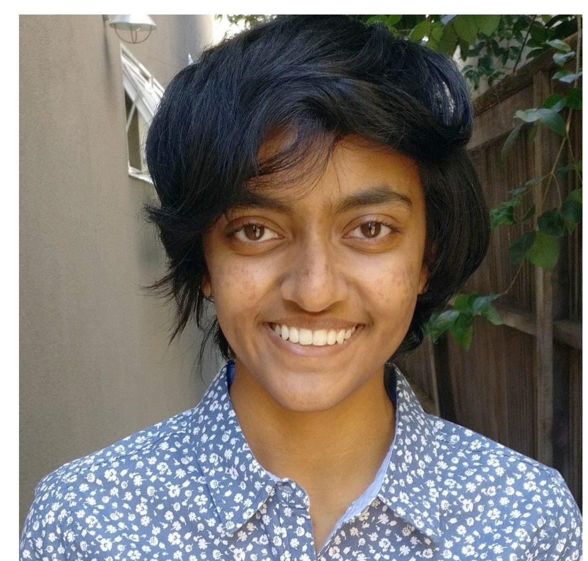
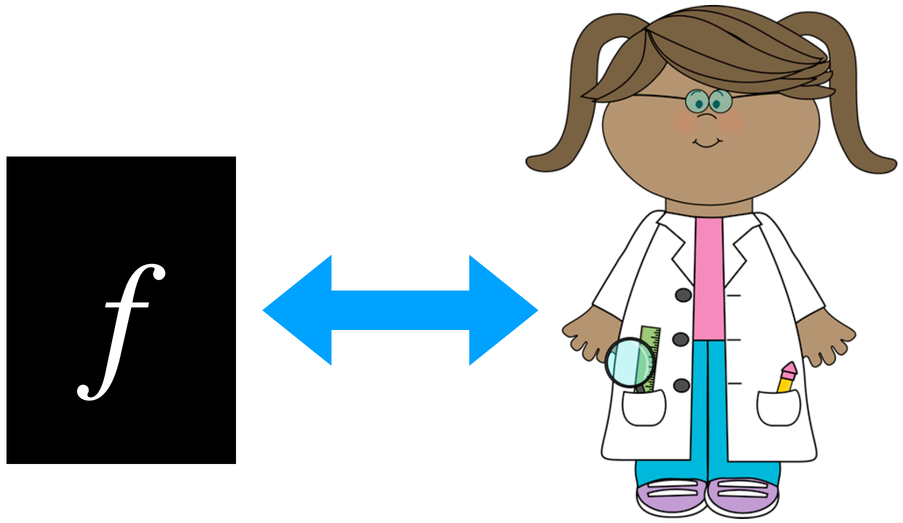
Locally Covert Learning

Justin Holmgren
NTT Research

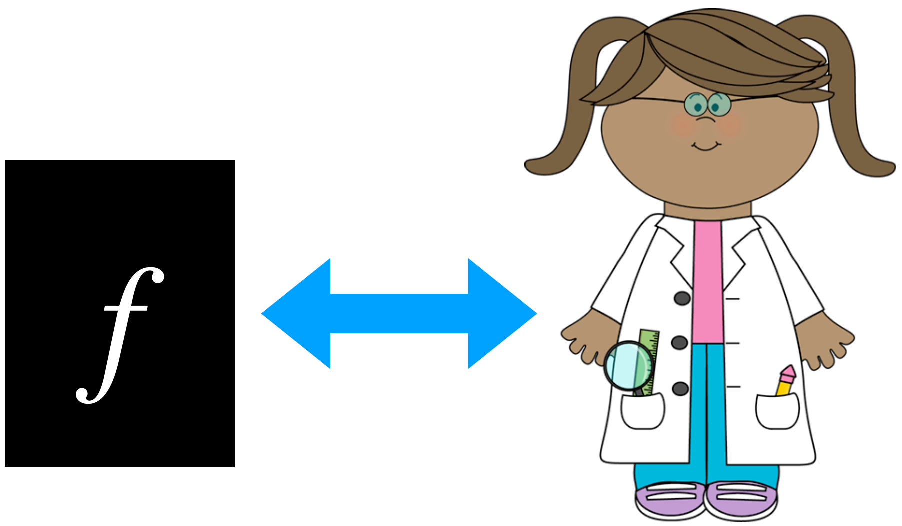
Ruta Jawale
UIUC



Learning Boolean Functions

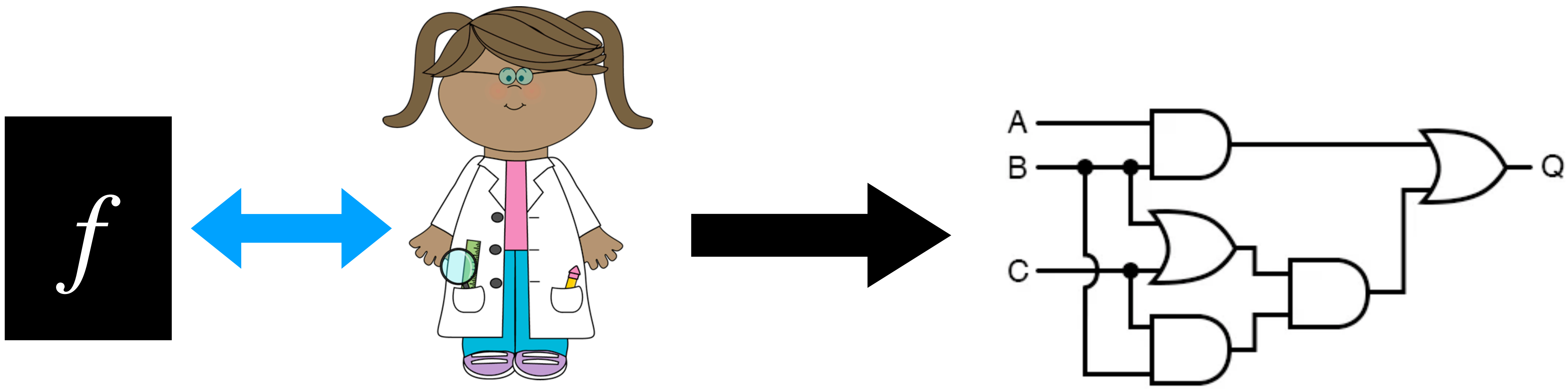


Learning Boolean Functions



1. Polynomial-time learner gets *access to a function* $f : \{0,1\}^n \rightarrow \{0,1\}$

Learning Boolean Functions



1. Polynomial-time learner gets *access to a function* $f : \{0,1\}^n \to \{0,1\}$

2. Learner's goal is to output a function h that agrees with f on most inputs.

The Learning Model, Part I

The Learning Model, Part I

By "$\mathscr{H}$ is learnable", we mean there exists a learning algorithm that is:

The Learning Model, Part I

By "$\mathscr{H}$ is learnable", we mean there exists a learning algorithm that is:

Efficient:
Given accuracy parameter α and confidence parameter δ, learner runs in time poly$(n,\ 1/\alpha,\ \log(1/\delta))$.

The Learning Model, Part I

By “$\mathscr{H}$ is learnable”, we mean there exists a learning algorithm that is:

Efficient:
Given accuracy parameter α and confidence parameter δ, learner runs in time poly$(n,\ 1/\alpha,\ \log(1/\delta))$.

(Weakly) Agnostic:
If target function f is ϵ-close to some $h^\star \in \mathscr{H}$,

The Learning Model, Part I

By "$\mathscr{H}$ is learnable", we mean there exists a learning algorithm that is:

Efficient:
Given accuracy parameter α and confidence parameter δ, learner runs in time poly$(n,\ 1/\alpha,\ \log(1/\delta))$.

(Weakly) Agnostic:
If target function f is ϵ-close to some $h^\star \in \mathscr{H}$,
the learner outputs a hypothesis h that is $(O(\epsilon) + \alpha)$-close to f,

The Learning Model, Part I

By “$\mathscr{H}$ is learnable”, we mean there exists a learning algorithm that is:

Efficient:
Given accuracy parameter α and confidence parameter δ, learner runs in time poly$(n,\ 1/\alpha,\ \log(1/\delta))$.

(Weakly) Agnostic:
If target function f is ϵ-close to some $h^\star \in \mathscr{H}$,
the learner outputs a hypothesis h that is $(O(\epsilon) + \alpha)$-close to f,
with all but δ probability.

The Learning Model, Part I

By "$\mathscr{H}$ is learnable", we mean there exists a learning algorithm that is:

Efficient:
Given accuracy parameter α and confidence parameter δ, learner runs in time poly$(n,\ 1/\alpha,\ \log(1/\delta))$.

(Weakly) Agnostic:
If target function f is ϵ-close to some $h^\star \in \mathscr{H}$,
the learner outputs a hypothesis h that is $(O(\epsilon) + \alpha)$-close to f,
with all but δ probability.

Distribution-Specific: "closeness" is measured wrt the uniform distribution

The Learning Model, Part I

By "$\mathscr{H}$ is learnable", we mean there exists a learning algorithm that is:

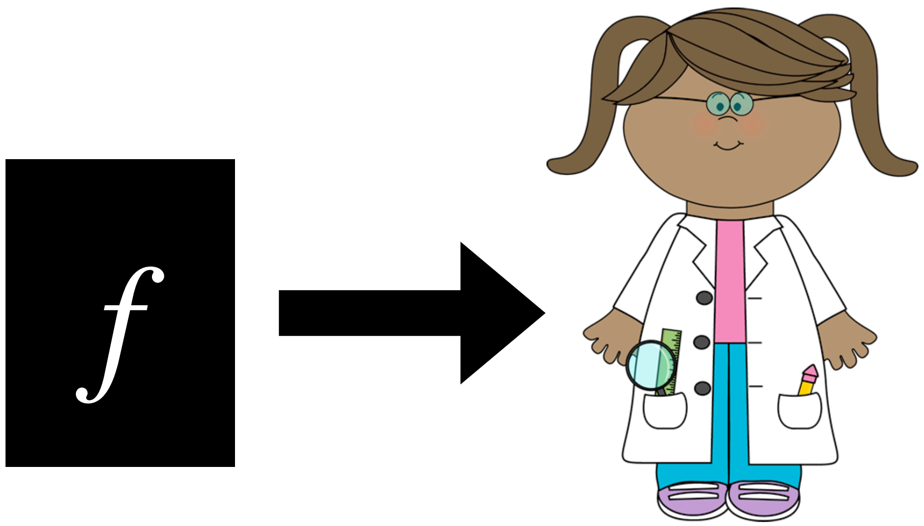
Efficient:
Given accuracy parameter α and confidence parameter δ, learner runs in time poly$(n,\ 1/\alpha,\ \log(1/\delta))$.

(Weakly) Agnostic:
If target function f is ϵ-close to some $h^\star \in \mathscr{H}$,
the learner outputs a hypothesis h that is $(O(\epsilon) + \alpha)$-close to f,
with all but δ probability.

Distribution-Specific: "closeness" is measured wrt the uniform distribution

Improper: Learner can output any circuit, not necessarily in $\mathscr{H}$.

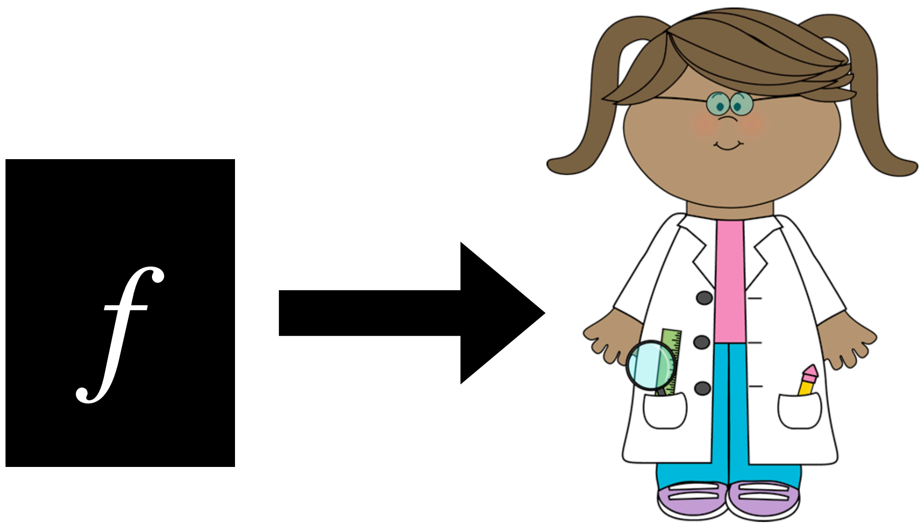
The Learning Model, Part II
Types of Function Access

