which semantics do you preserve? How?
Which automatic tests can verify it?

Test Suite
- Does it crash?
- Does it still communicate with CnC?
- Does it still encrypt the /home/ folder?

By Construction
- Add no-op operations
- Ensure it is not executed at runtime

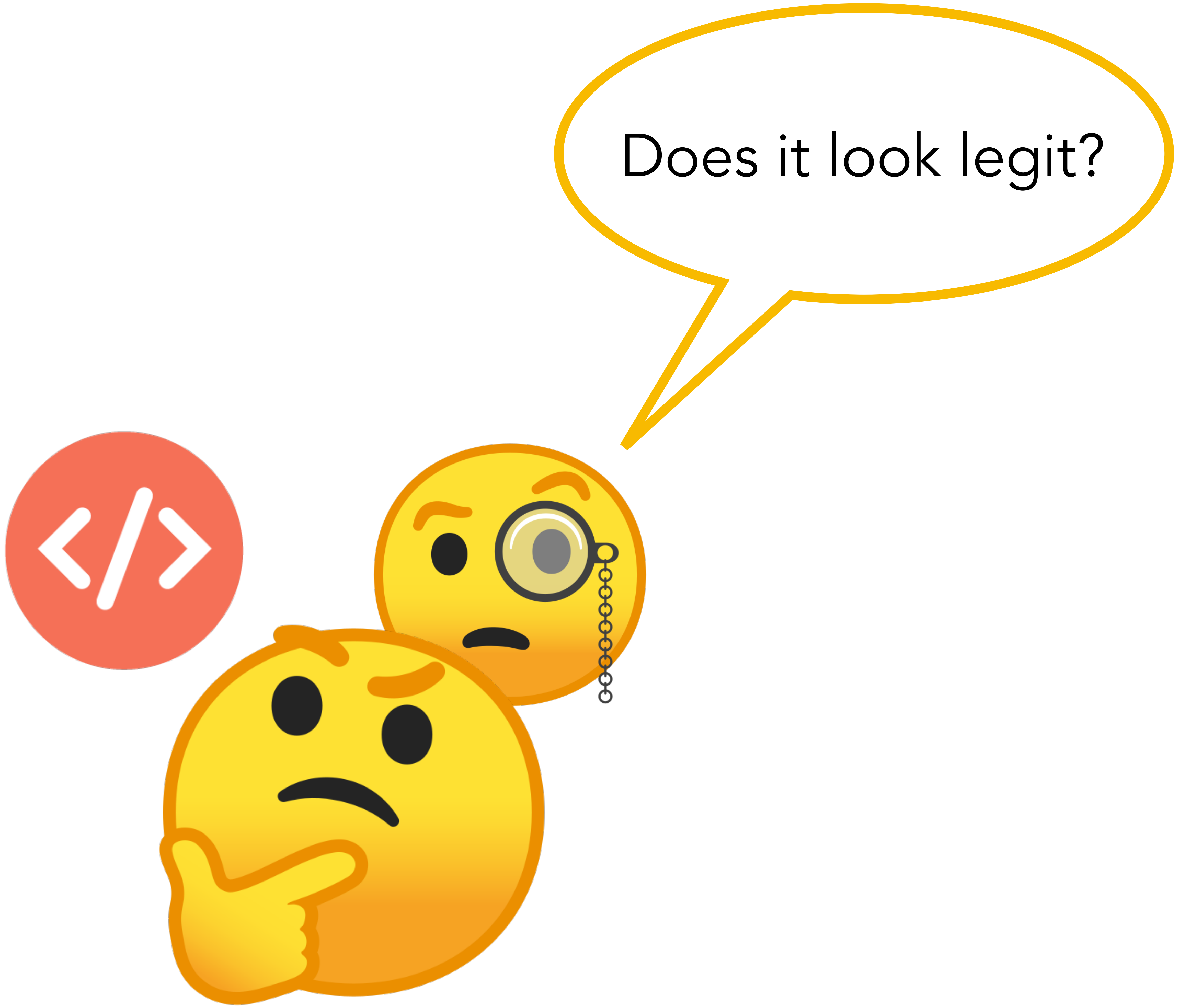


Problem-Space Constraints

Plausibility

Preserved Semantics

Available Transformations



Test Suite

- User studies
- Automated heuristics

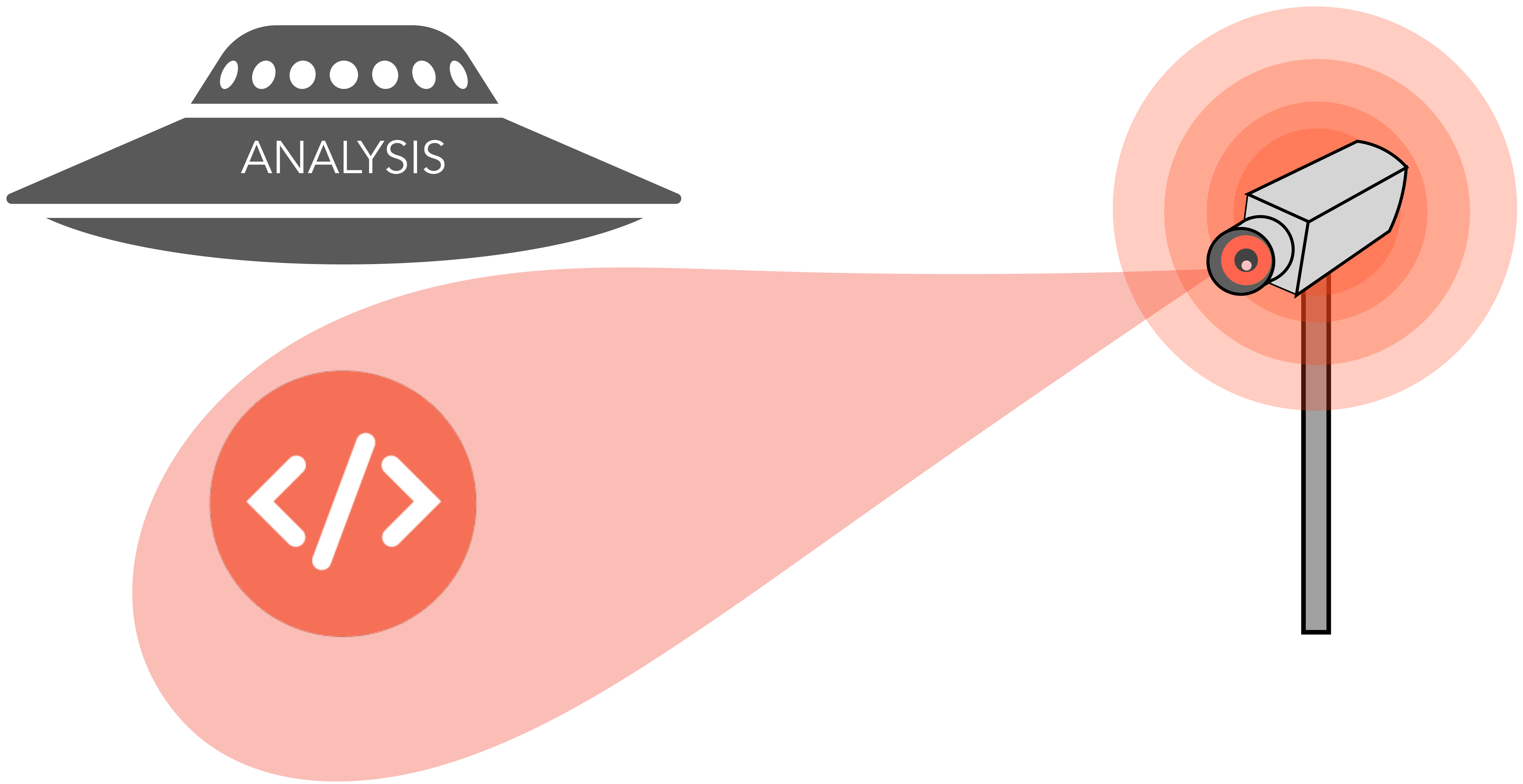
By Construction

- Taking precautions during mutation

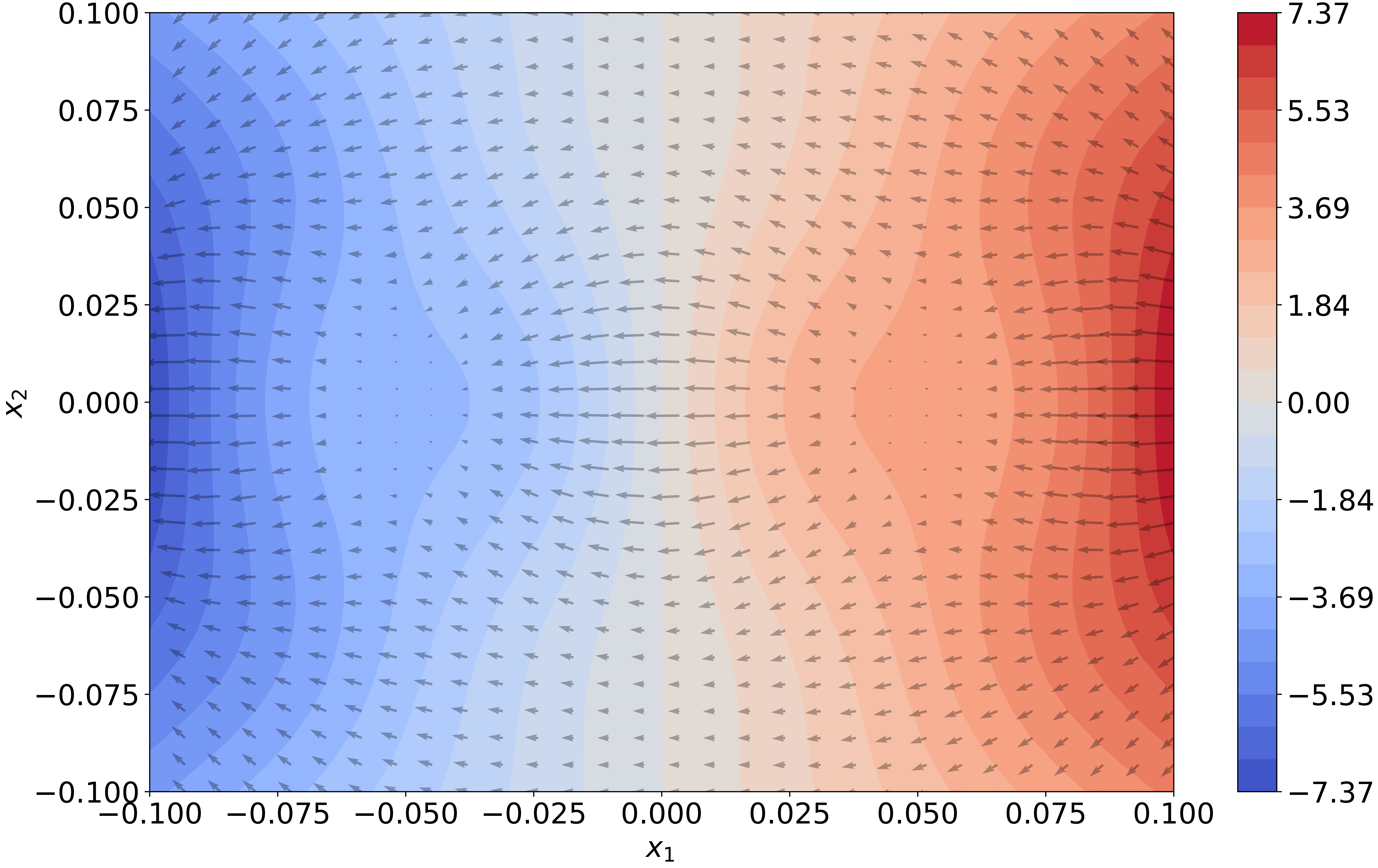
Problem-Space Constraints

- **Robustness to Preprocessing**
- Plausibility
- Preserved Semantics
- Available Transformations

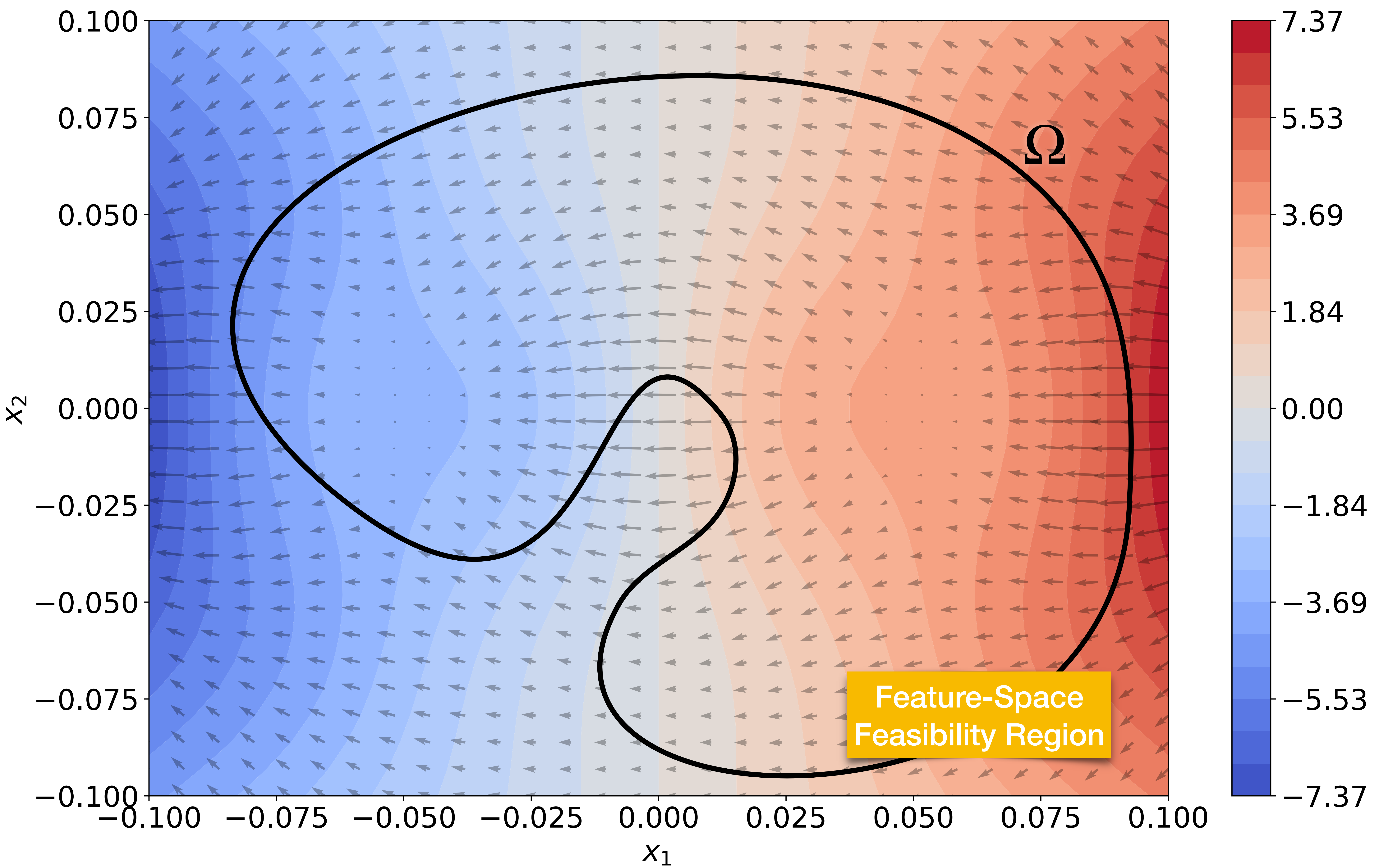
Which preprocessing are you considering?