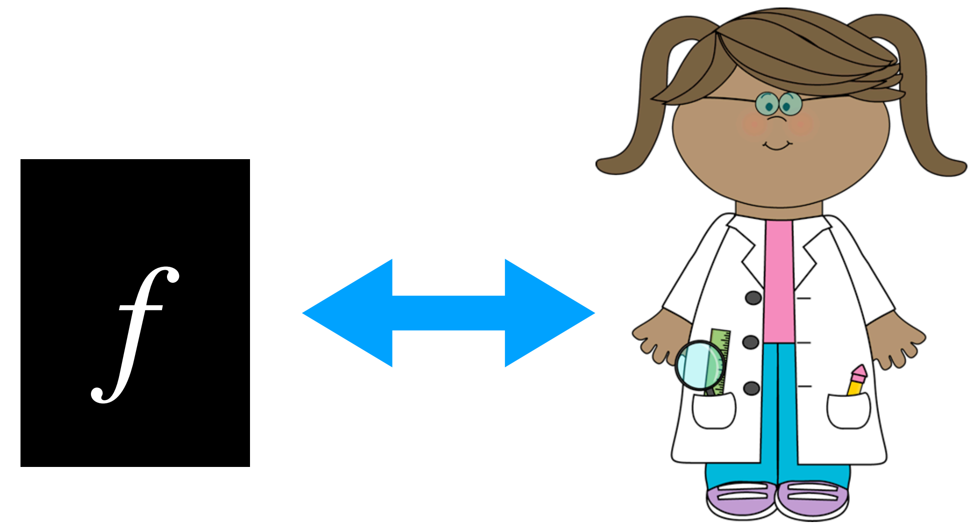


Passive Learning:

Learner gets pairs $(x, f(x))$
for uniformly random x

The Learning Model, Part II
Types of Function Access





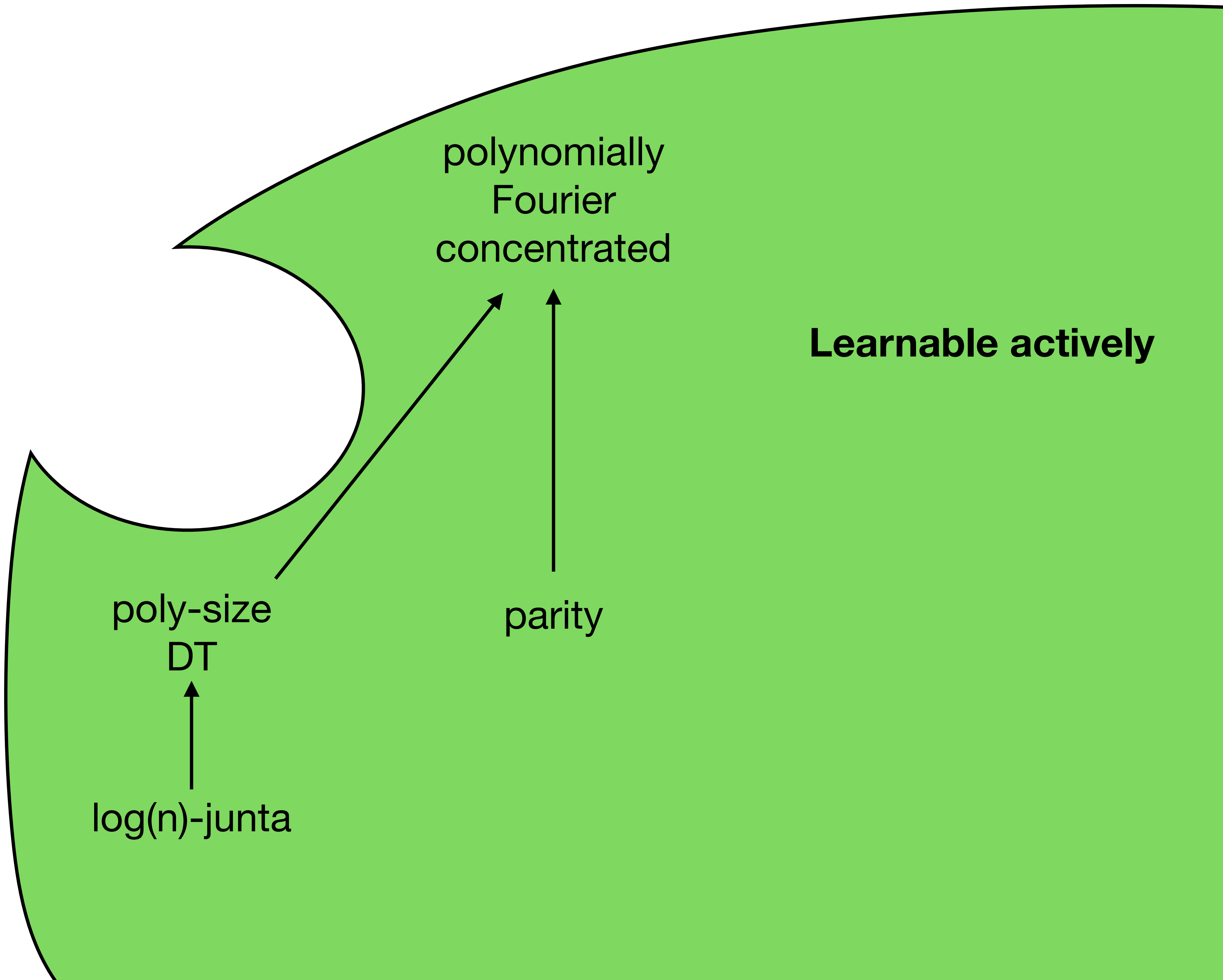
Passive Learning:

Learner gets pairs $(x, f(x))$ for uniformly random x

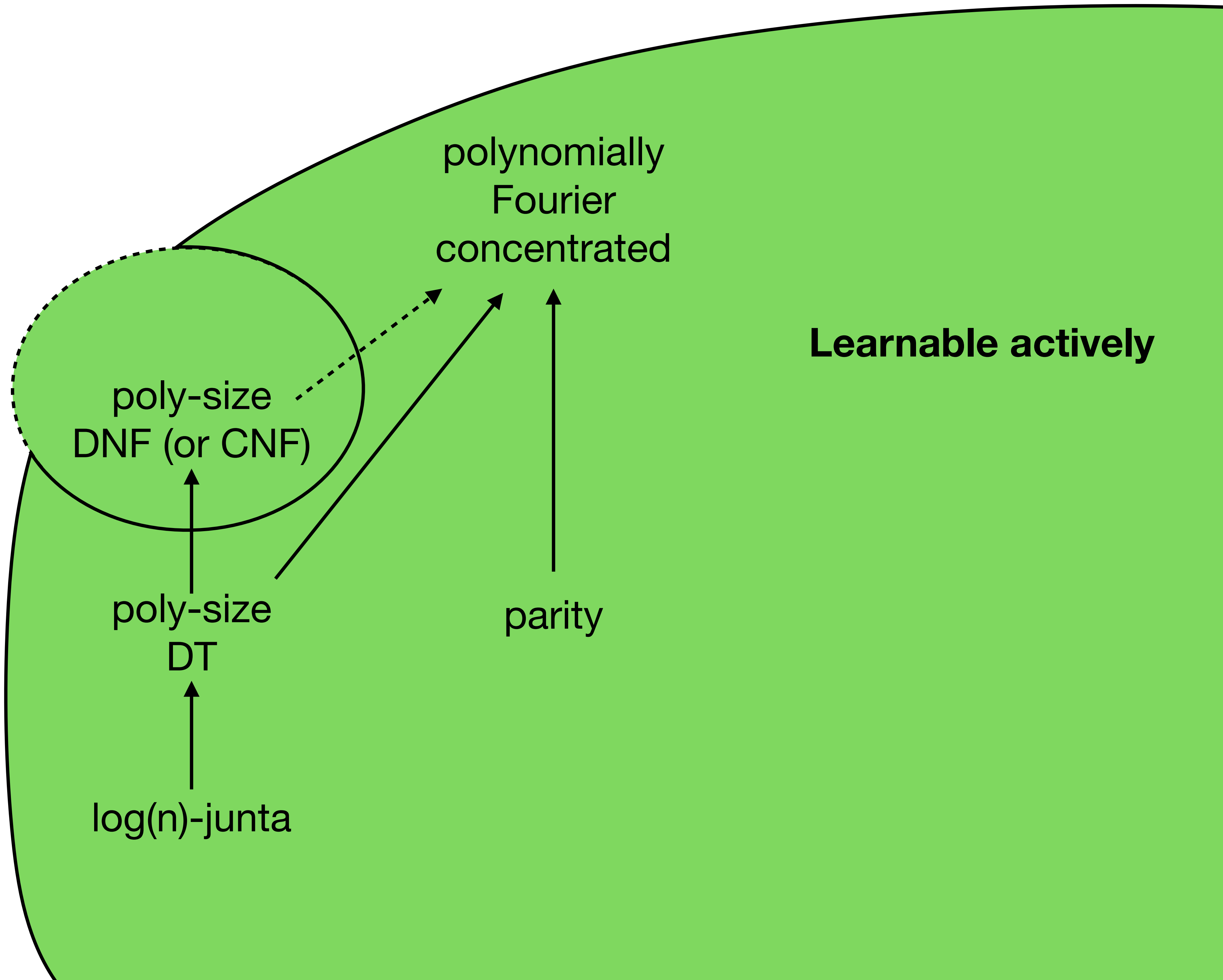
Active Learning:

Learner gets oracle access to f.