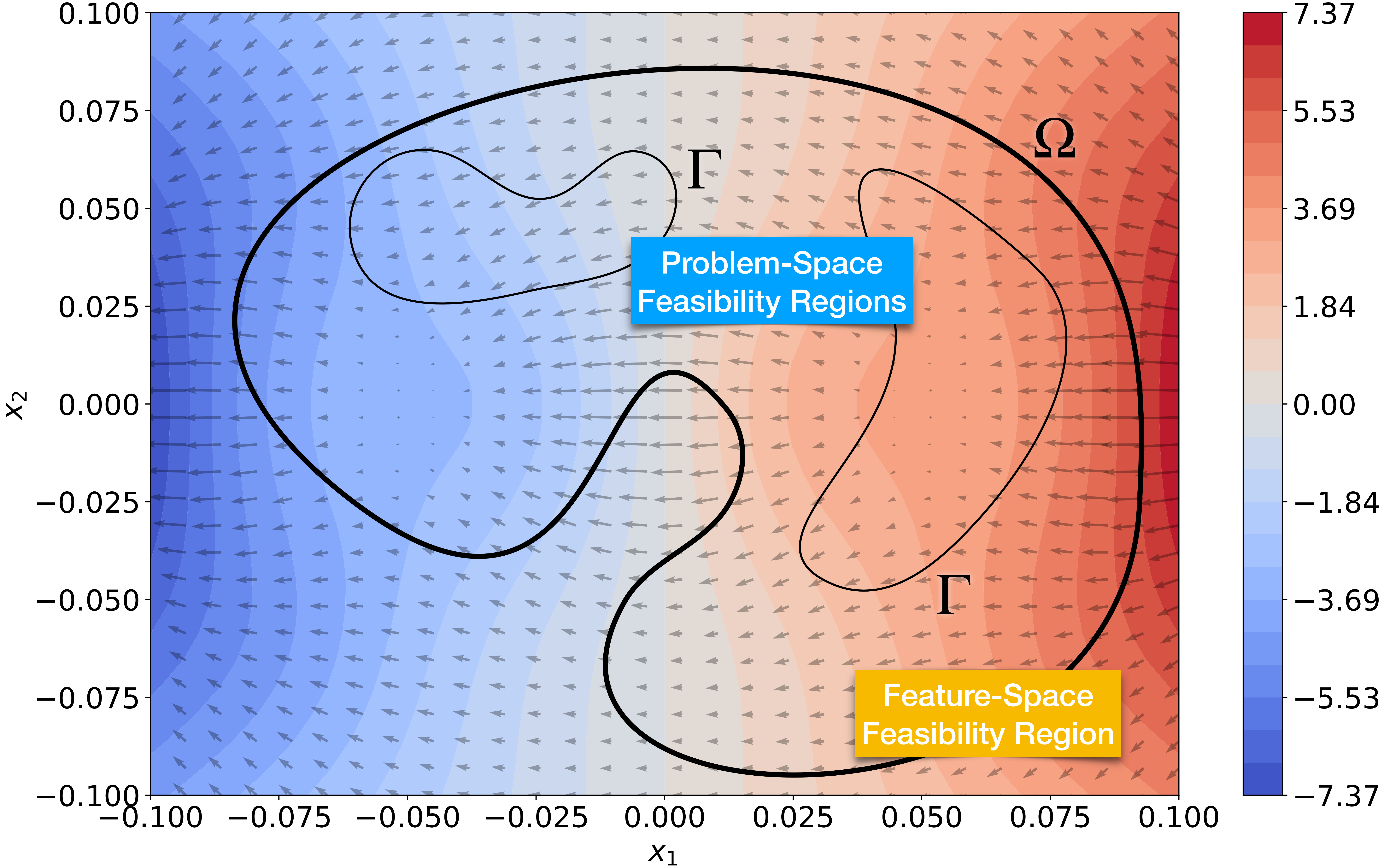
Side-effect Features



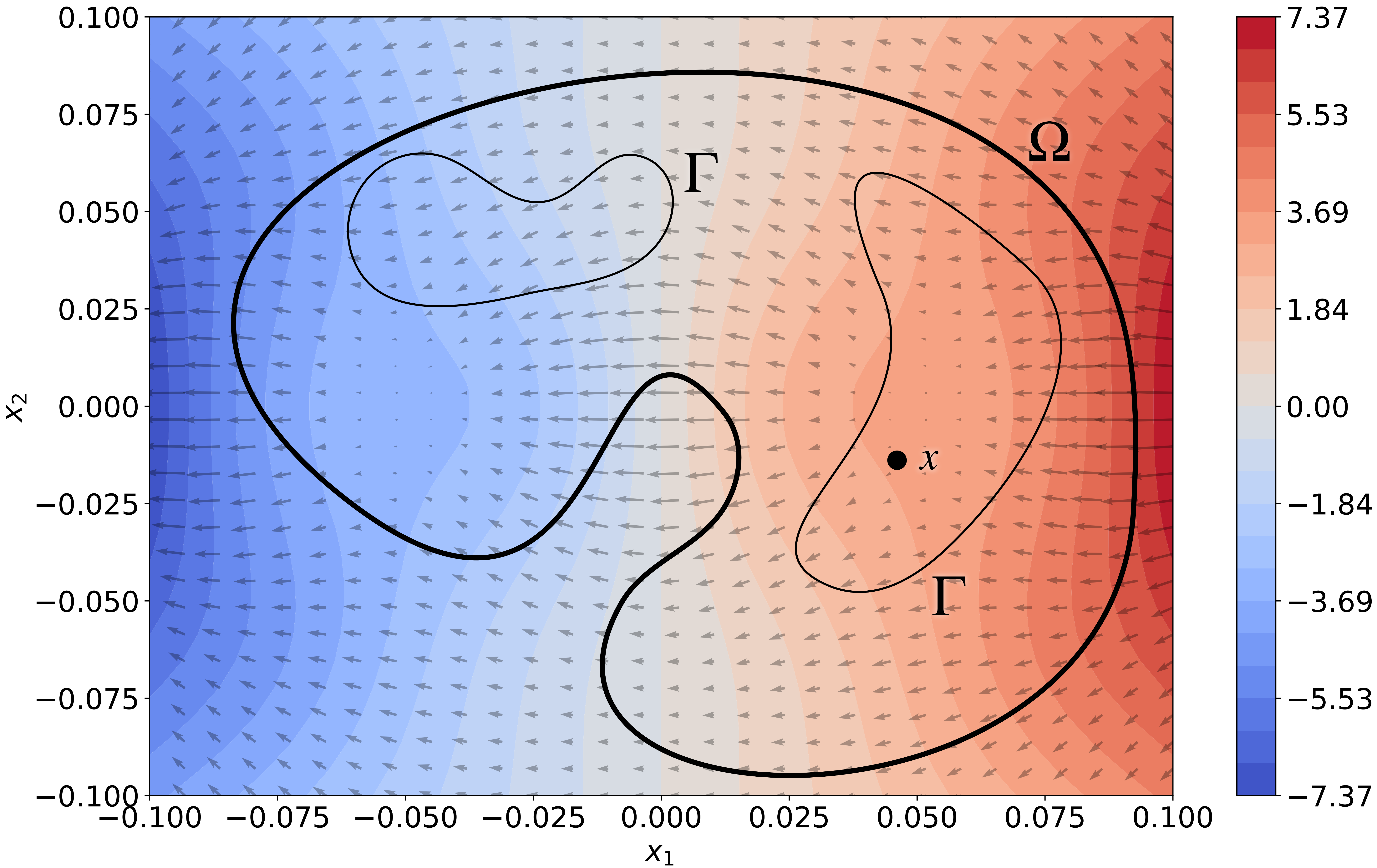
Side-effect Features



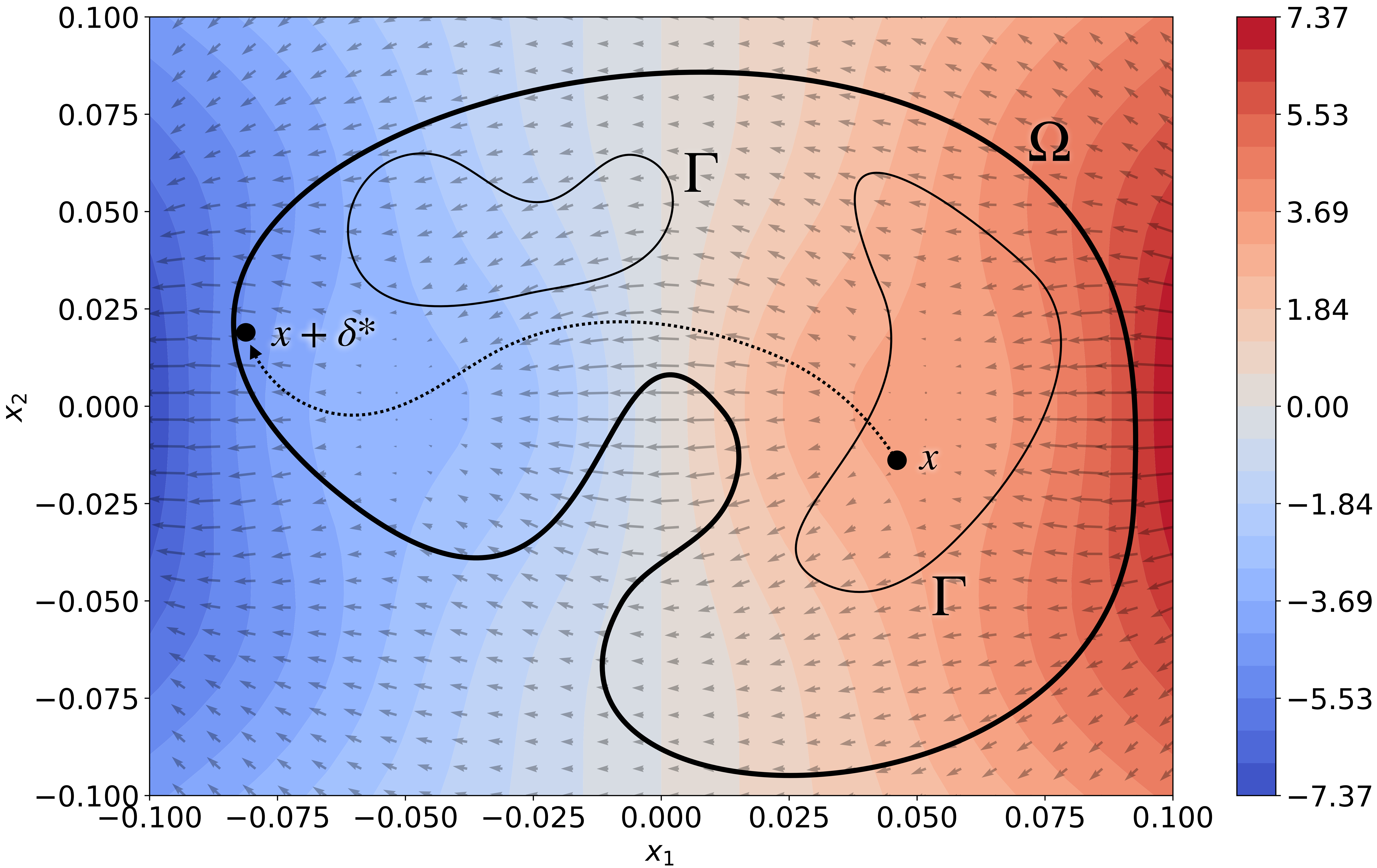
Side-effect Features



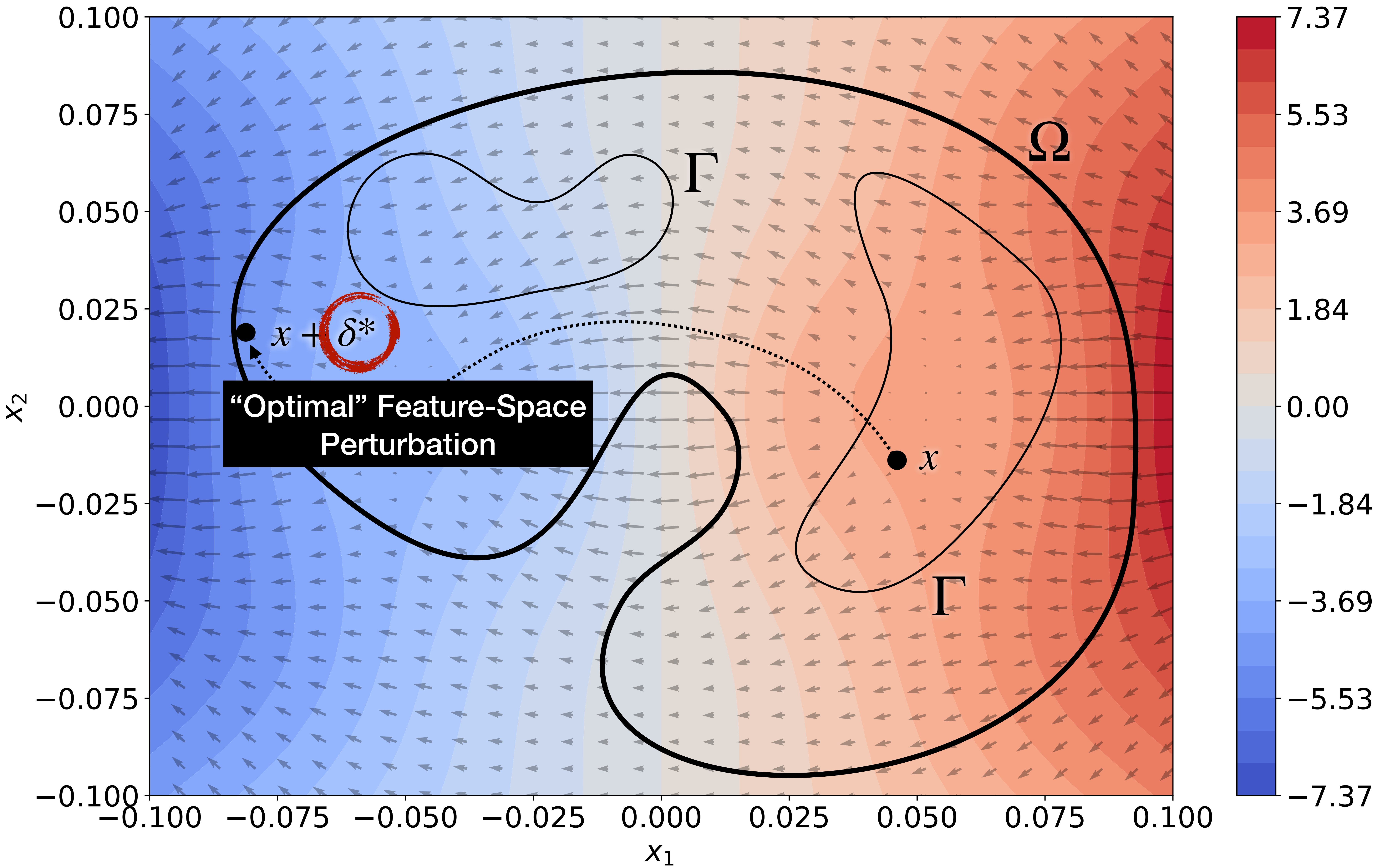
Side-effect Features



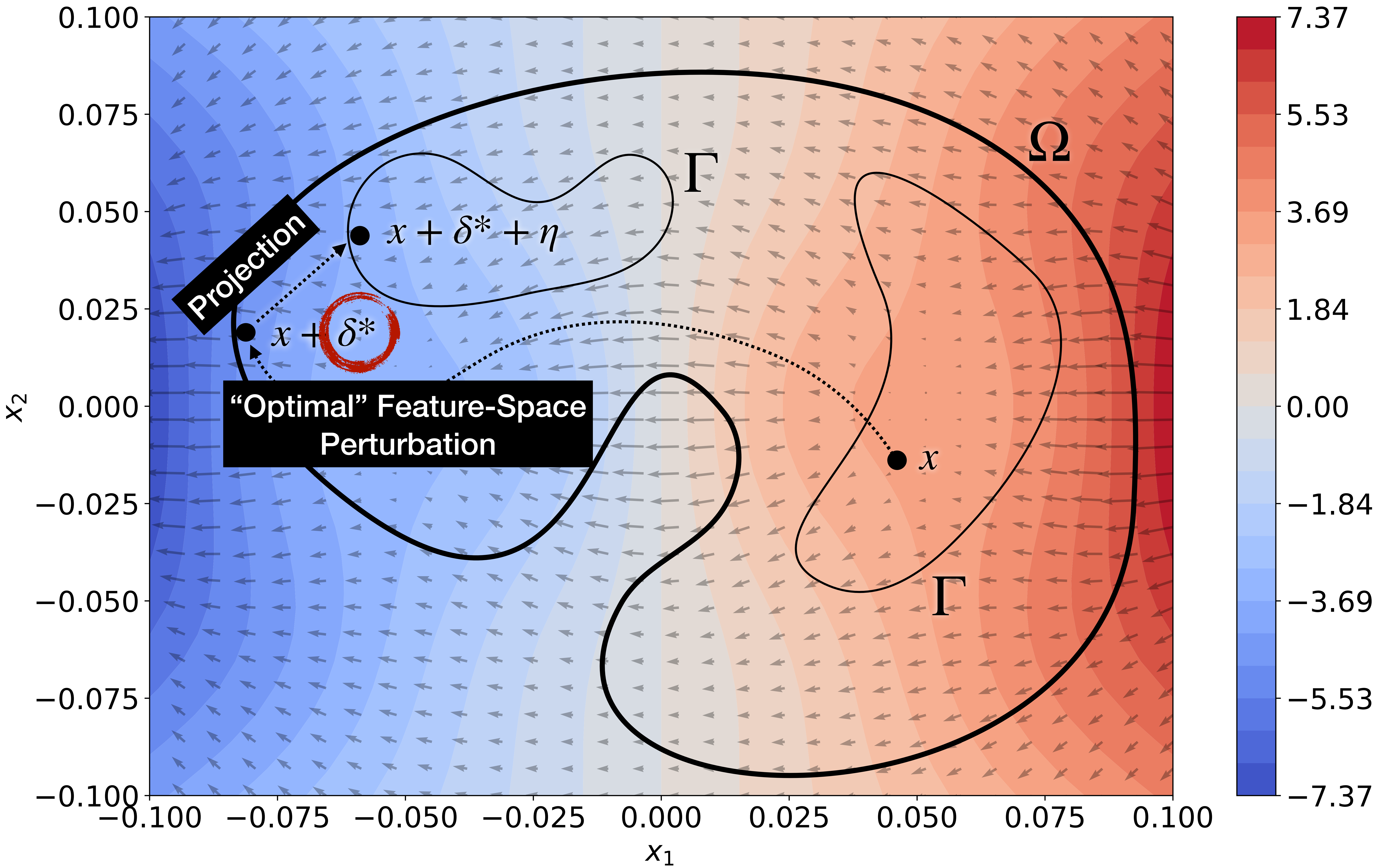
Side-effect Features



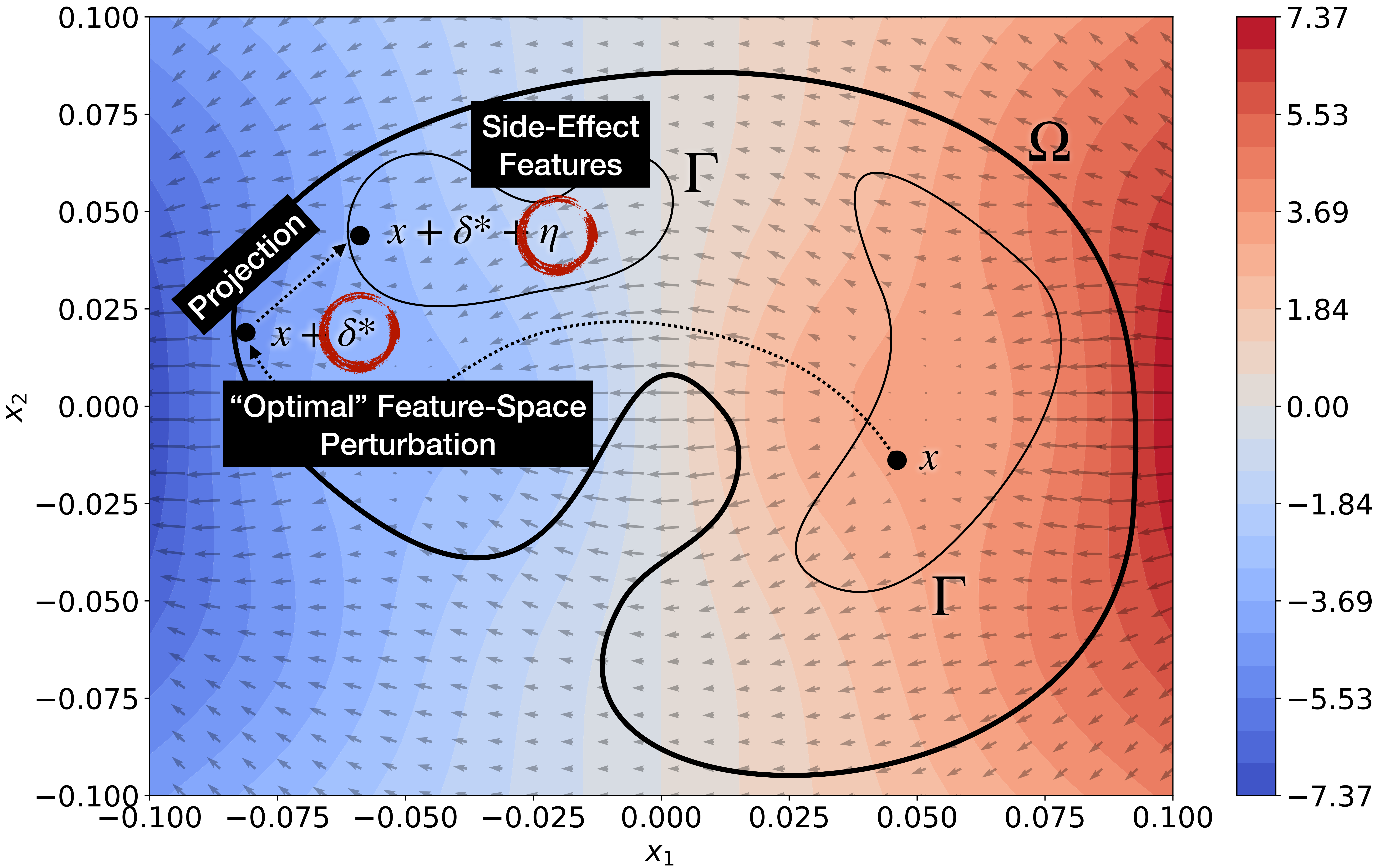
Side-effect Features



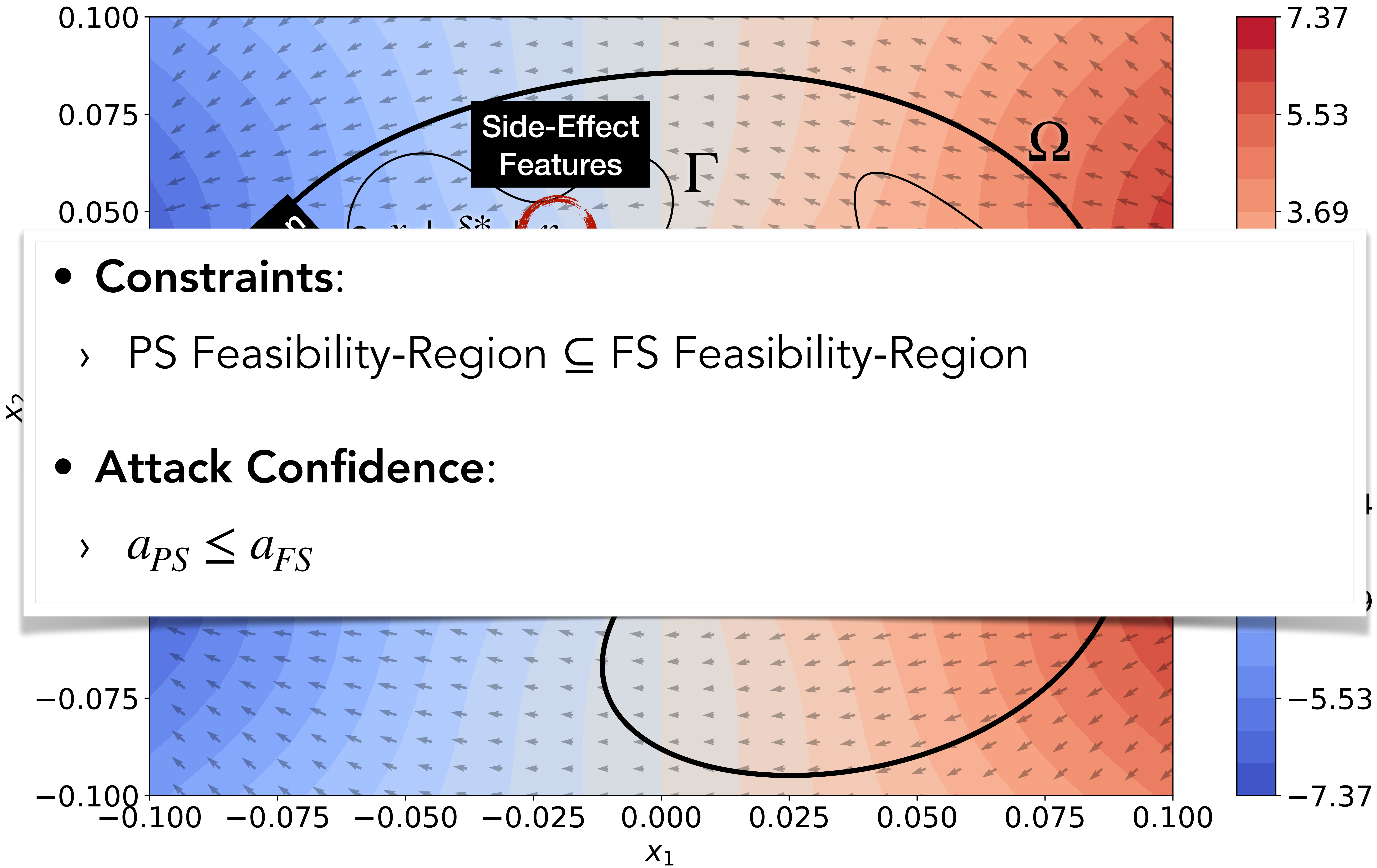
Side-effect Features



Side-effect Features



Side-effect Features



- **Constraints**:
  - PS Feasibility-Region ⊆ FS Feasibility-Region
- **Attack Confidence**:
  - $a_{PS} \leq a_{FS}$

Actionable Points

Actionable Points

Verify existence of feature-space attack

Necessary Condition for problem-space attacks

$\exists$ problem-space attack $\Longrightarrow$ $\exists$ feature-space attack

Proof 1 in paper

Actionable Points

SYSTEMS SECURITY RESEARCH LAB
KING'S College LONDON

Verify existence of feature-space attack

Necessary Condition for problem-space attacks

$\exists$ **problem-space attack** $\implies$ $\exists$ **feature-space attack**

Proof 1 in paper

Identify approximate inverse feature mapping

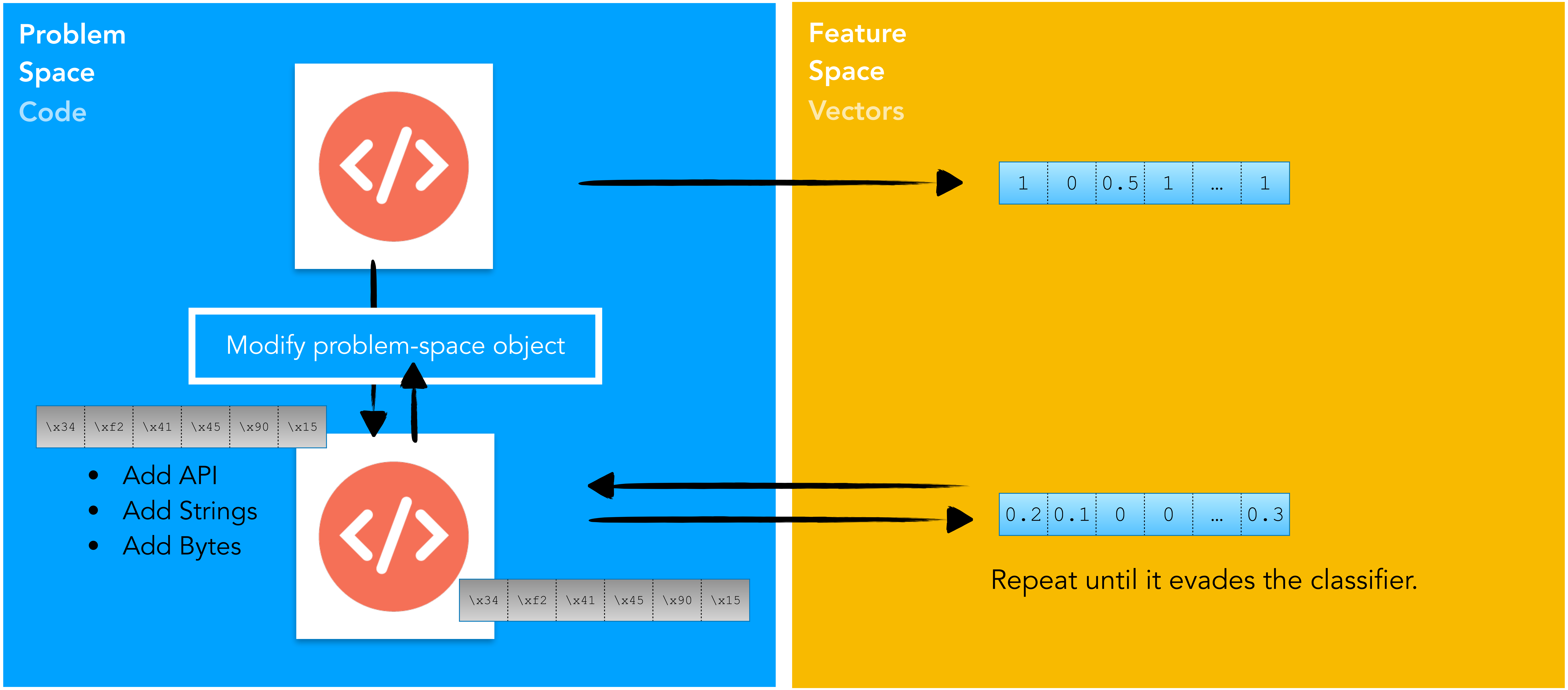
Sufficient Condition for problem-space attacks

$\exists$ **problem-space attack** $\impliedby$ $\exists$ **feature-space attack**, $\exists$ **approximate** φ^{-1}

Proof 2 in paper

Search Strategy

Problem-driven vs. Feature-Driven



Feature Space vs. Problem Space

$$\boldsymbol{\delta}^* = \arg\min_{\boldsymbol{\delta} \in \mathbb{R}^n} \quad f_t(\boldsymbol{x} + \boldsymbol{\delta})$$
$$\text{subject to:} \quad \boldsymbol{\delta} \models \Omega\,.$$

$$\begin{aligned} \text{argmin}_{\mathbf{T} \in \mathcal{T}} \quad & f_t(\varphi(\mathbf{T}(z))) = f_t(\boldsymbol{x} + \boldsymbol{\delta}^* + \boldsymbol{\eta}) \\ \text{subject to:} \quad & [\![z]\!]^\tau = [\![\mathbf{T}(z)]\!]^\tau, \quad \forall \tau \in \Upsilon \\ & \pi(\mathbf{T}(z)) = 1, \quad \forall \pi \in \Pi \\ & \mathbf{A}(\mathbf{T}(z)) = \mathbf{T}(z), \quad \forall \mathbf{A} \in \Lambda \end{aligned}$$

Feature-Space Constraints
- Lp perturbations
- Domain constraints for vectors

Search Strategy
- Feature-driven

Problem-Space Constraints
- Available Transformations
- Preserved Semantics
- Plausibility
- Robustness to Preprocessing

Search Strategy
- Feature-driven
- Problem-driven
- Hybrid

Feature Space vs. Problem Space

$$\boldsymbol{\delta}^* = \arg\min_{\boldsymbol{\delta} \in \mathbb{R}^n} \quad f_t(\boldsymbol{x} + \boldsymbol{\delta})$$
$$\text{subject to:} \quad \boldsymbol{\delta} \models \Omega\,.$$

$$\operatorname{argmin}_{\mathbf{T} \in \mathcal{T}} \quad f_t(\varphi(\mathbf{T}(z))) = f_t(\boldsymbol{x} + \boldsymbol{\delta}^* + \boldsymbol{\eta})$$
$$\text{subject to:} \quad [\![z]\!]^{\tau} = [\![\mathbf{T}(z)]\!]^{\tau}, \quad \forall \tau \in \Upsilon$$
$$\pi(\mathbf{T}(z)) = 1, \quad \forall \pi \in \Pi$$
$$\mathbf{A}(\mathbf{T}(z)) = \mathbf{T}(z), \quad \forall \mathbf{A} \in \Lambda$$

Feature-Space Constraints

- Lp perturbations
- Domain constraints for vectors

Search Strategy

- Feature-driven

Problem-Space Constraints

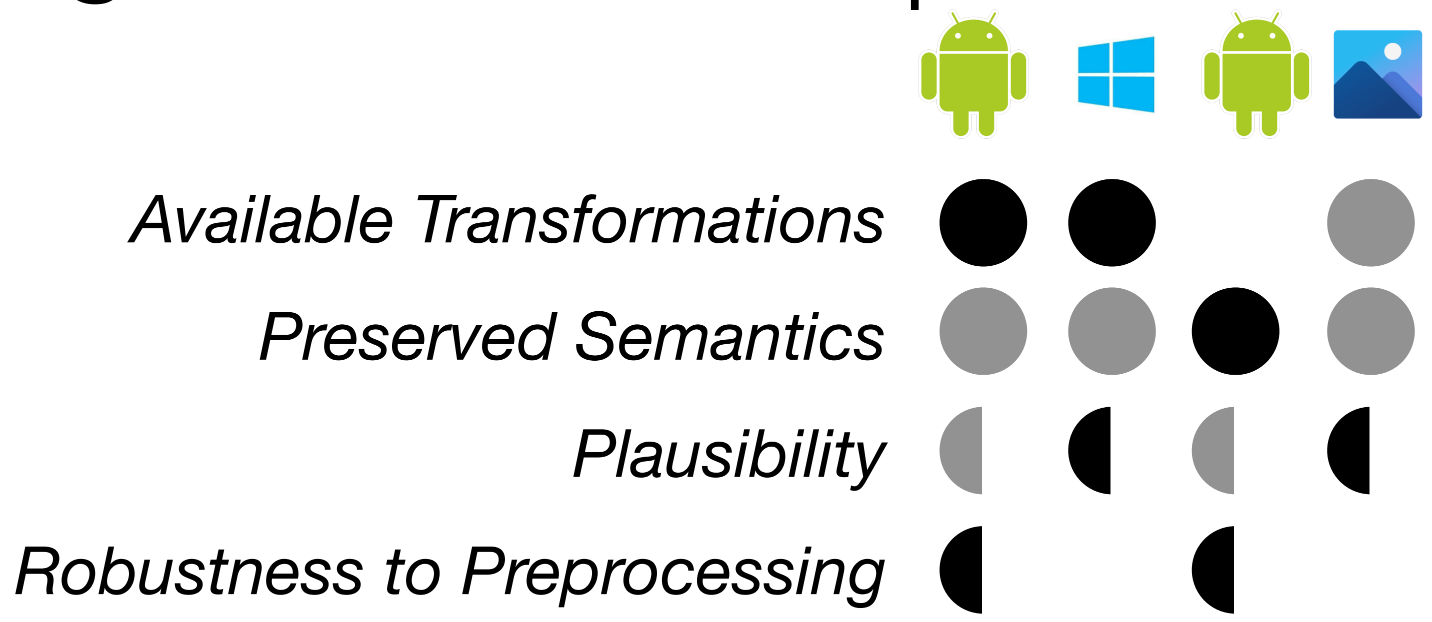
- Available Transformations
- Preserved Semantics
- Plausibility
- Robustness to Preprocessing

Search Strategy

- Feature-driven
- Problem-driven
- Hybrid

Works for Multiple Domains

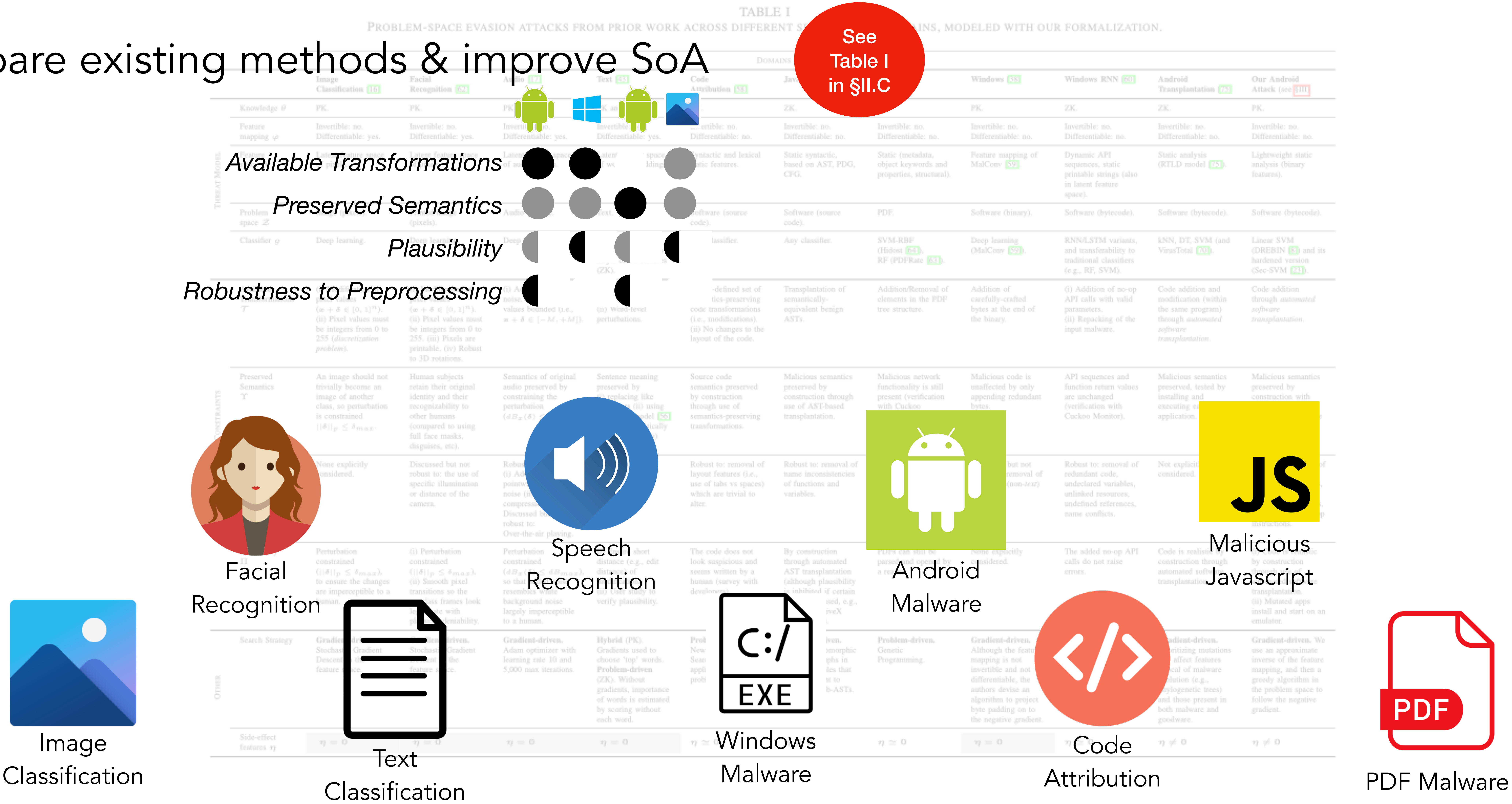
Compare existing methods & improve SoA



See Table I in §II.C

Works for Multiple Domains

Compare existing methods & improve SoA



Android Attack

Prior Work on Adv. Malware

Prior work was fundamental to initially explore problem-space attacks.
We propose a principled approach that supports reasoning.

Available Transformations

- Limiting #features modified [ESORICS'17]

Robustness to Preprocessing

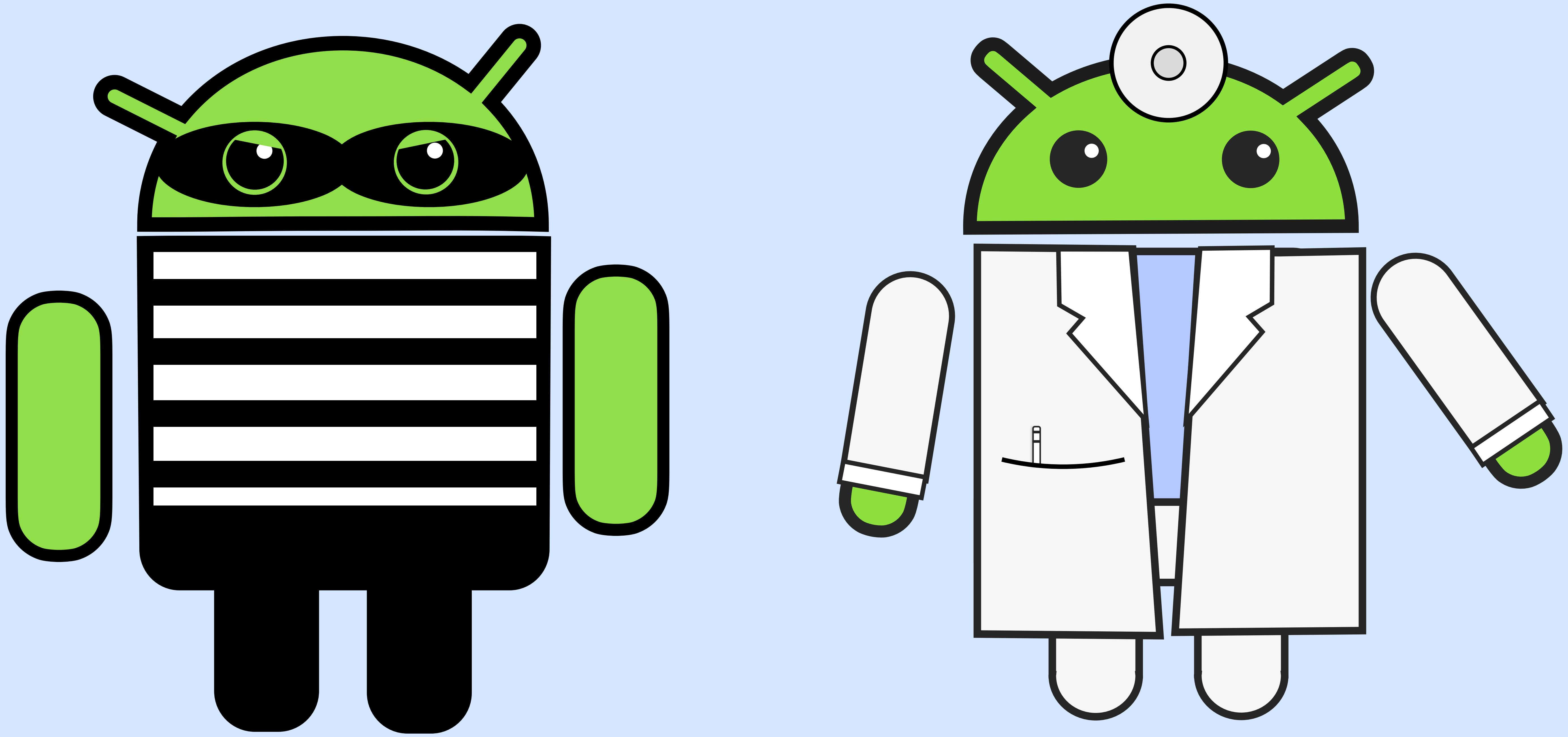
- Removable unused permissions [ESORICS'17]
- Removal code (unreachable, no-op) [EUSIPCO'18, RAID'18]
- Unclear [ACSAC'18]

Preserved Semantics

- Highly unstable transformations [ACSAC'18]

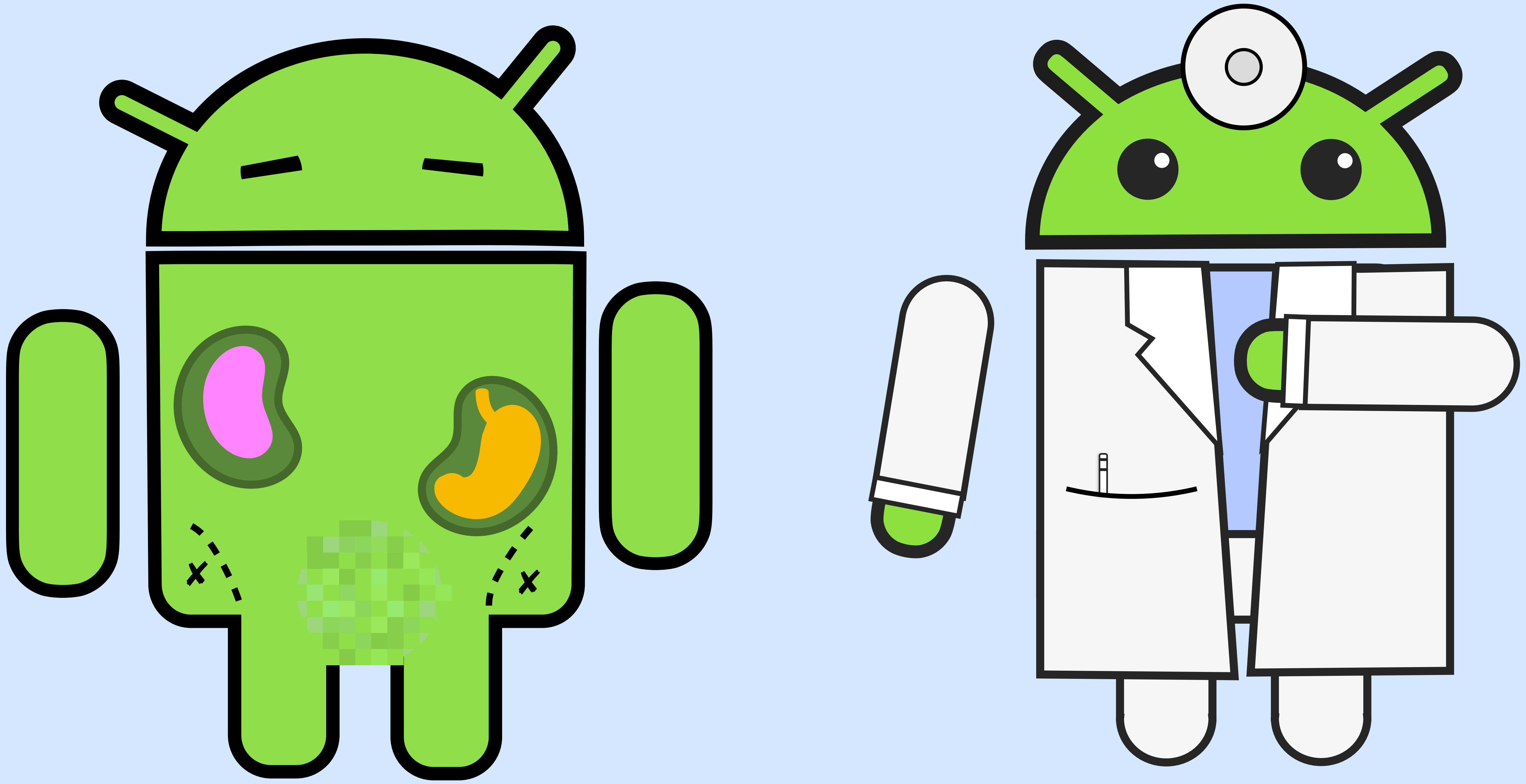
Our Android Attack

Our Android Attack



Our Android Attack

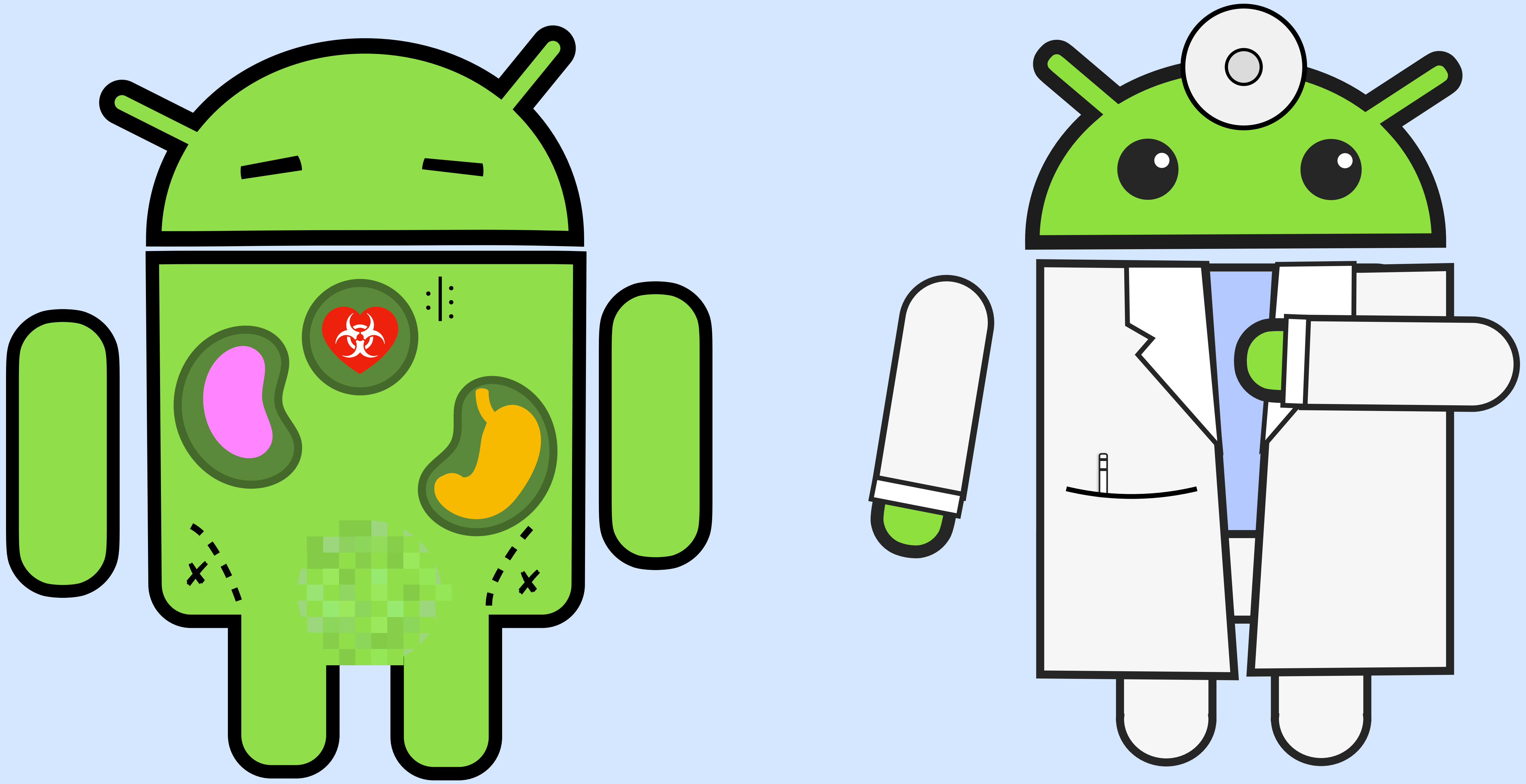
Available Transformations
Code addition through automated software transplantation.