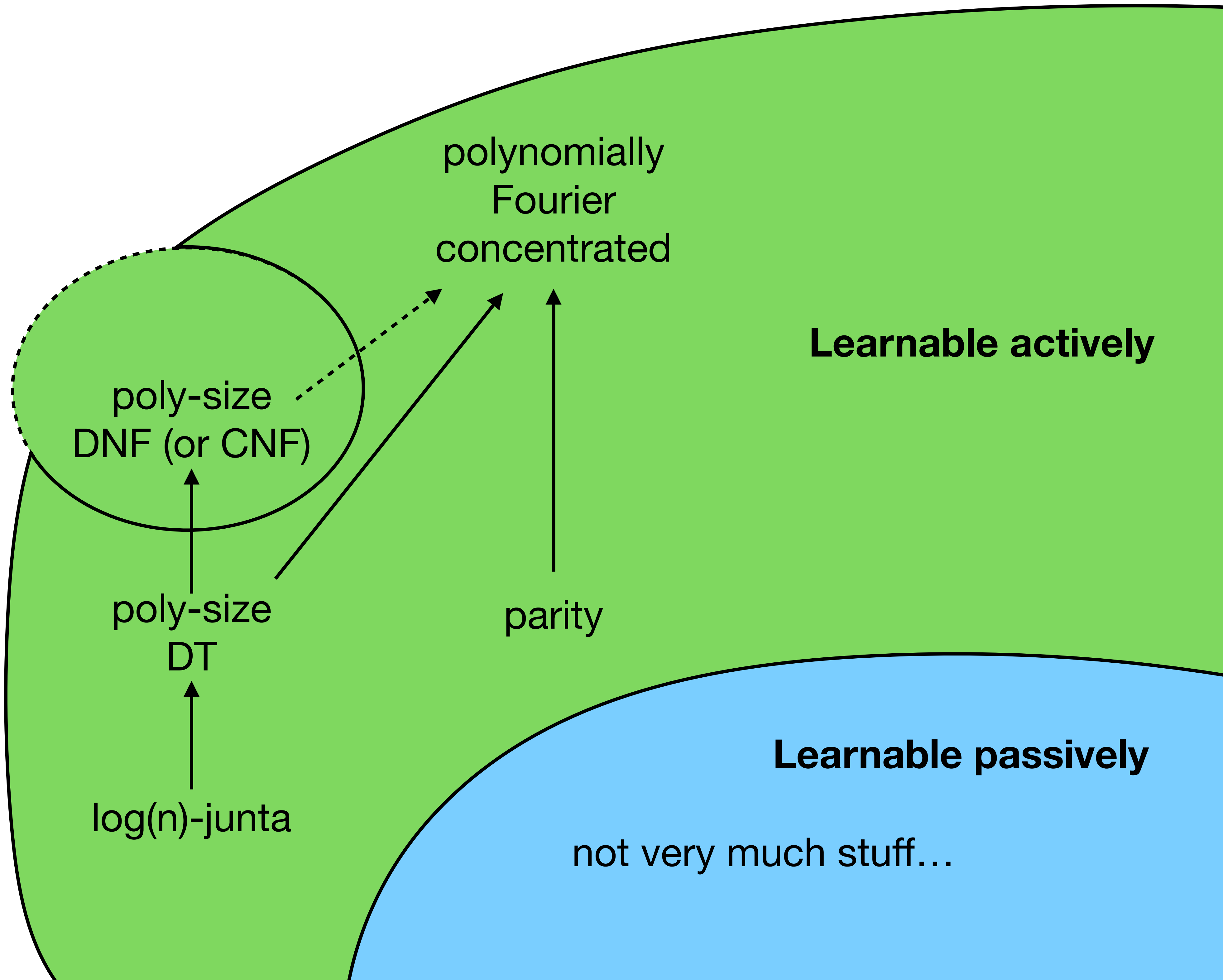
Passive vs. Active Learning



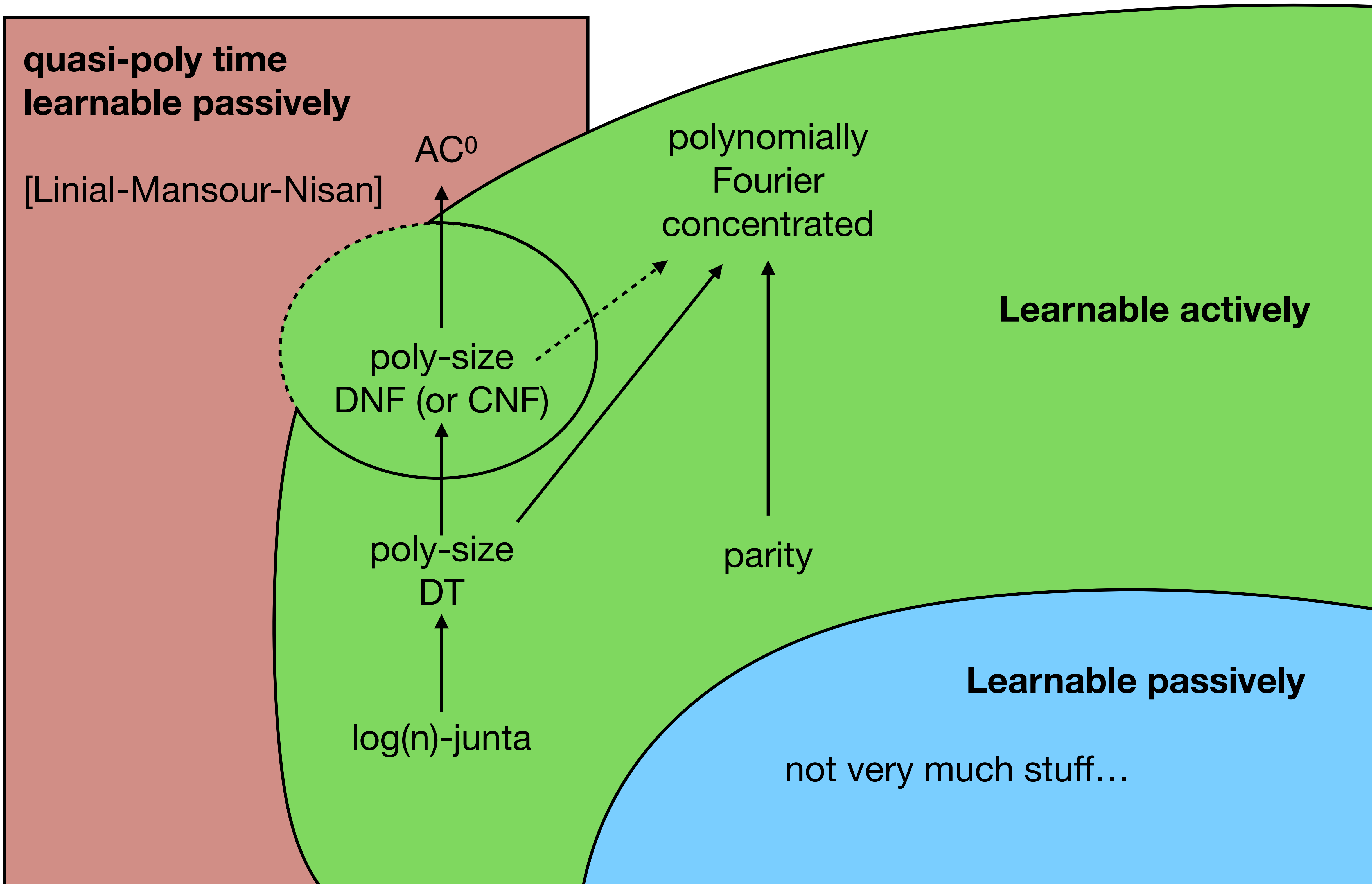
Passive vs. Active Learning



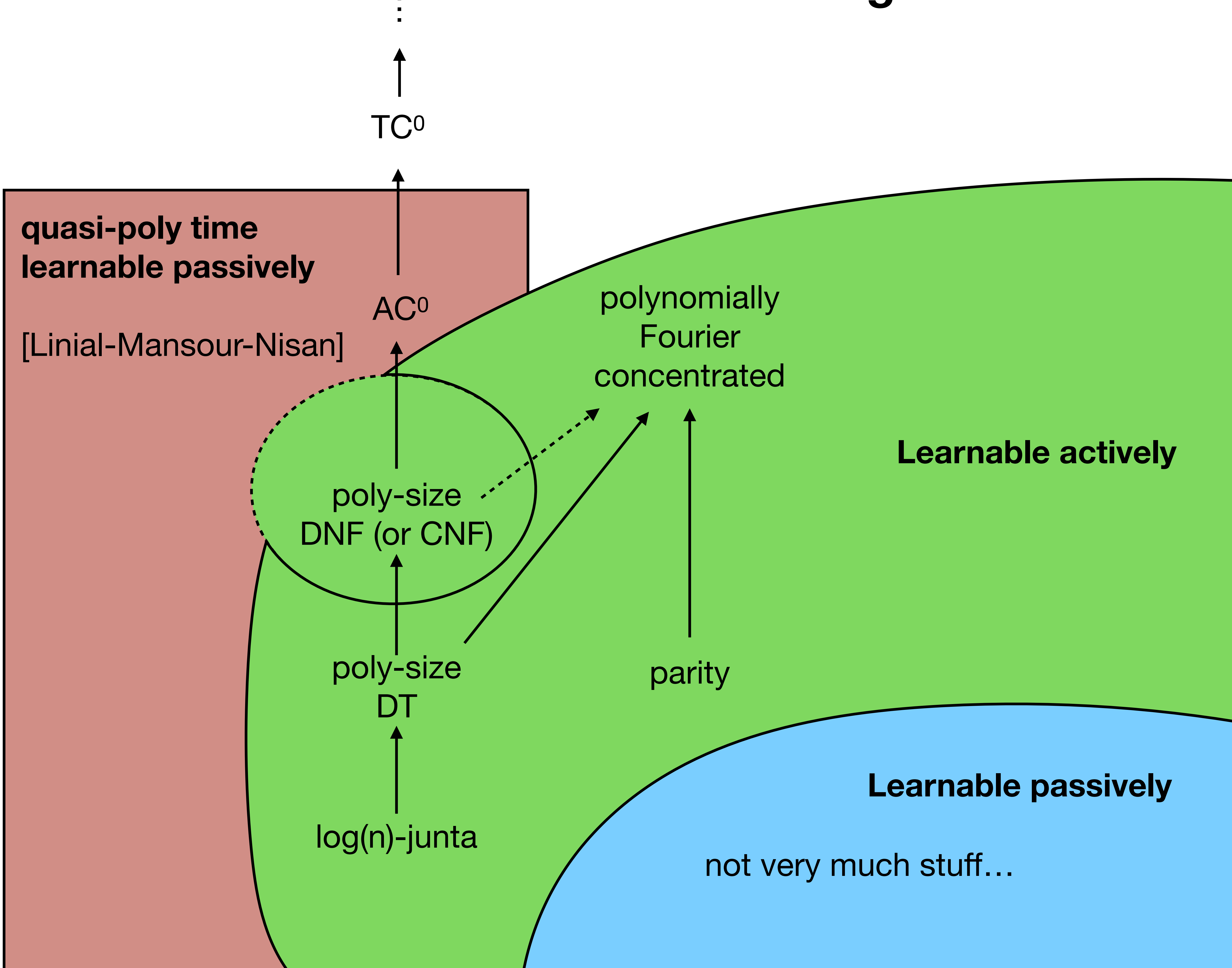
Passive vs. Active Learning



Passive vs. Active Learning



Passive vs. Active Learning



where’s the adversary?

Adversarial Learning

Adversarial Learning

By now a burgeoning field. Includes, but not limited to:

Adversarial Learning