Our Android Attack

Available Transformations
Code addition through automated software transplantation.

Preserved Semantics
Malicious semantics preserved by construction using opaque predicates (inserted code is not executed at runtime).

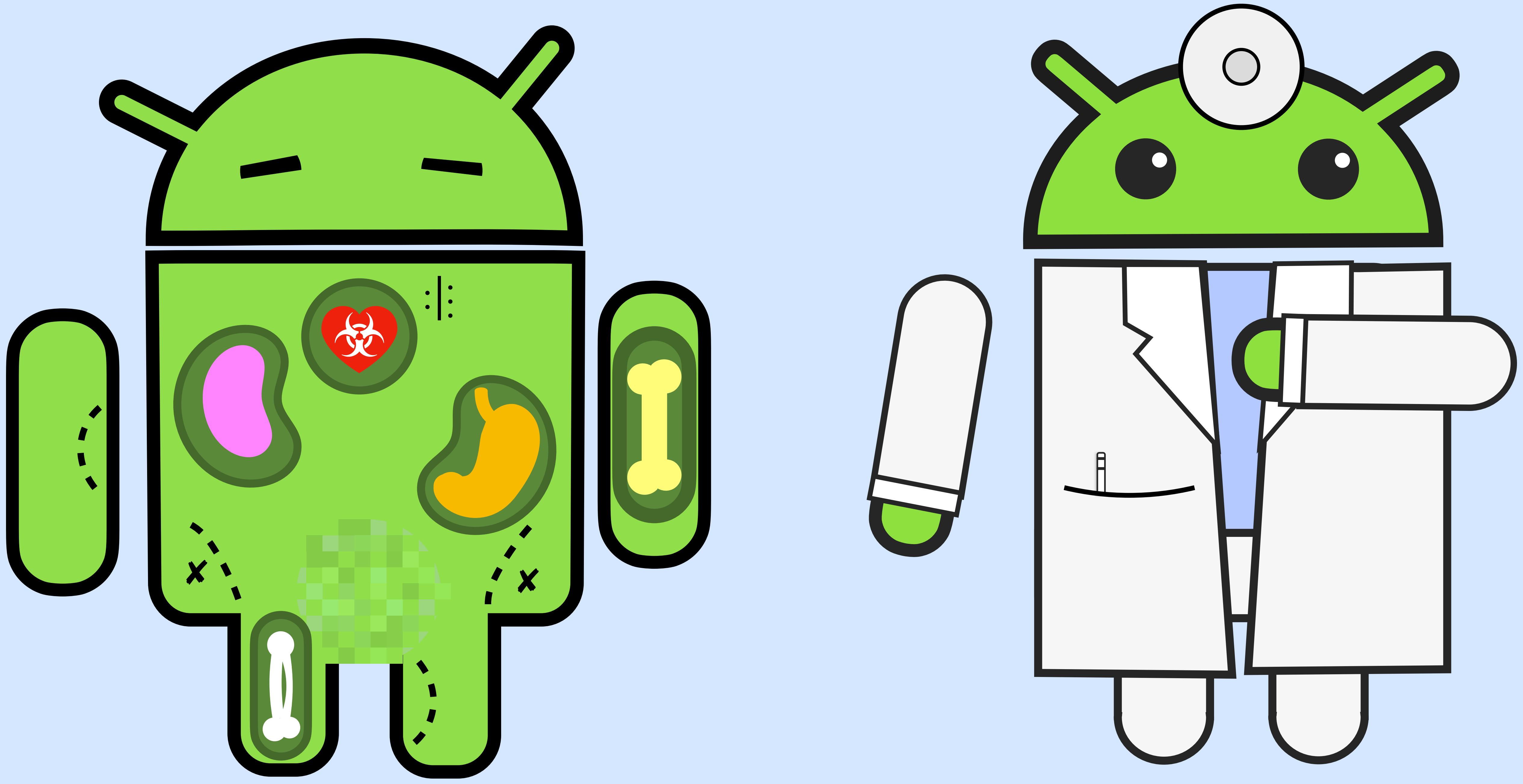


Our Android Attack

Available Transformations
Code addition through automated software transplantation.

Preserved Semantics
Malicious semantics preserved by construction using opaque predicates (inserted code is not executed at runtime).

Robustness to Preprocessing
We're robust to:
- removal of redundant code
- undeclared variables
- unlinked resources
- undefined references
- naming conflicts
- no-op instructions.



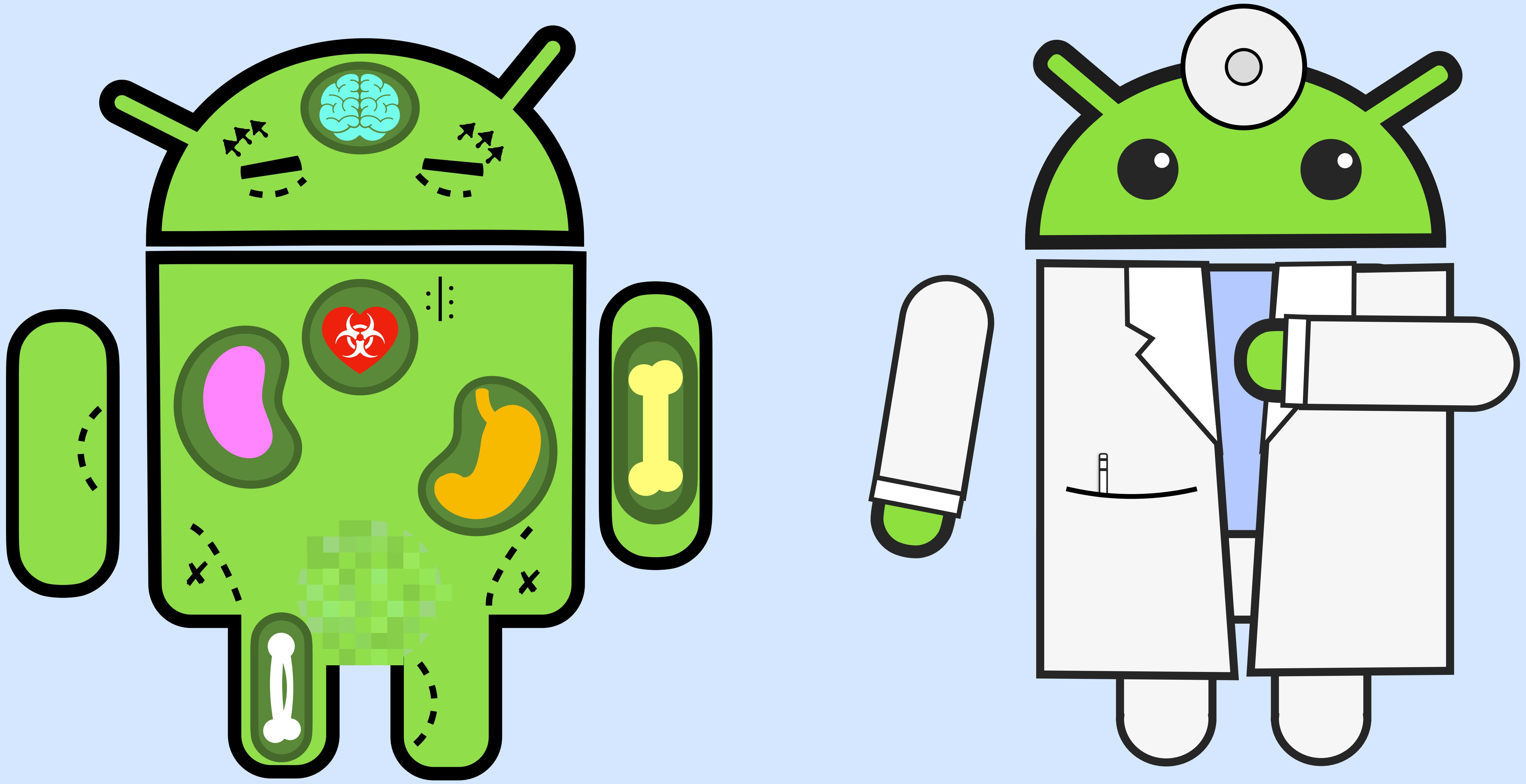
Our Android Attack

Available Transformations
Code addition through automated software transplantation.

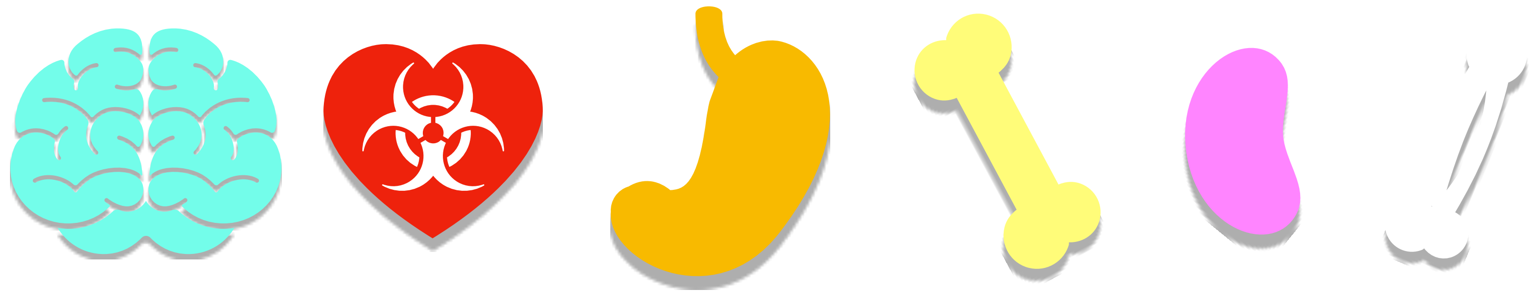
Preserved Semantics
Malicious semantics preserved by construction using opaque predicates (inserted code is not executed at runtime).

Robustness to Preprocessing
We're robust to:
- removal of redundant code
- undeclared variables
- unlinked resources
- undefined references
- naming conflicts
- no-op instructions.

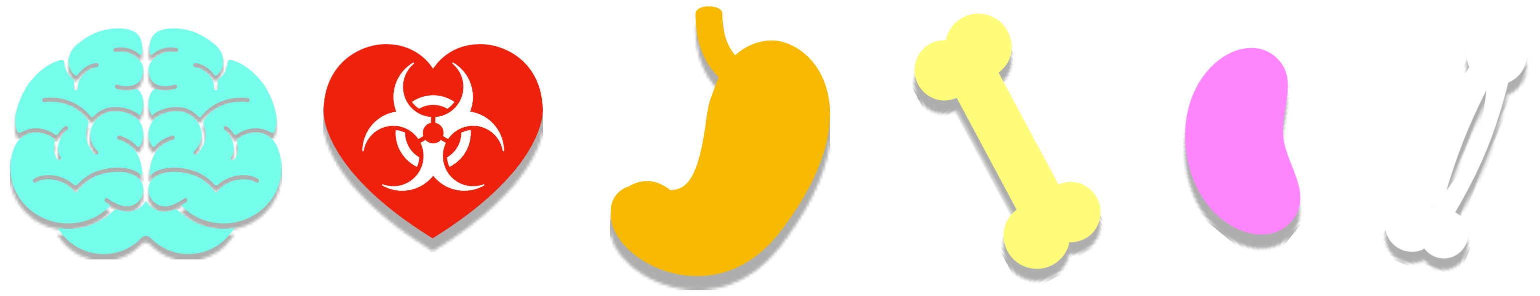
Plausibility
Only realistic code is injected (rather than orphaned urls, api calls, etc.)
Mutated apps install and start on an emulator.