By now a burgeoning field. Includes, but not limited to:

1. **Covert Learning**: [Canetti-Karchmer ’21, IKOS ’19]
   curious eavesdropper tries to piggyback on the queries of an active learner.

Adversarial Learning

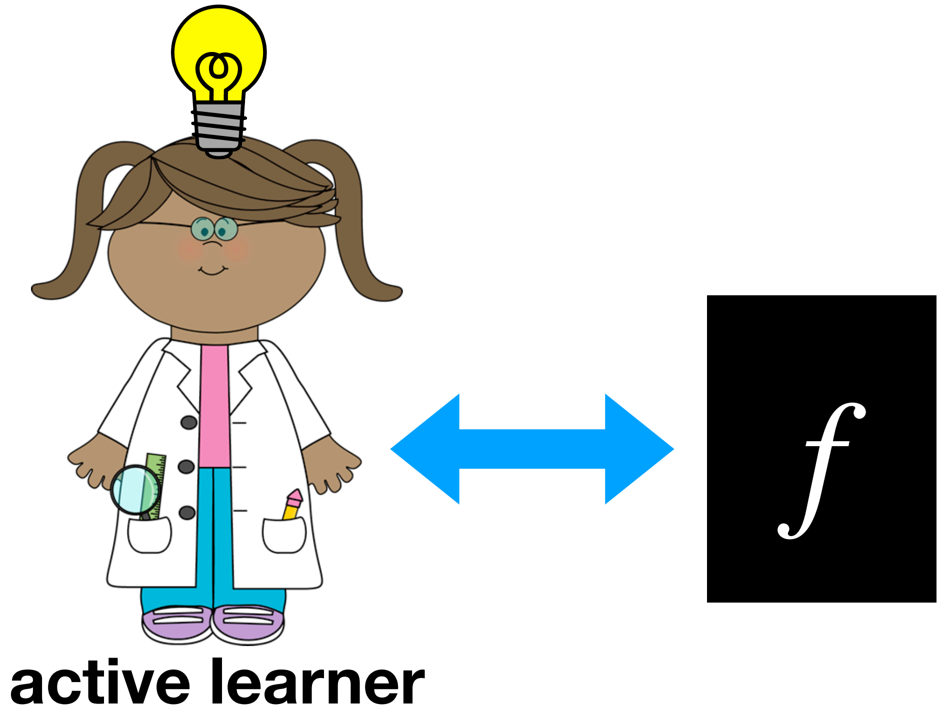
By now a burgeoning field. Includes, but not limited to:

1. **Covert Learning**: [Canetti-Karchmer '21, IKOS '19]
   curious eavesdropper tries to piggyback on the queries of an active learner.

2. **Verifiable Learning:** [Goldwasser-Rothblum-Shafer-Yehudayoff '20]
   untrusted prover claims that a hypothesis h approximates f near-optimally (compared to some class of functions).

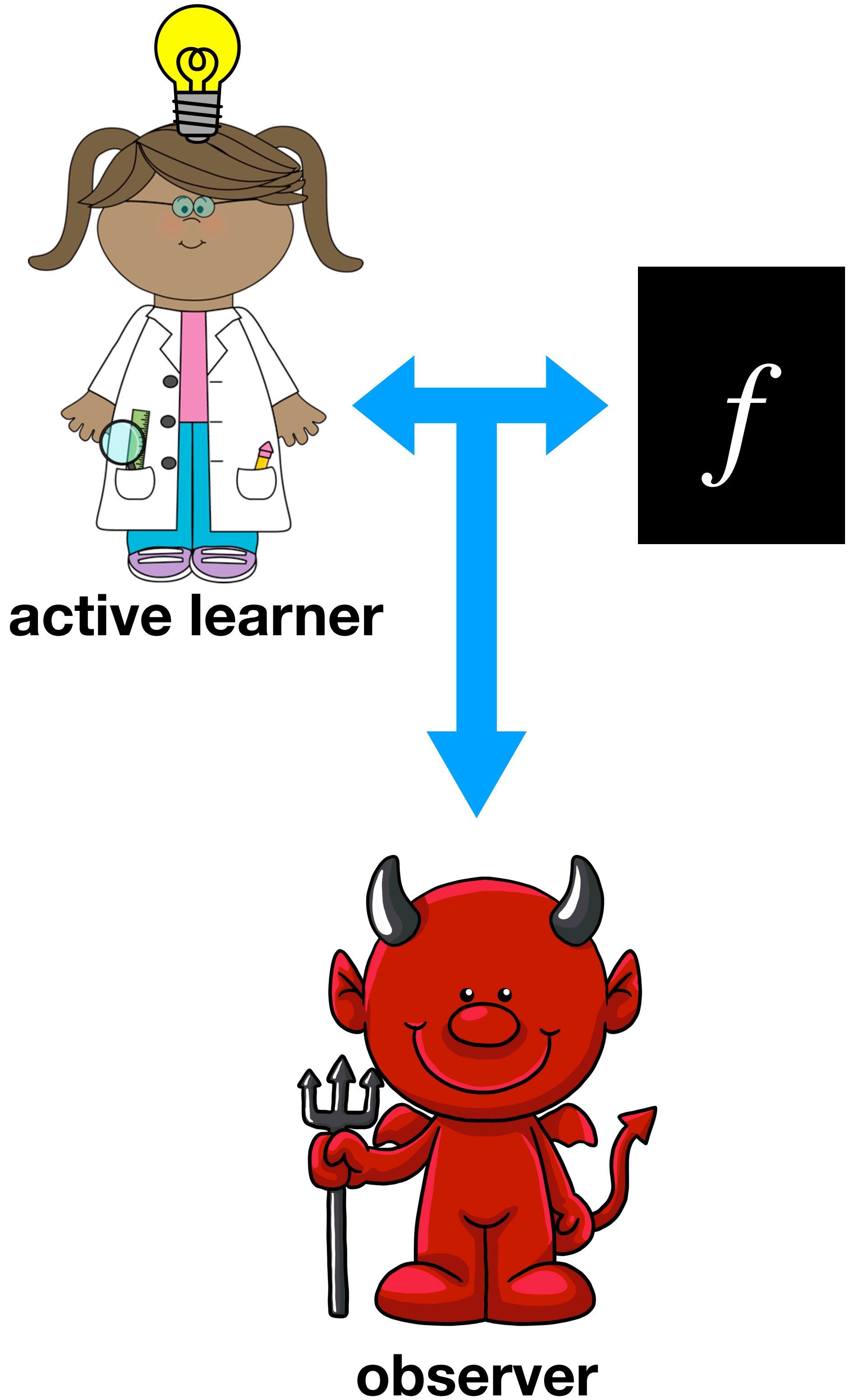
Covert Learning

[Canetti-Karchmer '21, IKOS '19]