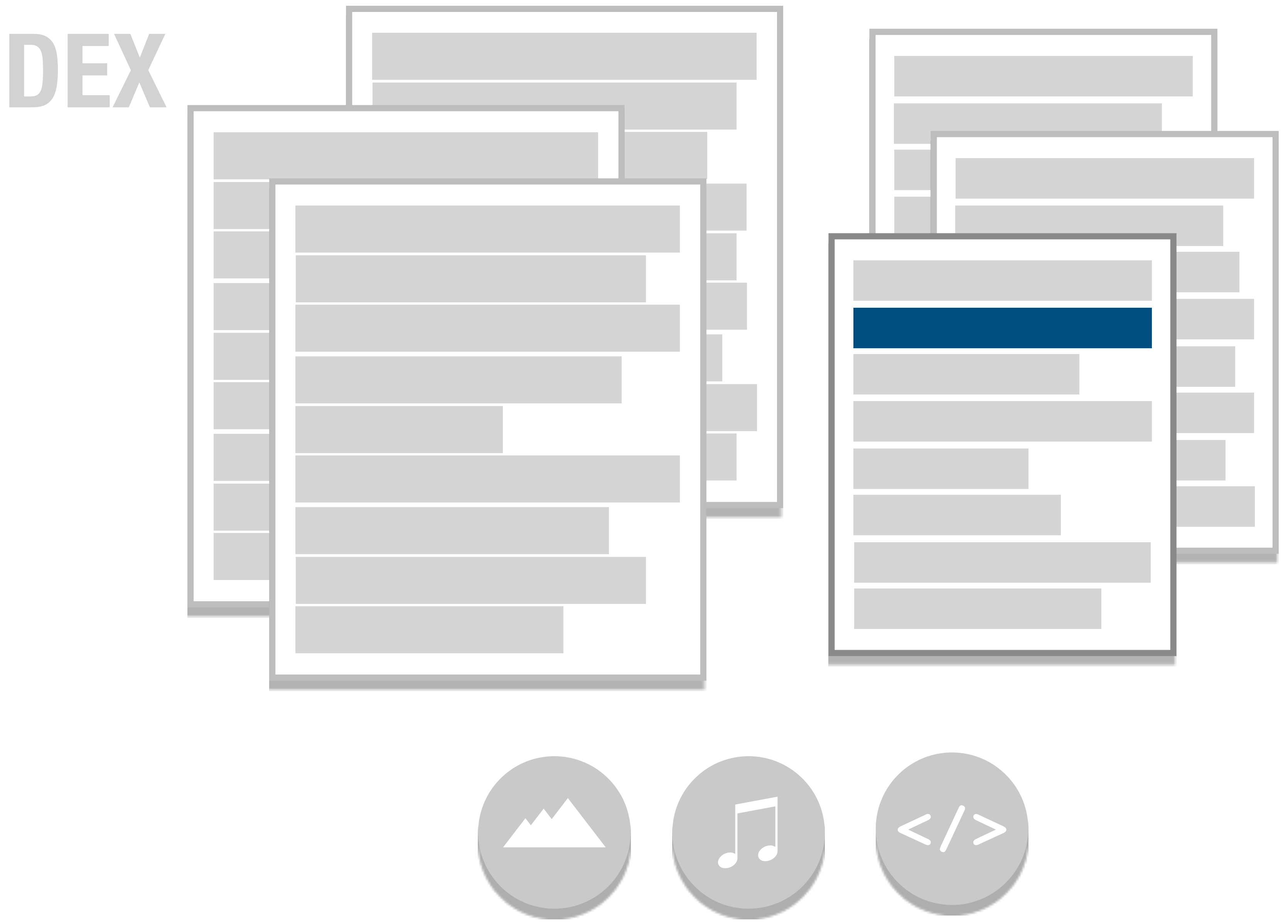


Organ Harvesting



Organ Harvesting

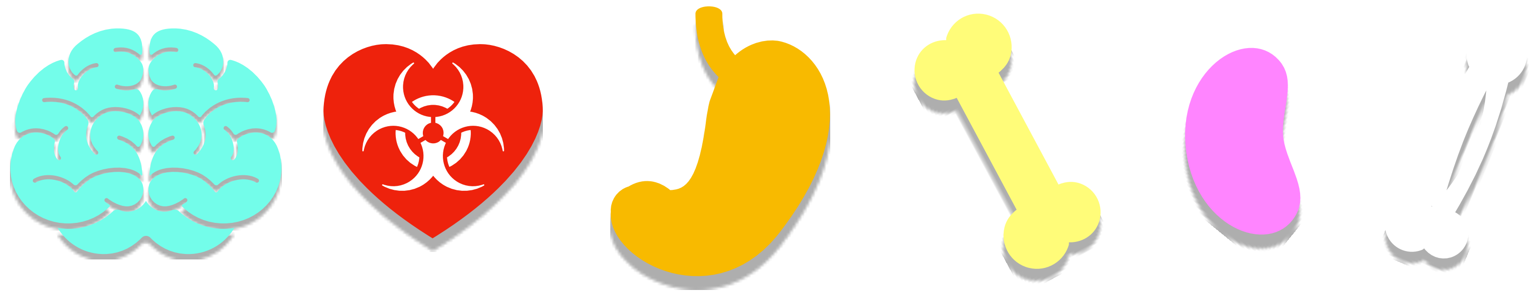


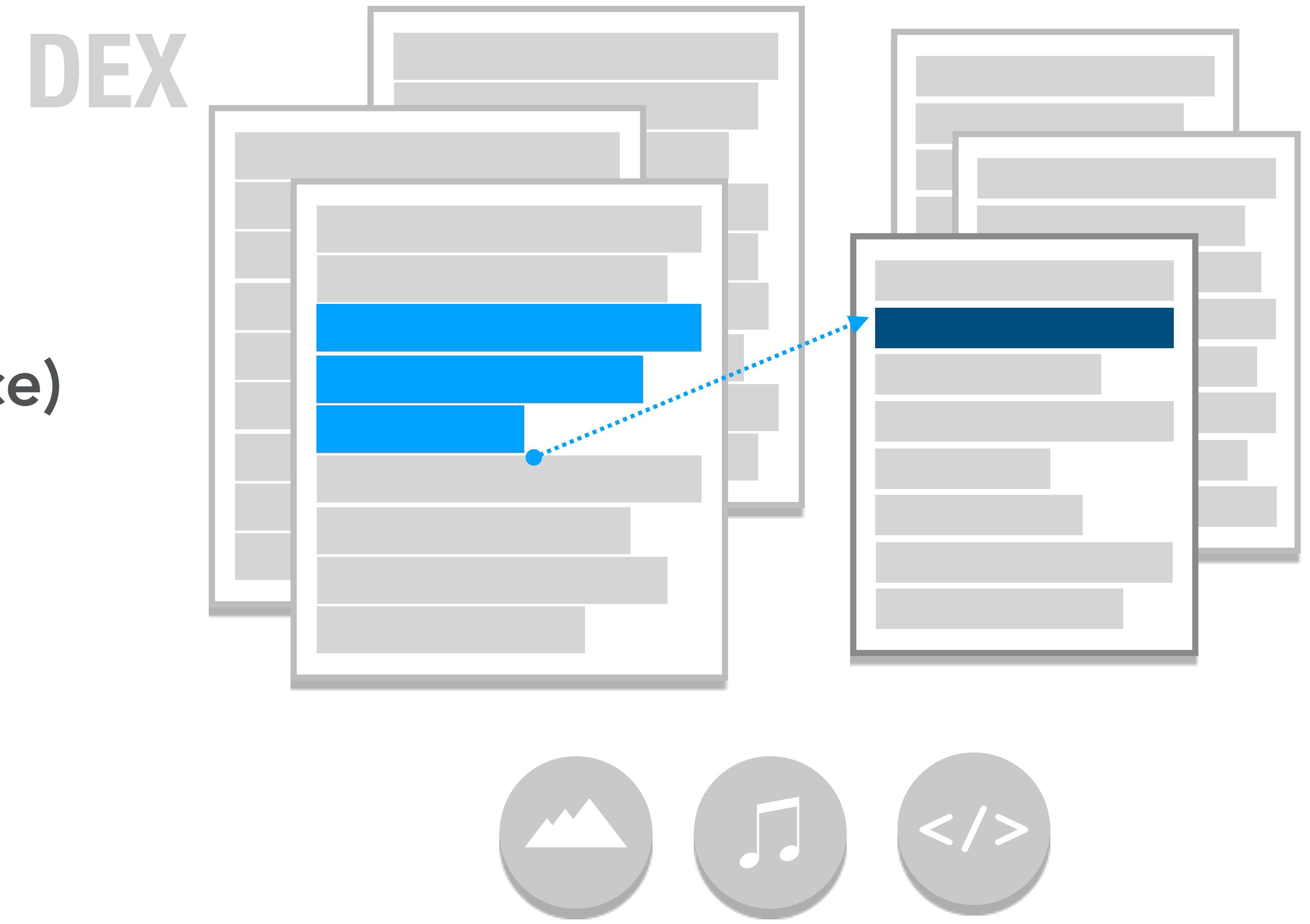


❶ Identify feature entry point

Identify activity in dex

Organ Harvesting

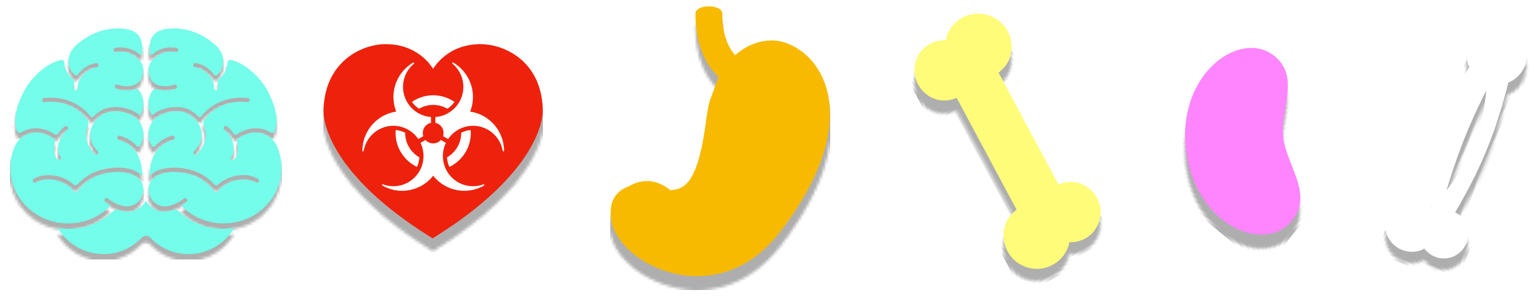


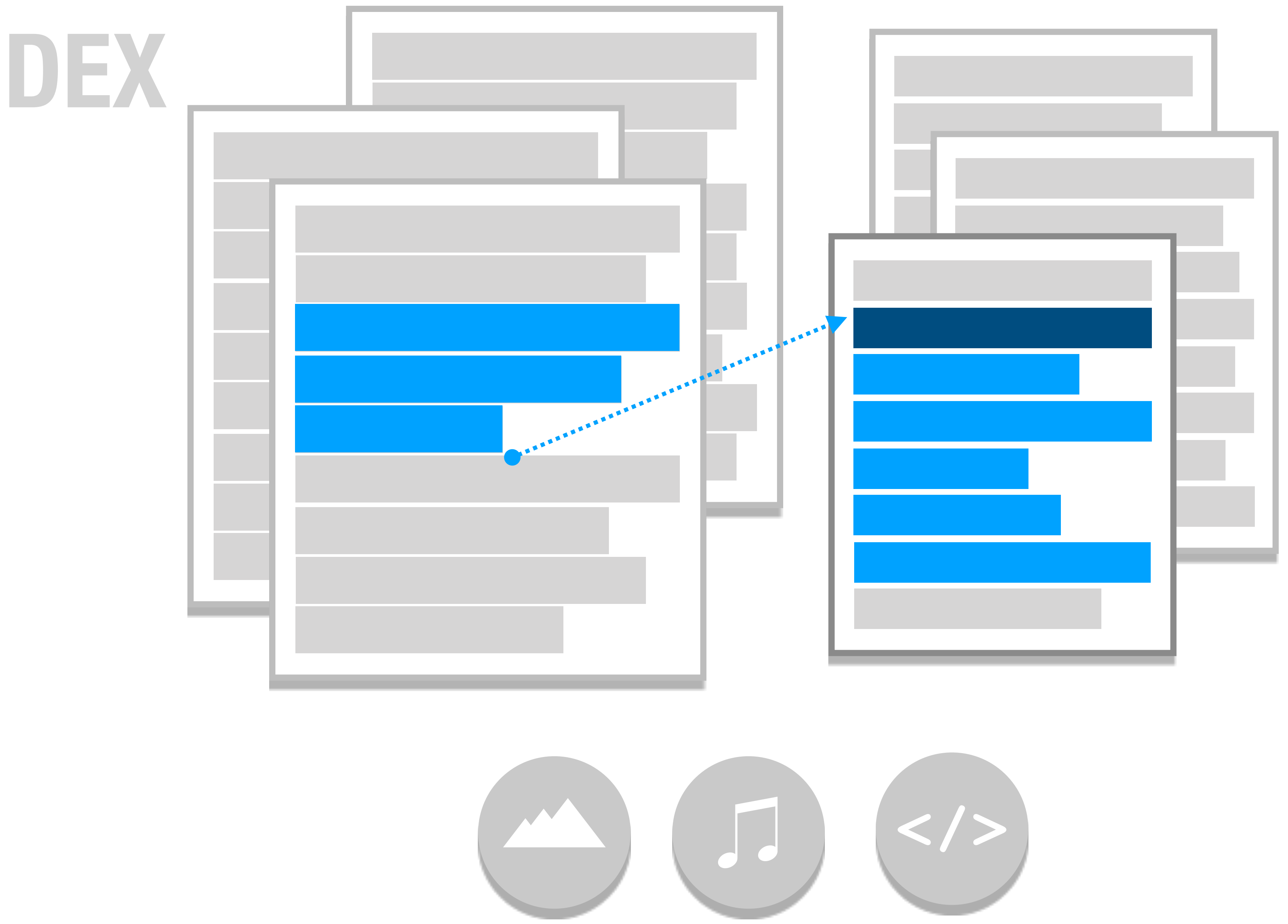


2 Choose any vein (backward slice)

Extract intent creation and startActivity()

Organ Harvesting

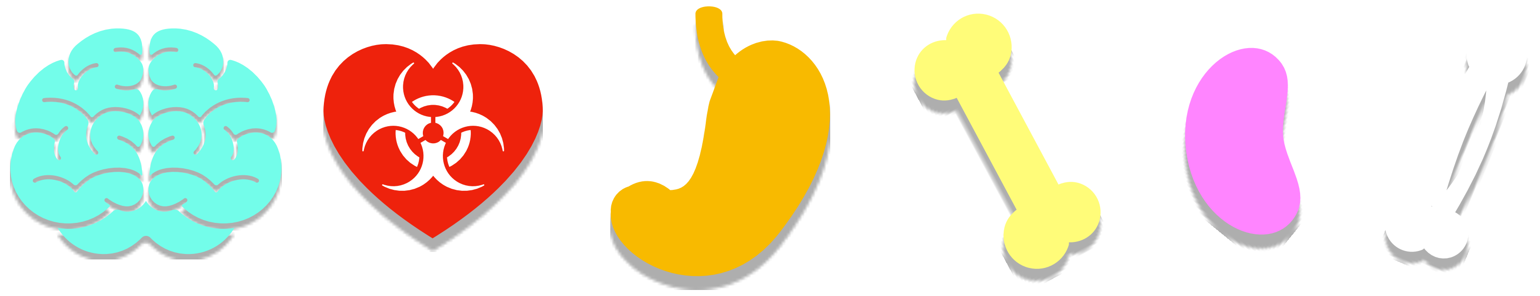




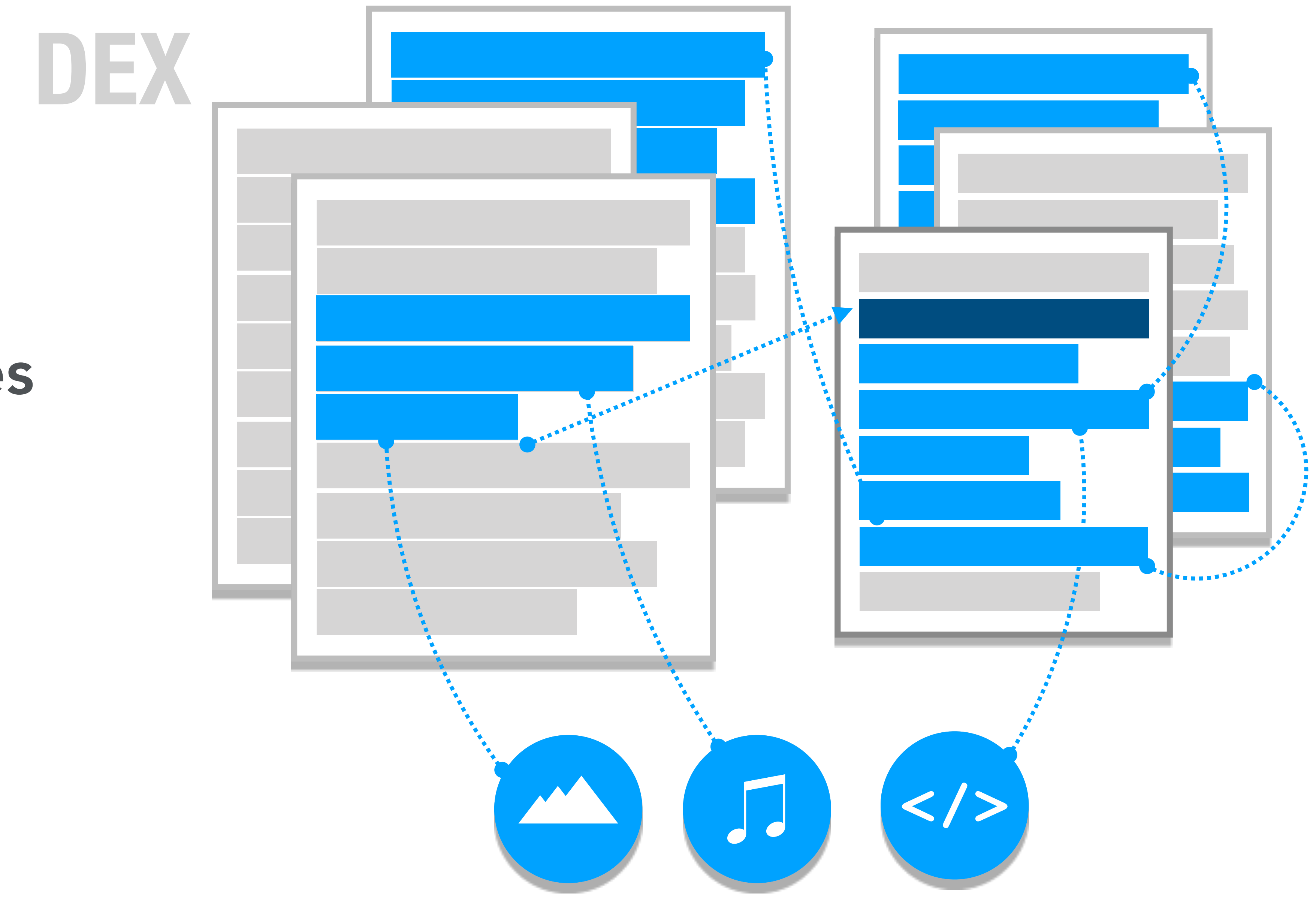
❸ Collect organ (forward slice)

Gather activity definition

Organ Harvesting

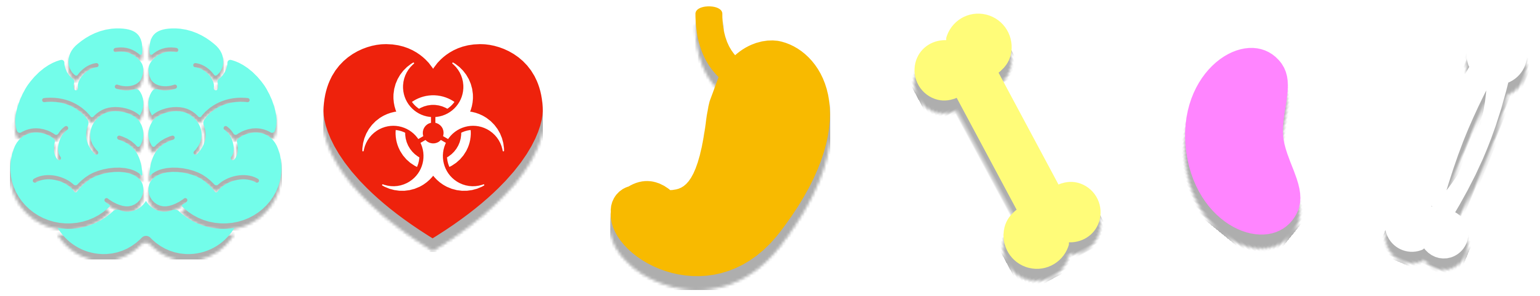


4 Include transitive dependencies

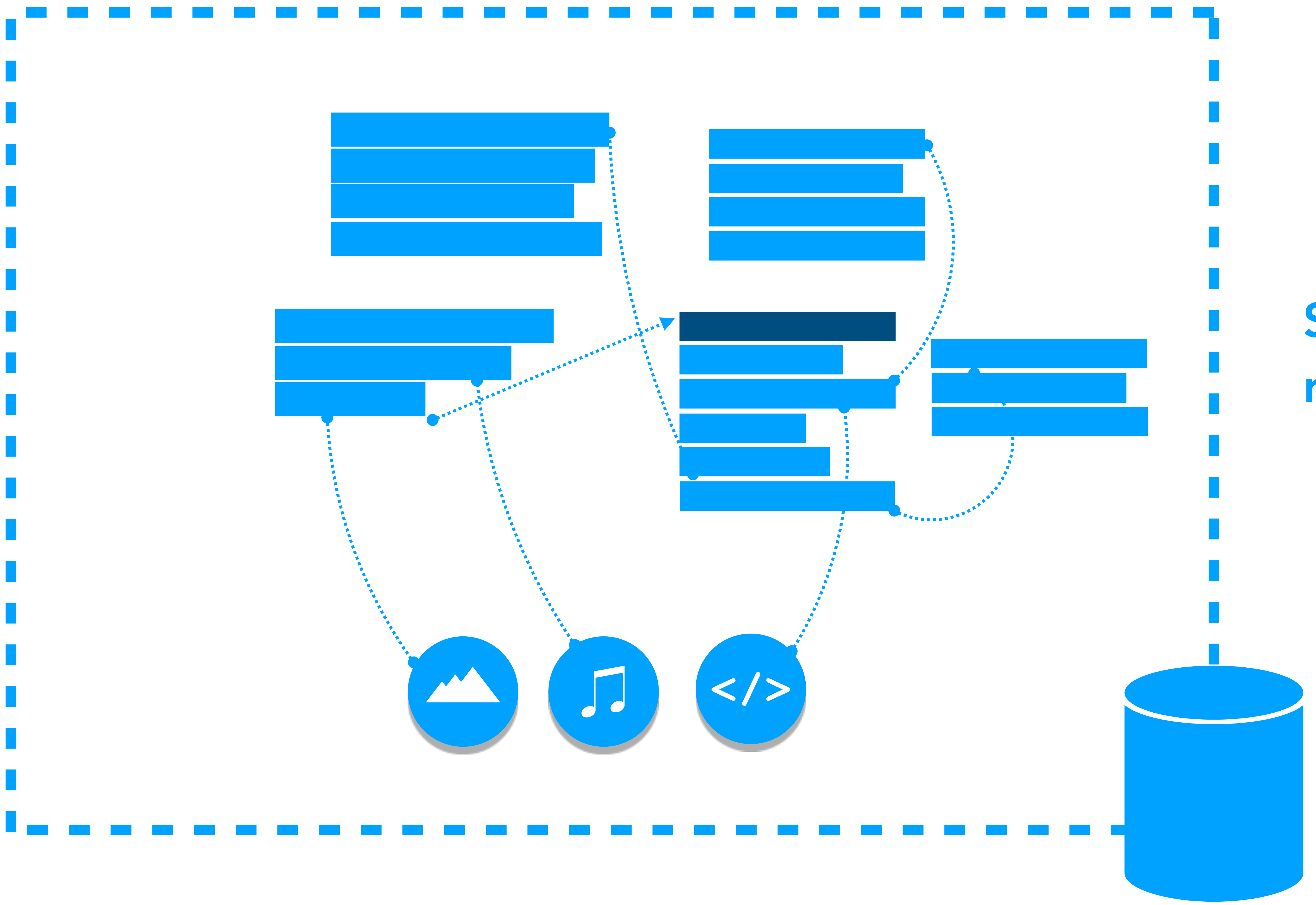


Recursively collect dependencies

Organ Harvesting

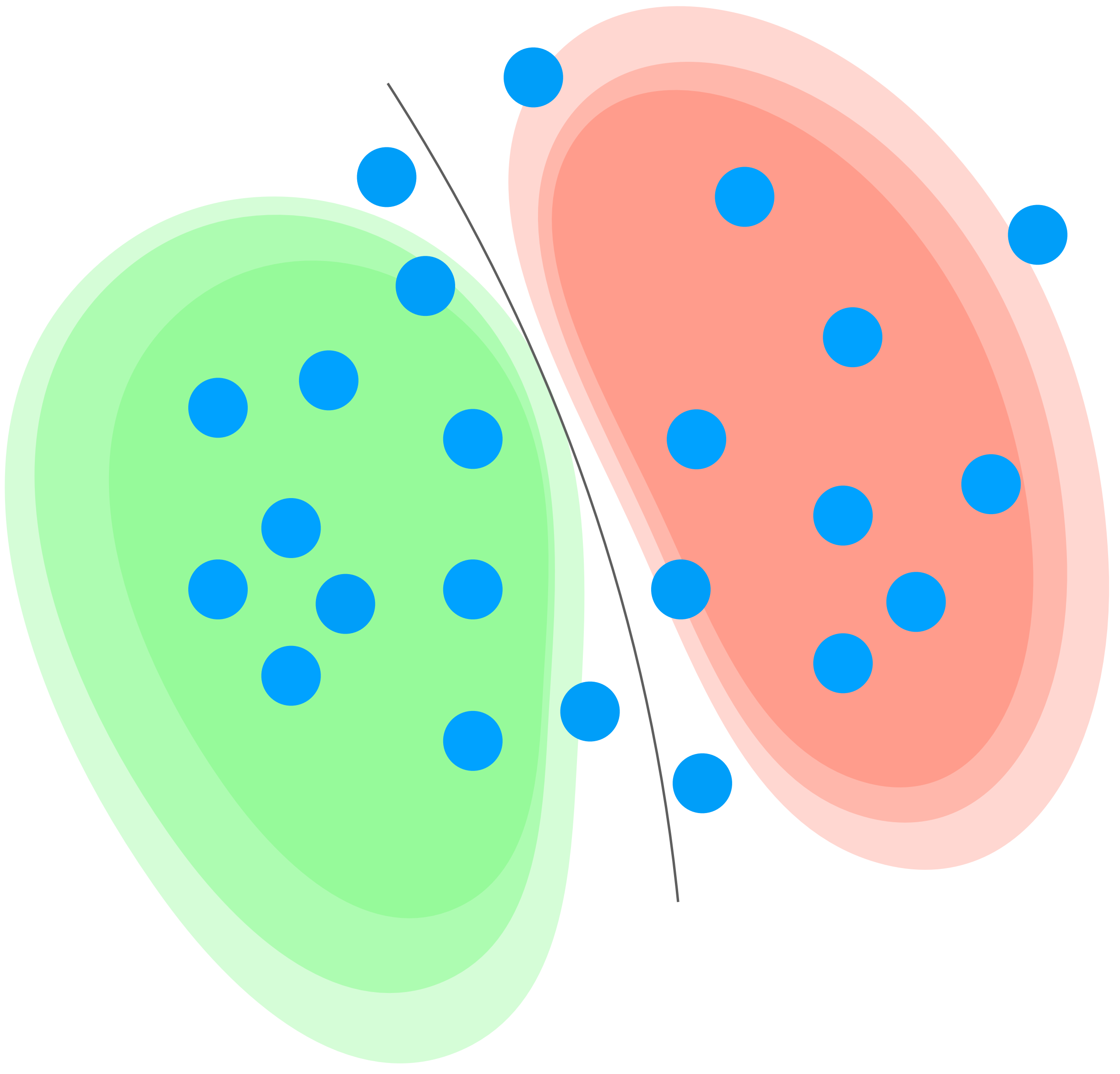


5 Store organ in an “ice box”



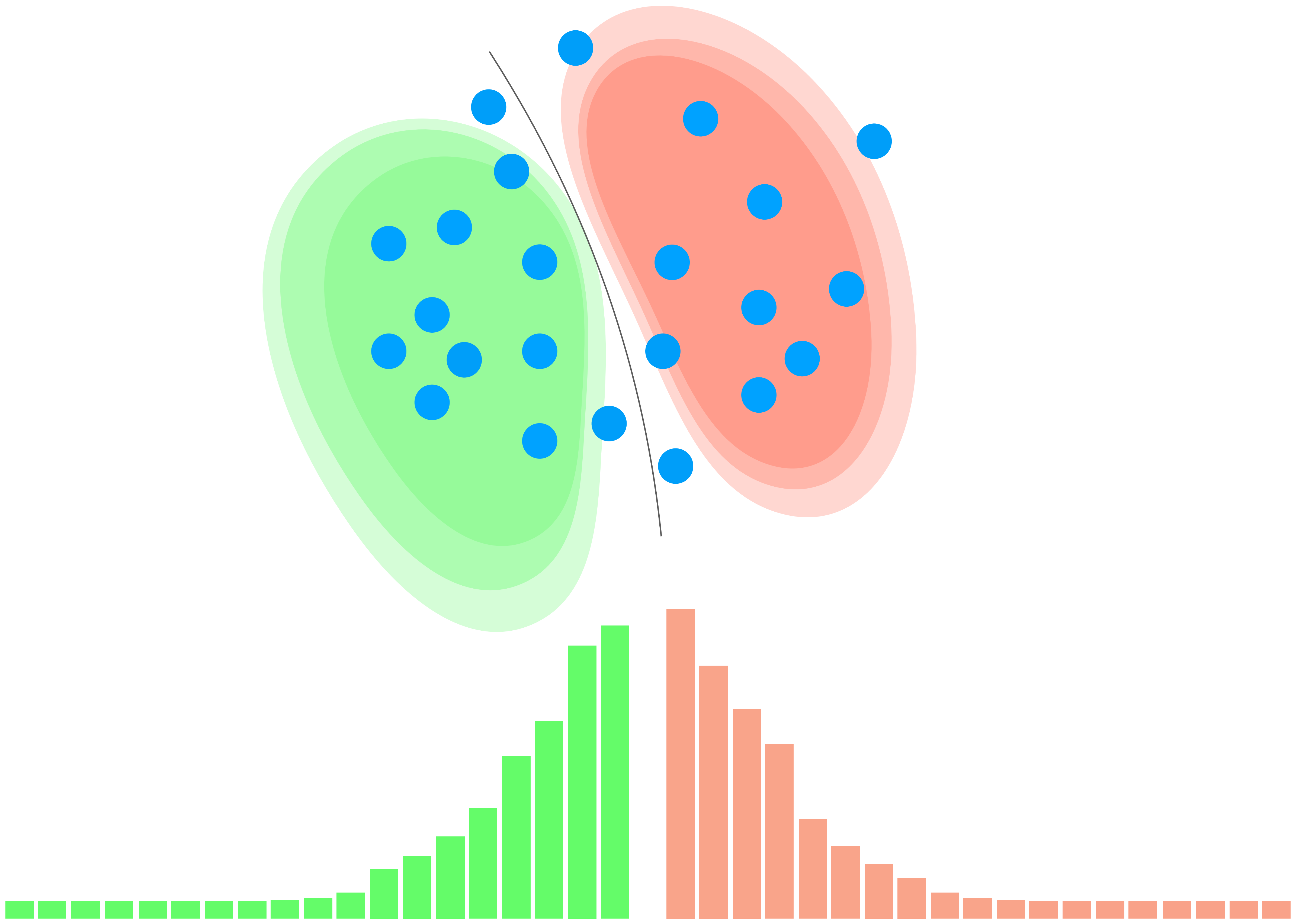
Save gadget to a database ready for the attack

Attack Overview



Attack Overview

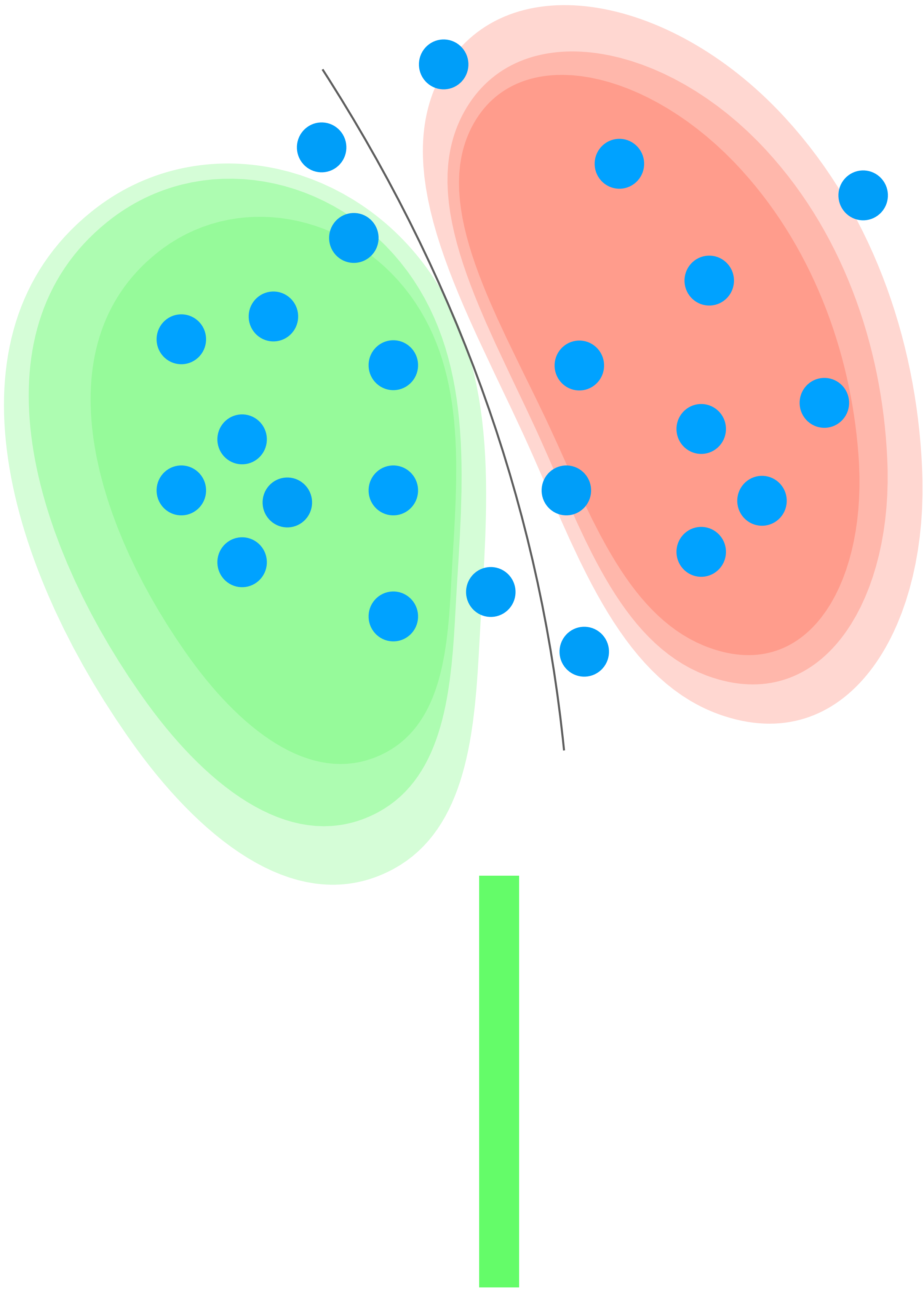
Given a trained target model



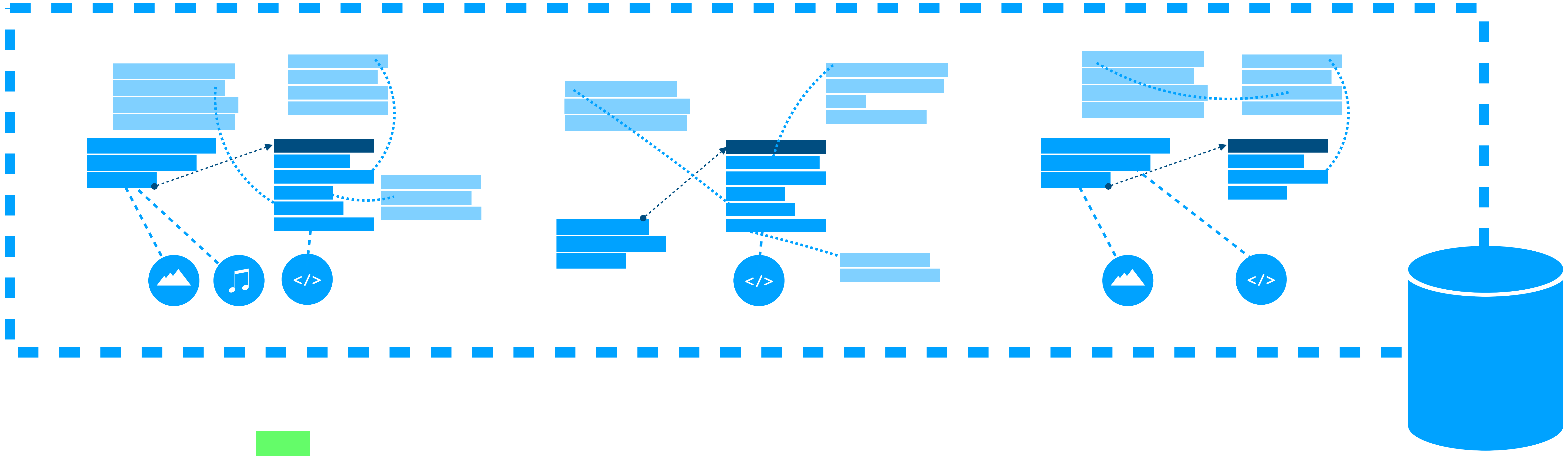
Attack Overview

Given a trained target model

First pick feature with greatest 'benign' weight



Attack Overview



Given a trained target model

First pick feature with greatest 'benign' weight

Find a corresponding organ from the ice box

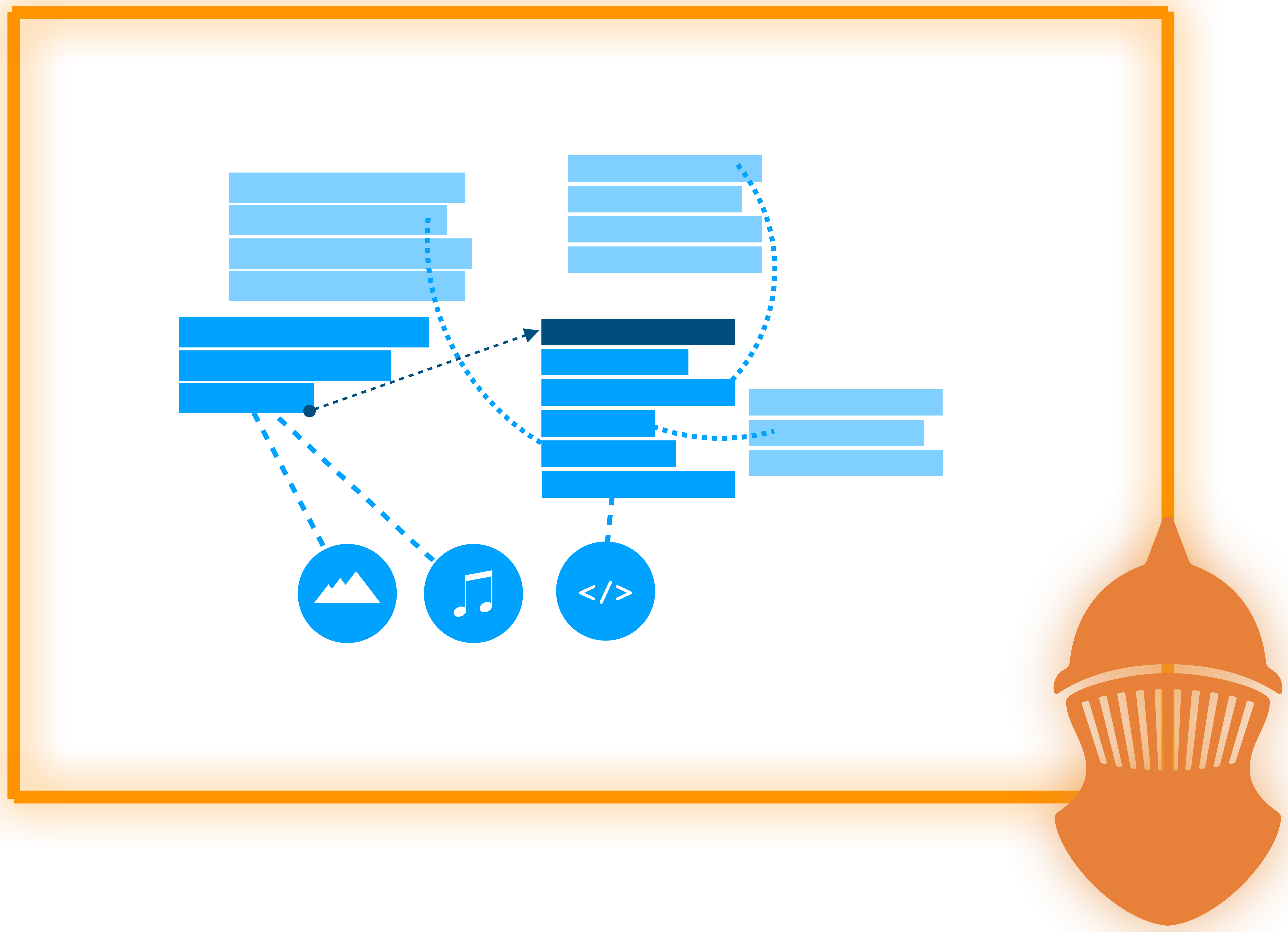
Attack Overview

Given a trained target model

First pick feature with greatest 'benign' weight

Find a corresponding organ from the ice box

Wrap the organ in an opaque predicate



Attack Overview

Given a trained target model

First pick feature with greatest 'benign' weight

Find a corresponding organ from the ice box

Wrap the organ in an opaque predicate

Inject the new benign code and repackage

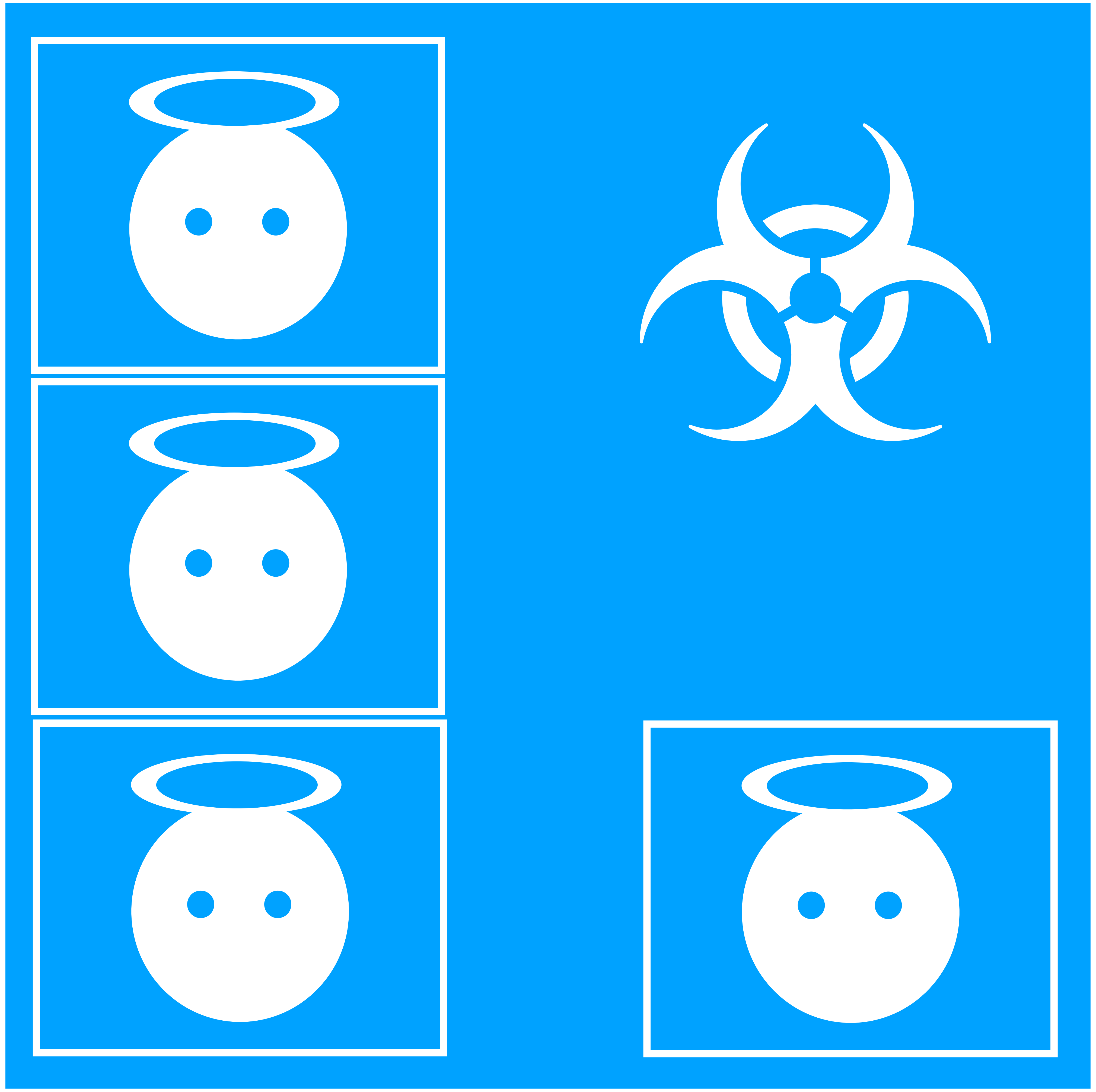
Opaque Predicates

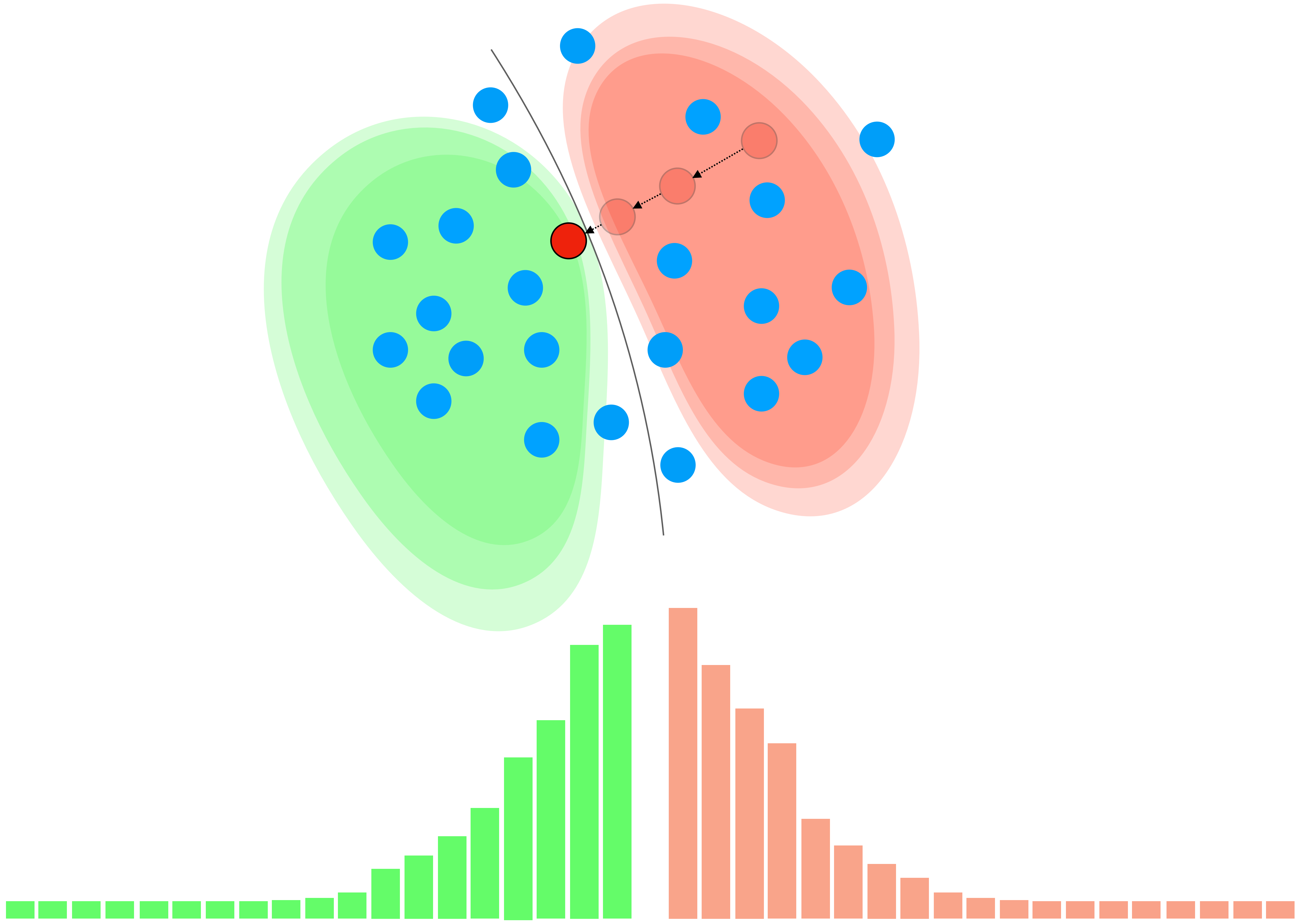
```java
void opaque() {
  Random random = new Random();
  this();
  boolean[] arrayOfBoolean = new boolean[40];
  byte b1;
  for (b1 = 0; b1 < arrayOfBoolean.length; b1++)
    arrayOfBoolean[b1] = random.nextBoolean();
  b1 = 1;
  for (byte b2 = 0; b2 < 184.0D; b2++) {
    boolean bool = false;
    for (byte b = 0; b < 3; b++)
      bool |= arrayOfBoolean[random.nextInt(
              arrayOfBoolean.length)];
    if (!bool)
      b1 = 0;
  }
  if (b1 != 0) {

    // Beginning of adapted vein
    Context context = ((Context)this).
         getApplicationContext();
    Intent intent = new Intent();
    this(this, h.a(this, cxim.qngg.TEhr.sFiQa.class));
    intent.putExtra("1", h.p(this));
    intent.addFlags(268435456);
    startActivity(intent);
    h.x(this);
    return;
    // End of adapted vein

  }
}
```

- Example of opaque predicate in **JSketch**
  - Opaque predicate wraps an *adapted vein*
  - Random k-SAT parameters

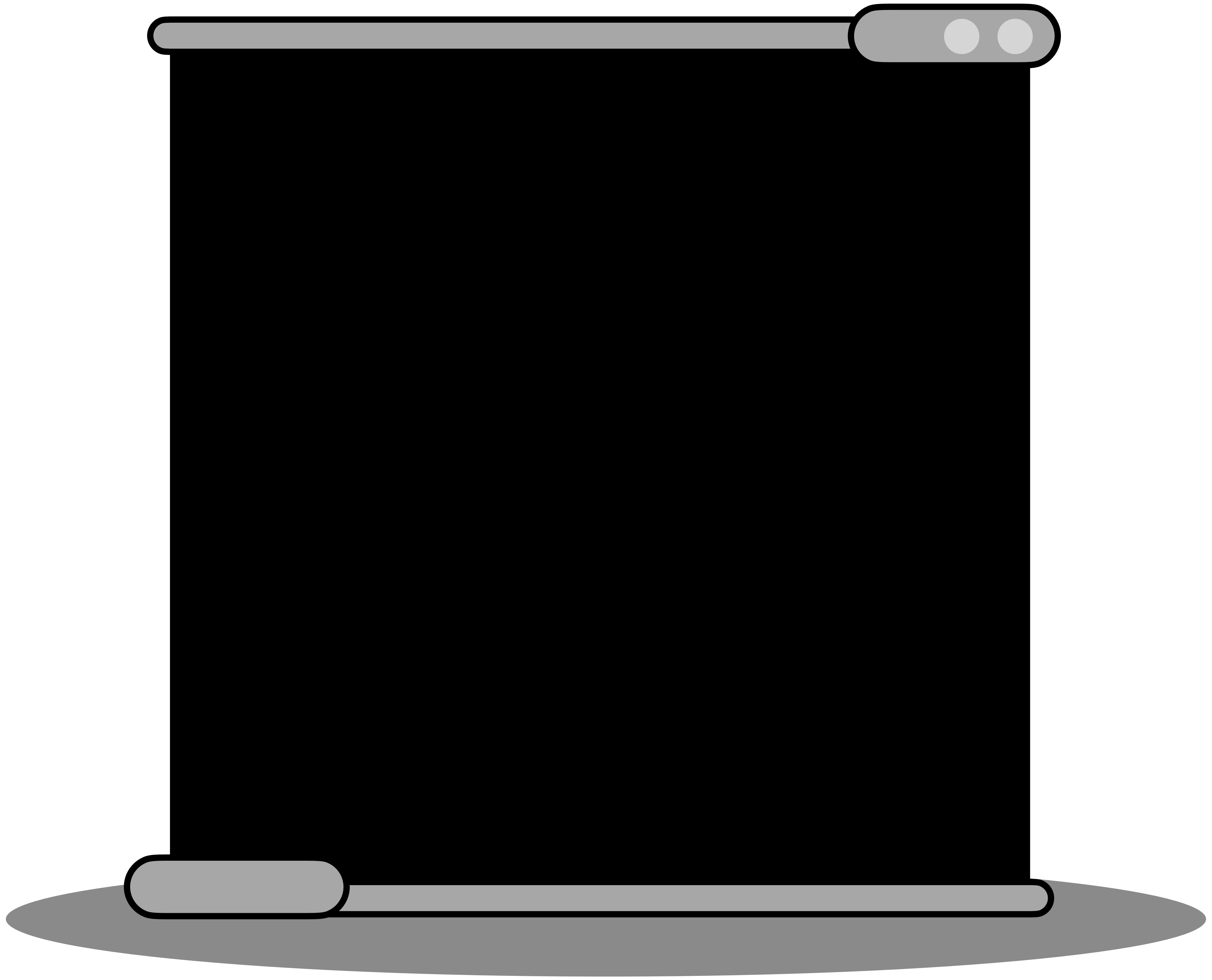
Attack Overview

Continue choosing benign features until the app is misclassified



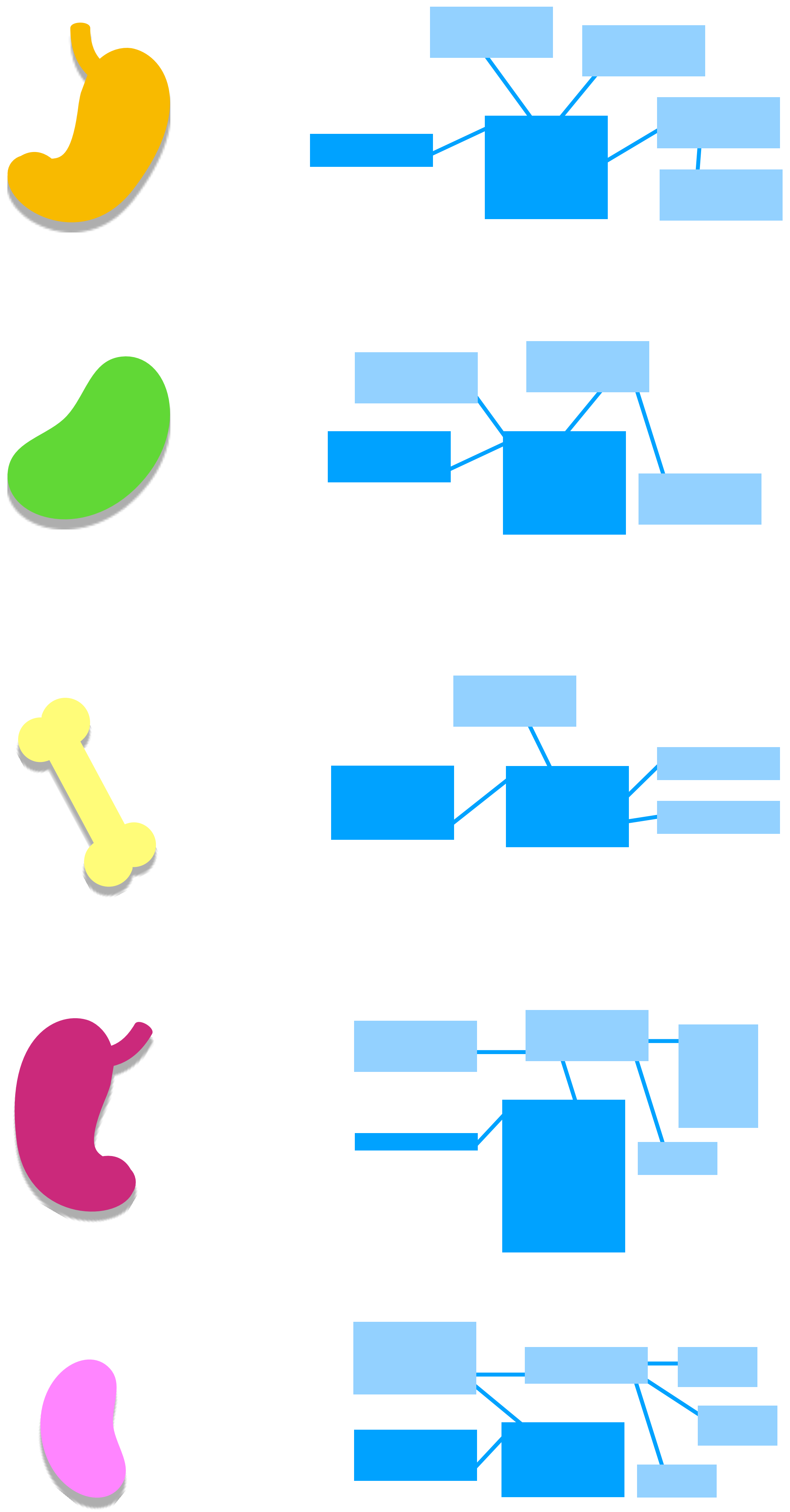


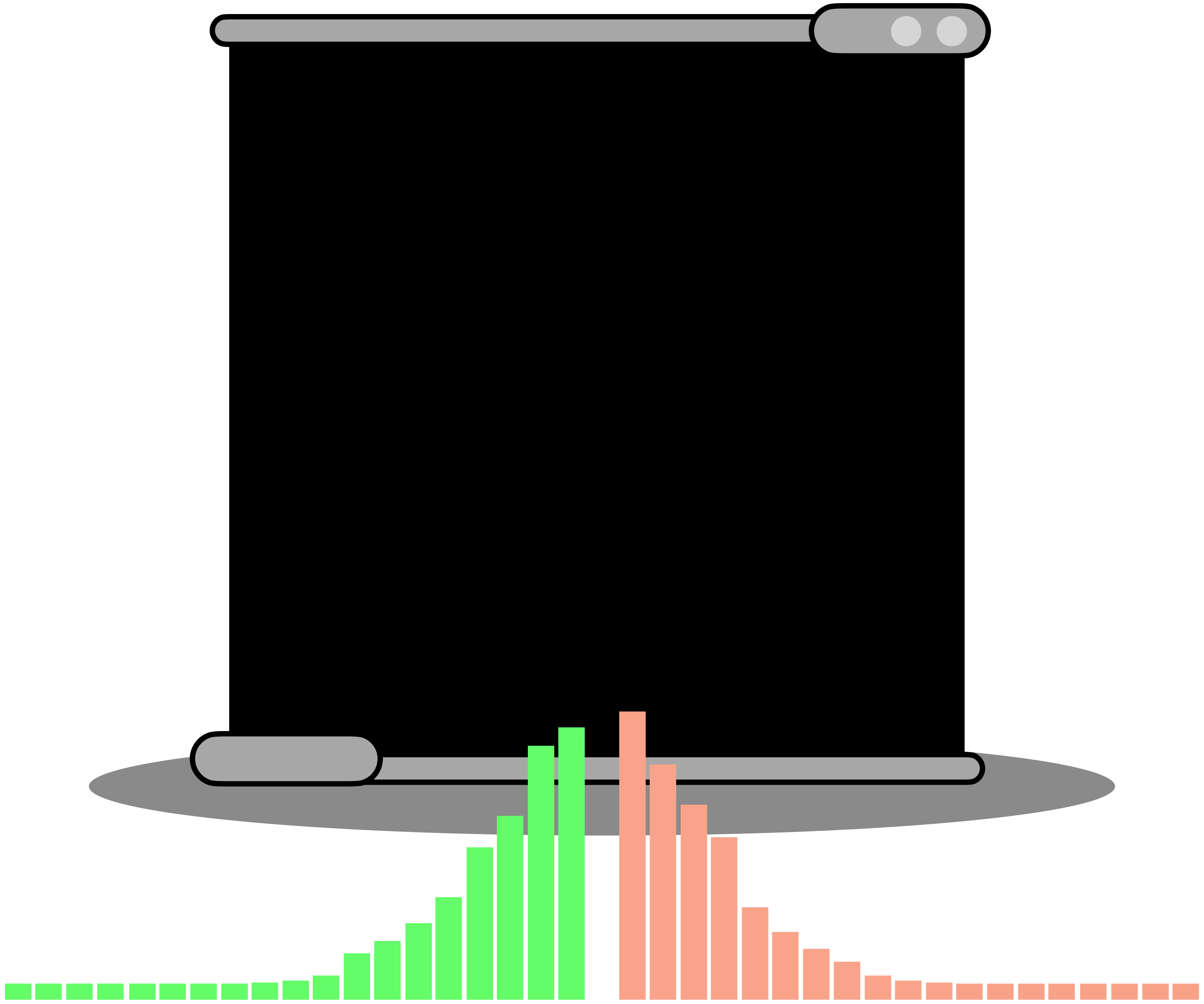
Side-Effects



Side-Effects

Each organ contains side-effect features.

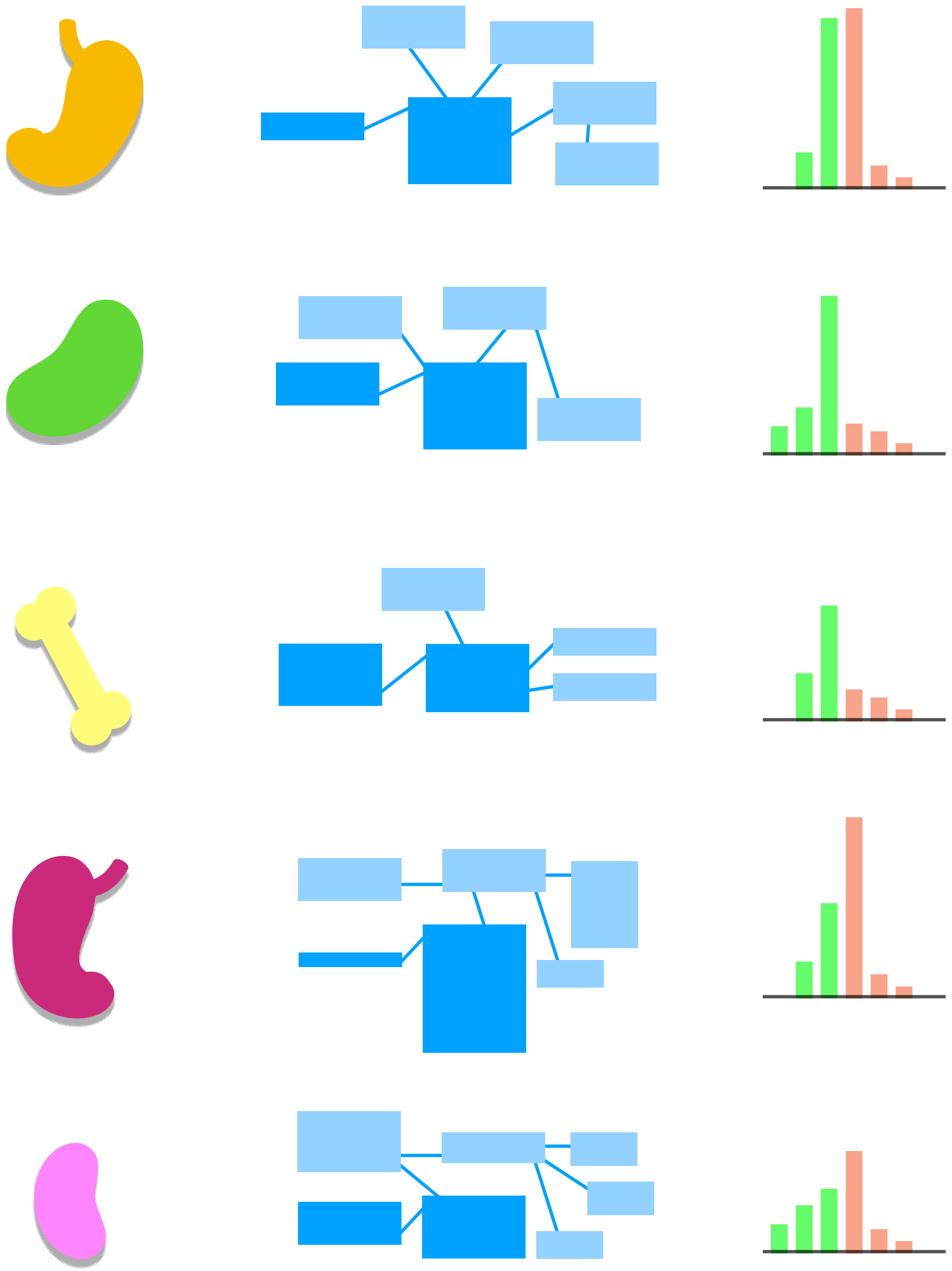


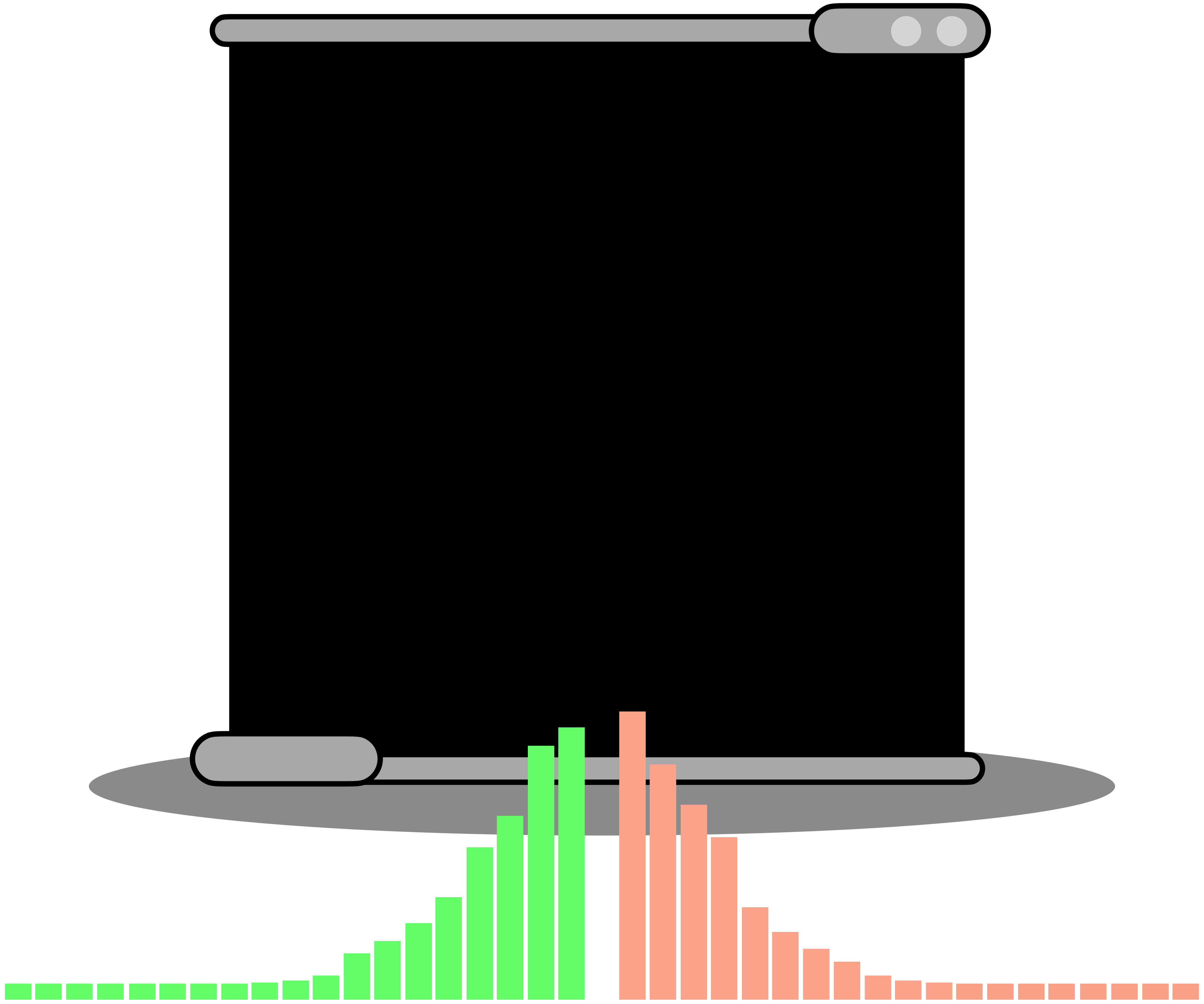


Side-Effects

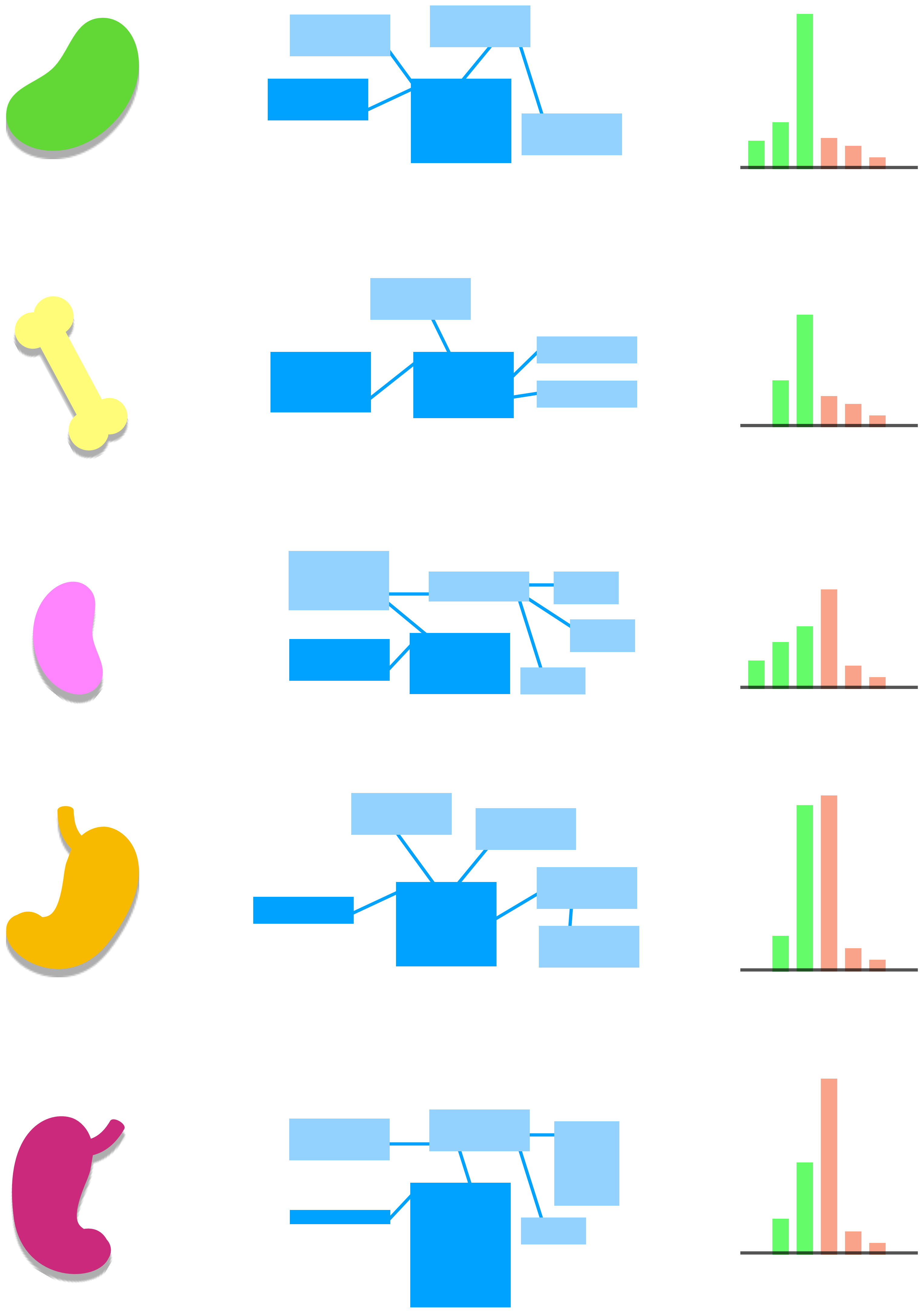
Each organ contains side-effect features.

We can sum target features, positive, and negative side effects





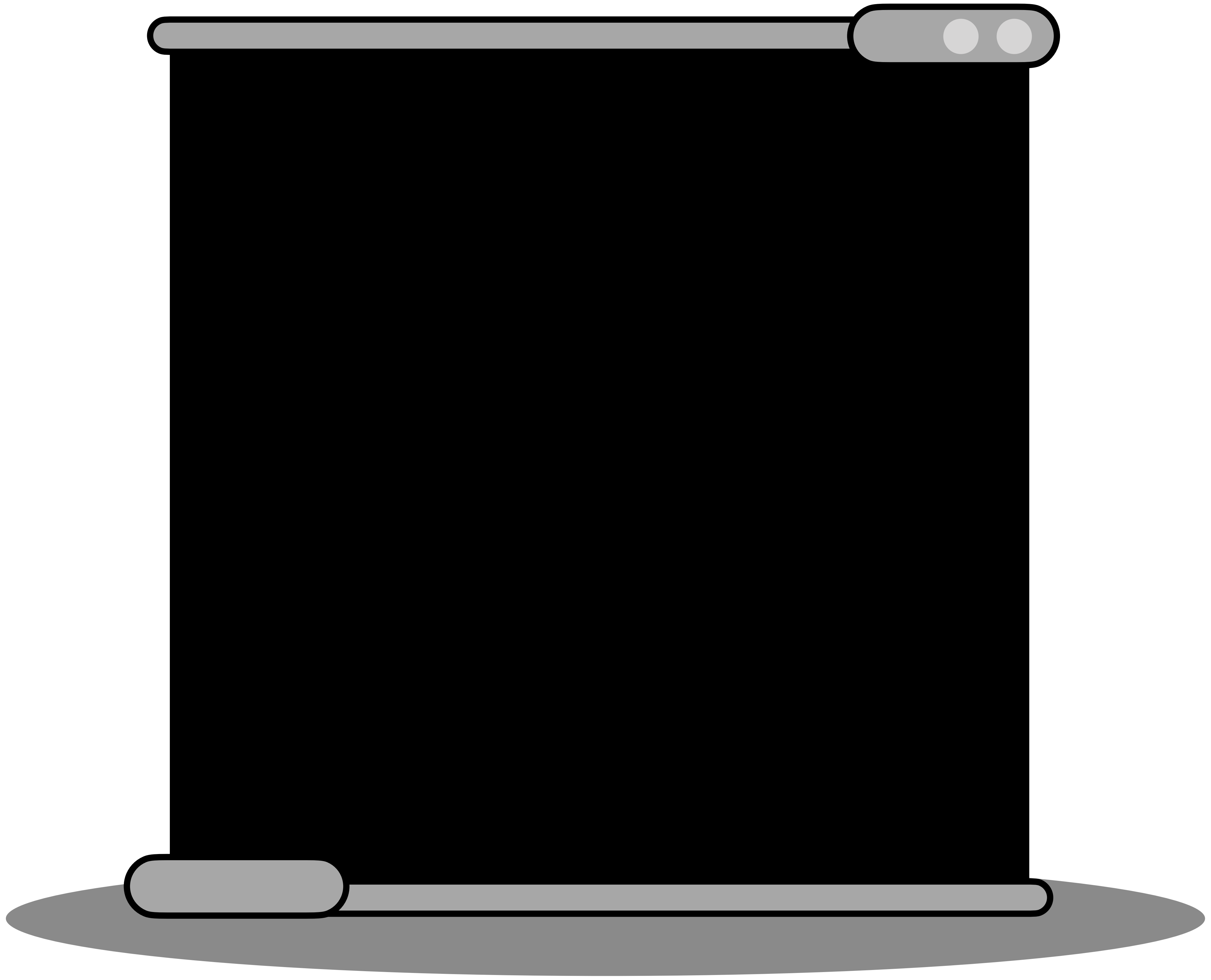
Side-Effects



Each organ contains side-effect features.

We can sum target features, positive, and negative side effects

to choose organs in order of their overall benign weight



Android Attack: Experiments

Experimental Testbed

SYSTEMS SECURITY RESEARCH LAB

KING'S College LONDON

Dataset

- ~170K Android apps (10% malware) from Jan 2017 to Dec 2018
- 66% training - 34% testing (random split, to remove "concept drift" as a variable)
  [Jordaney et al., "*TRANSCEND*", USENIX Sec 2017;
  Pendlebury et al., "*TESSERACT*", USENIX Sec 2019]

Experimental Testbed

SYSTEMS SECURITY RESEARCH LAB | KING'S College LONDON

Dataset

- ~170K Android apps (10% malware) from Jan 2017 to Dec 2018
- 66% training - 34% testing (random split, to remove "concept drift" as a variable) [Jordaney et al., "*TRANSCEND*", USENIX Sec 2017; Pendlebury et al., "*TESSERACT*", USENIX Sec 2019]

Detection algorithms

- DREBIN [NDSS'14]: Linear SVM, binary feature space
- Sec-SVM [TDSC'17]: Feature-space defense for DREBIN (uses "more evenly-distributed weights")

Experimental Testbed

Dataset

- ~170K Android apps (10% malware) from Jan 2017 to Dec 2018
- 66% training - 34% testing (random split, to remove "concept drift" as a variable)
  [Jordaney et al., "*TRANSCEND*", USENIX Sec 2017;
  Pendlebury et al., "*TESSERACT*", USENIX Sec 2019]

Detection algorithms

- DREBIN [NDSS'14]: Linear SVM, binary feature space
- Sec-SVM [TDSC'17]: Feature-space defense for DREBIN
  (uses "more evenly-distributed weights")

Attack Configurations

- Low Confidence (L): overcome decision boundary
- High Confidence (H): first quartile of benign distribution

Results

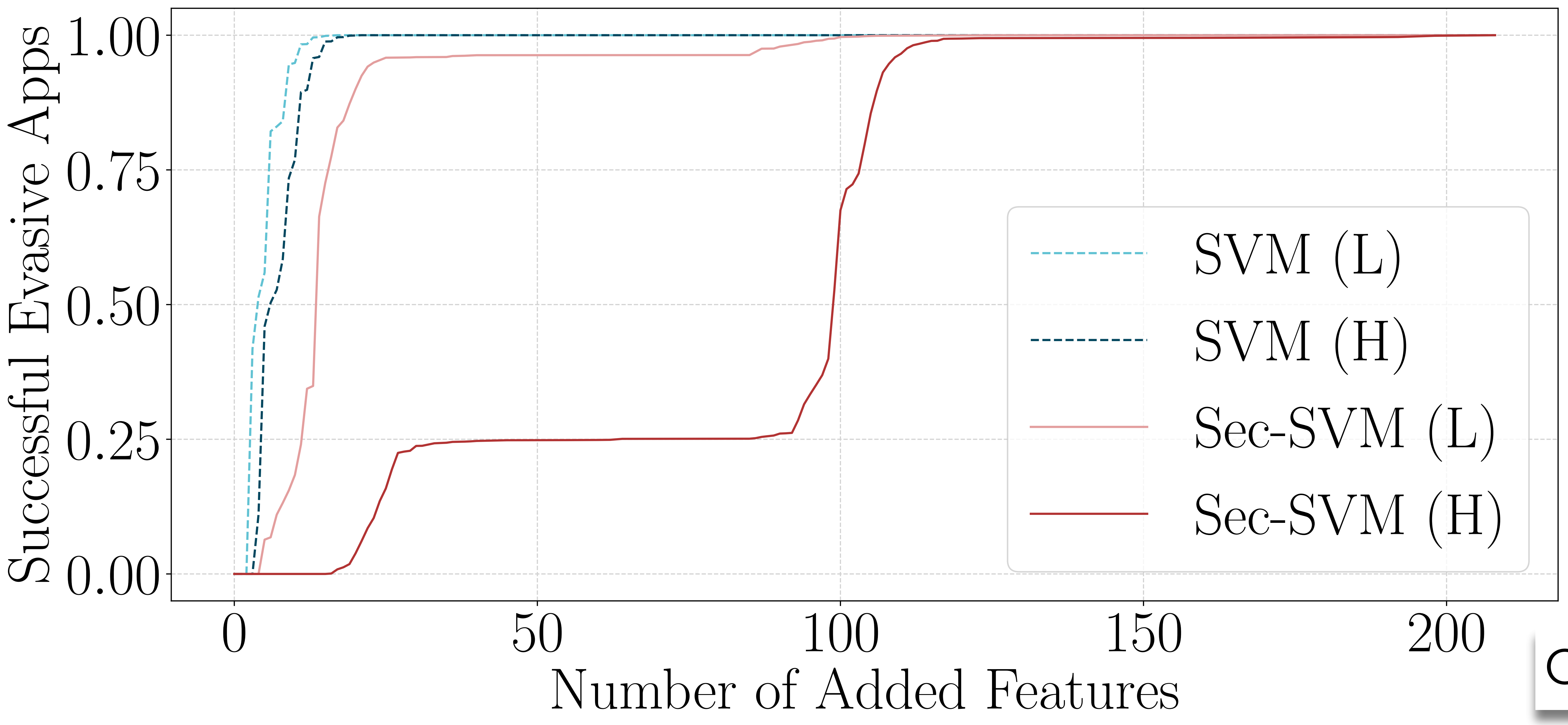
- **Attack Success Rate: 100%**
  - ~15K adversarial malware in total

Results

- **Attack Success Rate: 100%**
  - ~15K adversarial malware in total
- **Experimental Questions**
  - We do not limit the feature-space perturbation: what is the **feature-space impact**?
    - e.g., in images or audio there is a point in minimizing the perturbation
  - How much are the **app statistics** affected?
  - Is the attack practical? How much **time** does it take?

Results: What is the impact on Feature Space?

Results: What is the impact on Feature Space?



Out of 10K features

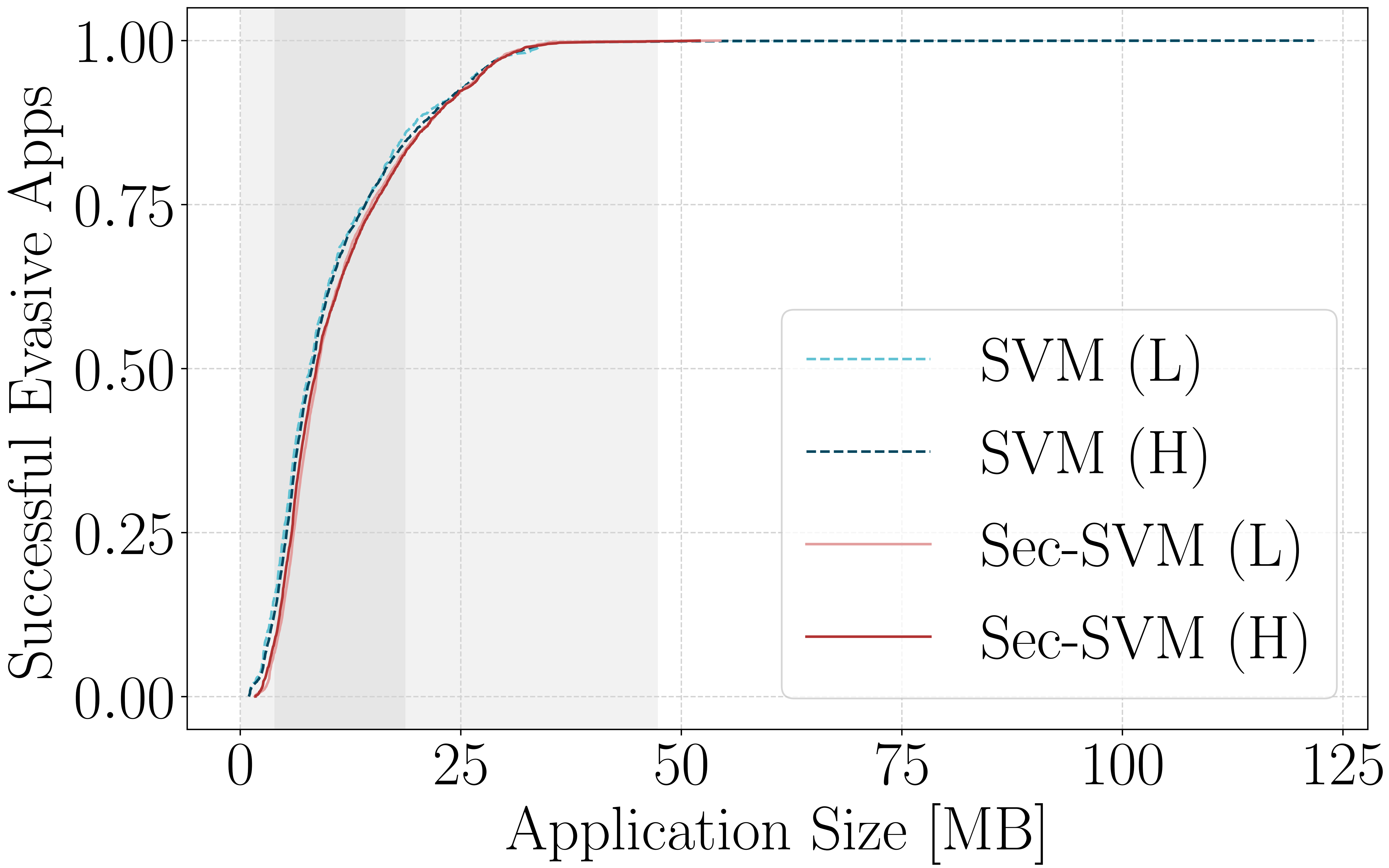
- Perturbations include side-effect features
- Sec-SVM (feature-space defense) forces the attacker to modify more features
  - Security-Performance trade-off
- **Next slides**: Does adding many features affect significantly app statistics?

Results: How much are app statistics affected?

- Adding all these features (+ side-effect features), what does it do to app statistics?
  - Limiting feature-space perturbations δ does not affect problem-space attack

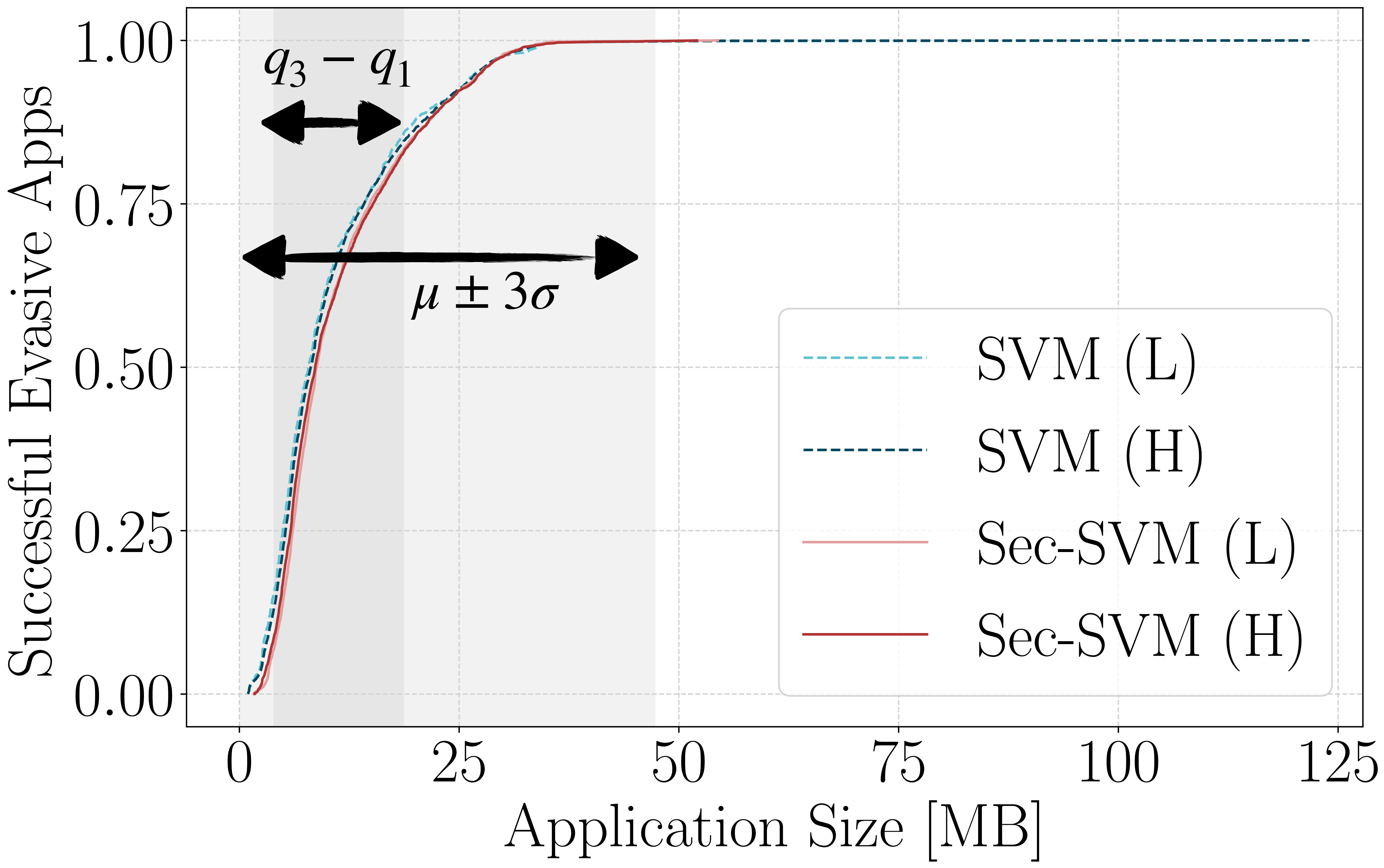
Results: How much are app statistics affected?

- Adding all these features (+ side-effect features), what does it do to app statistics?
  - Limiting feature-space perturbations δ does not affect problem-space attack



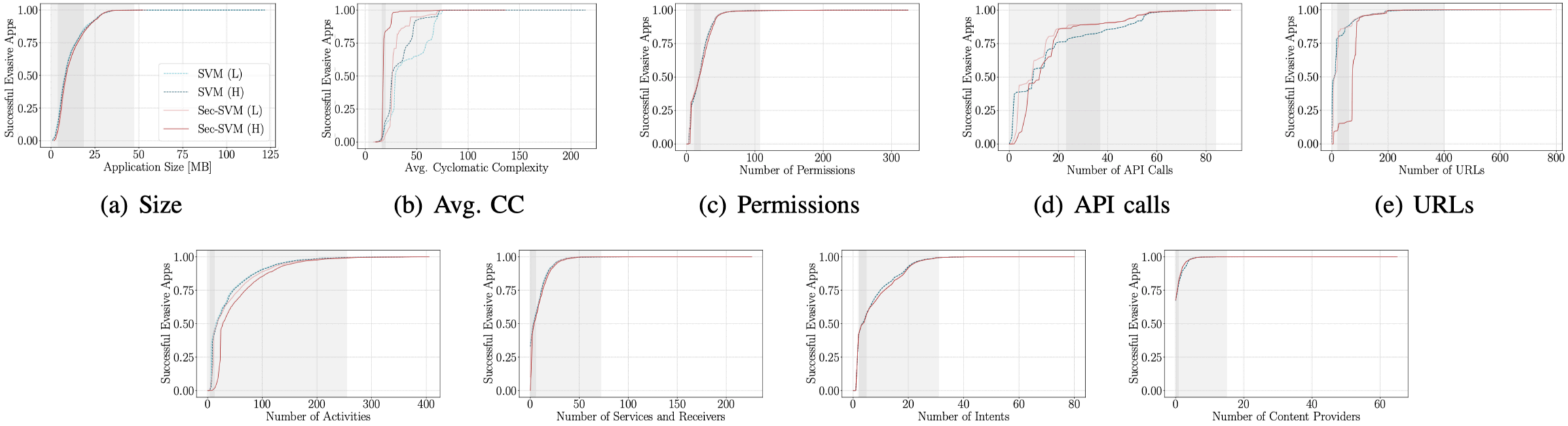
Results: How much are app statistics affected?

- Adding all these features (+ side-effect features), what does it do to app statistics?
  - Limiting feature-space perturbations δ does not affect problem-space attack



Results: How much are app statistics affected?

- Adding all these features (+ side-effect features), what does it do to app statistics?
  - Limiting feature-space perturbations δ does not affect problem-space attack

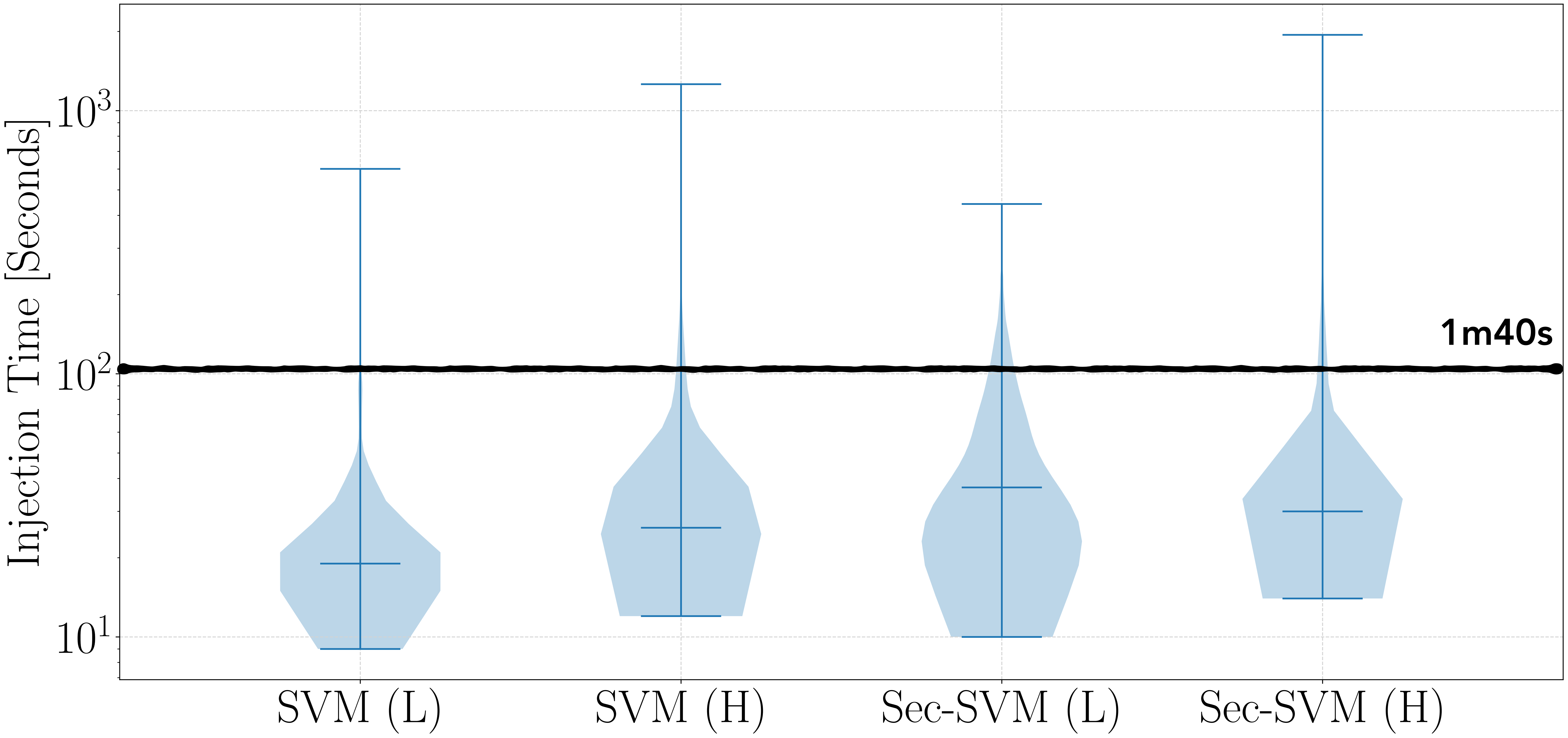


(a) Size (b) Avg. CC (c) Permissions (d) API calls (e) URLs

(f) Activities (g) Services and Receivers (h) Intents (i) Content Providers

Results: How much time does an attack take?

- In most cases, **less than 2 minutes** to create an adversarial example



Conclusions

SYSTEMS SECURITY RESEARCH LAB

KING'S College LONDON

Problem-Space Adversarial ML Attacks

- Novel reformulation of adversarial attacks
- Novel end-to-end adversarial malware generation

Project website (with code):

- https://s2lab.kcl.ac.uk/projects/intriguing/

Problem-Space Constraints

- Available Transformations
- Preserved Semantics
- Plausibility
- Robustness to Preprocessing

Search Strategies

- Gradient-driven
- Problem-driven
- Hybrid

Fabio Pierazzi*, Feargus Pendlebury*, Jacopo Cortellazzi, Lorenzo Cavallaro, "**Intriguing Properties of Adversarial ML Attacks in the Problem Space**", *IEEE Symposium on Security and Privacy (S&P)*, 2020

Attack Code: Publicly available

- **Project website: https://s2lab.kcl.ac.uk/projects/intriguing/**
  - Attack Code and Dataset (released May 1, 2020)
  - Available to Researchers under MIT license

SYSTEMS SECURITY RESEARCH LAB

KING'S College LONDON

Problem-Space Adversarial ML Attacks

- Novel reformulation of adversarial attacks
- Novel end-to-end adversarial malware generation

Project website (with code):

- https://s2lab.kcl.ac.uk/projects/intriguing/

Problem-space attacks research is just beginning!

Problem-Space Constraints

- Available Transformations
- Preserved Semantics
- Plausibility
- Robustness to Preprocessing

Search Strategies

- Gradient-driven
- Problem-driven
- Hybrid

Fabio Pierazzi*, Feargus Pendlebury*, Jacopo Cortellazzi, Lorenzo Cavallaro, "**Intriguing Properties of Adversarial ML Attacks in the Problem Space**", *IEEE Symposium on Security and Privacy (S&P)*, 2020