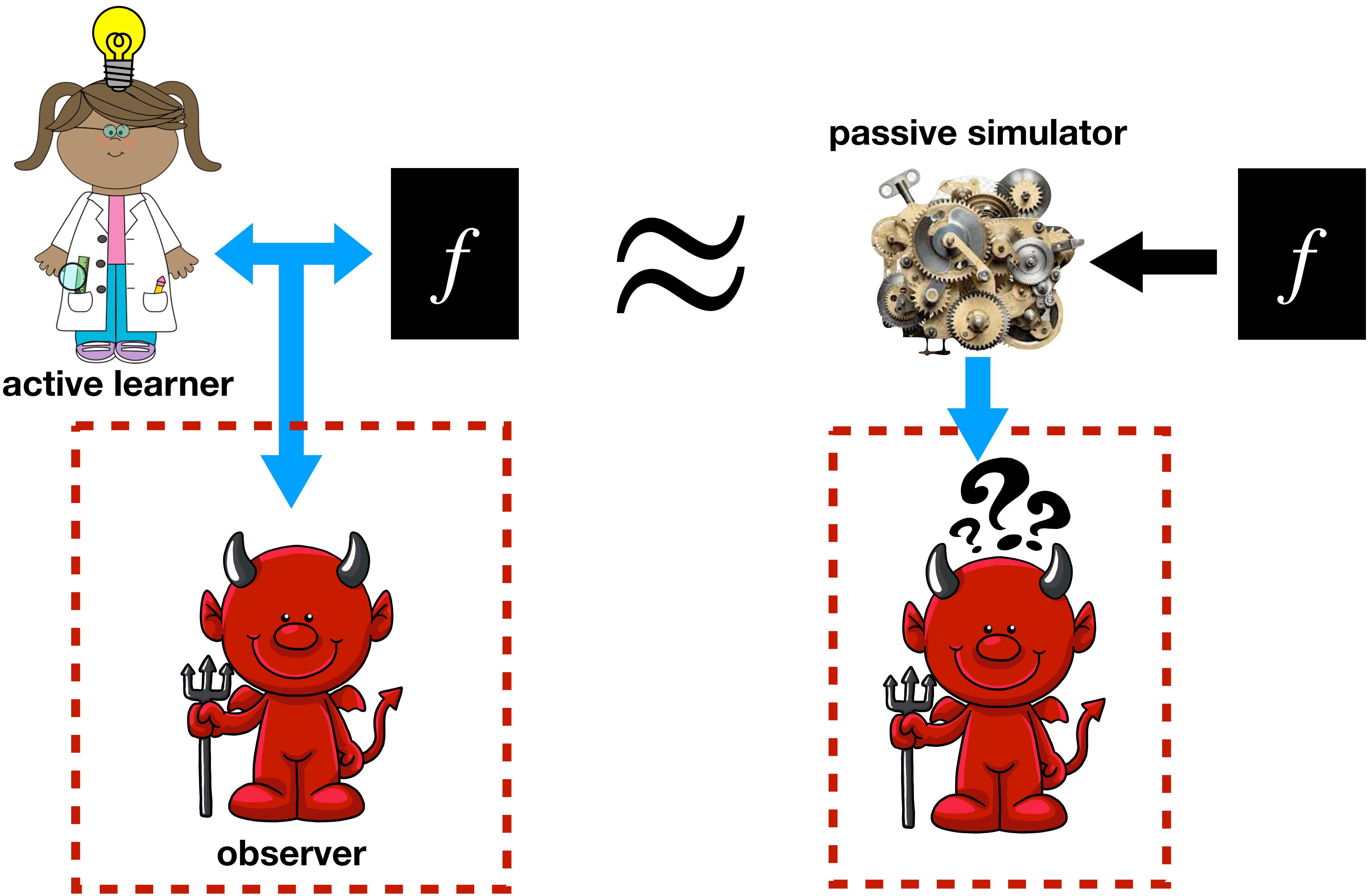
Covert Learning

[Canetti-Karchmer '21, IKOS '19]



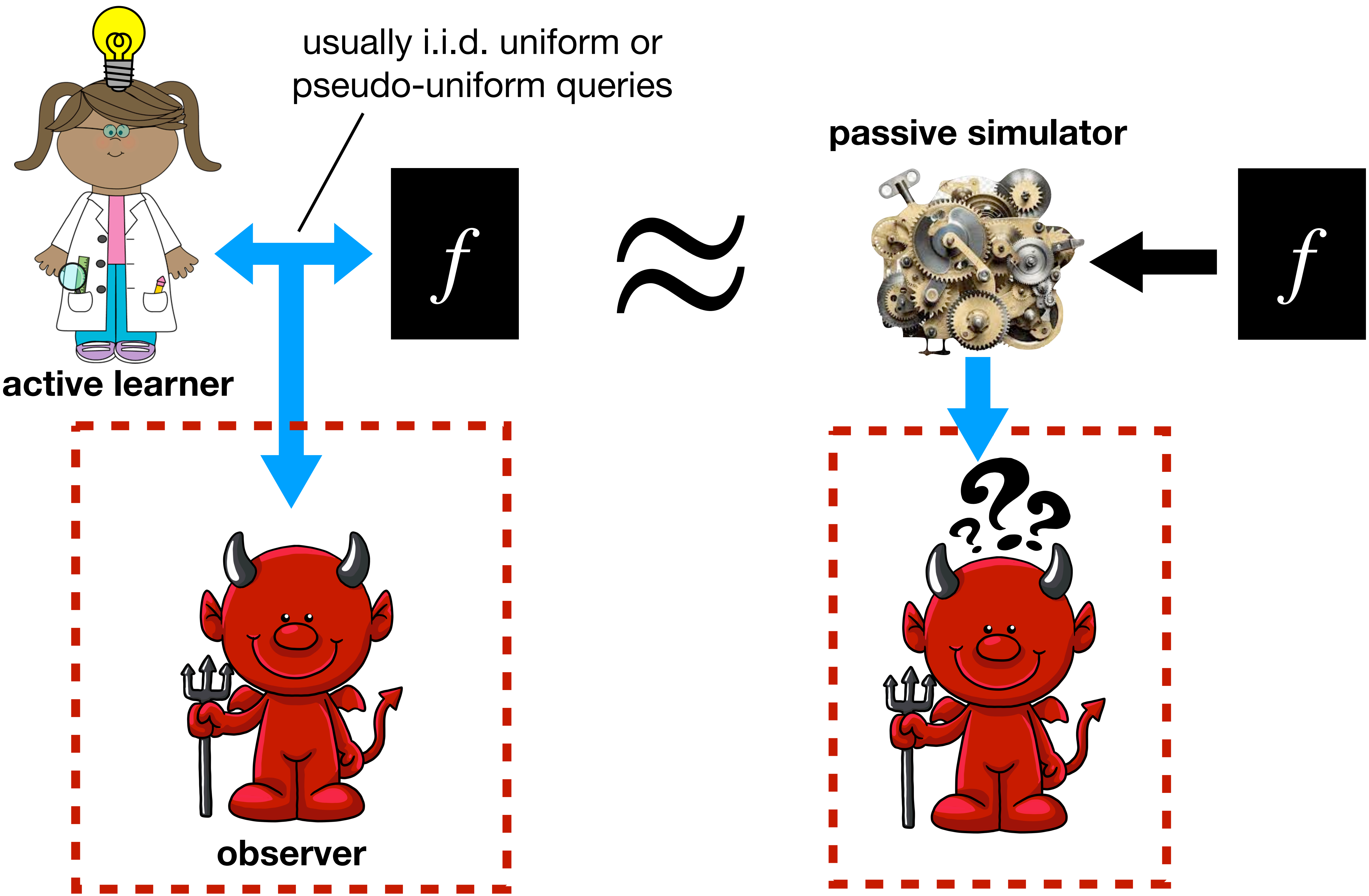
Covert Learning

[Canetti-Karchmer ’21, IKOS ’19]



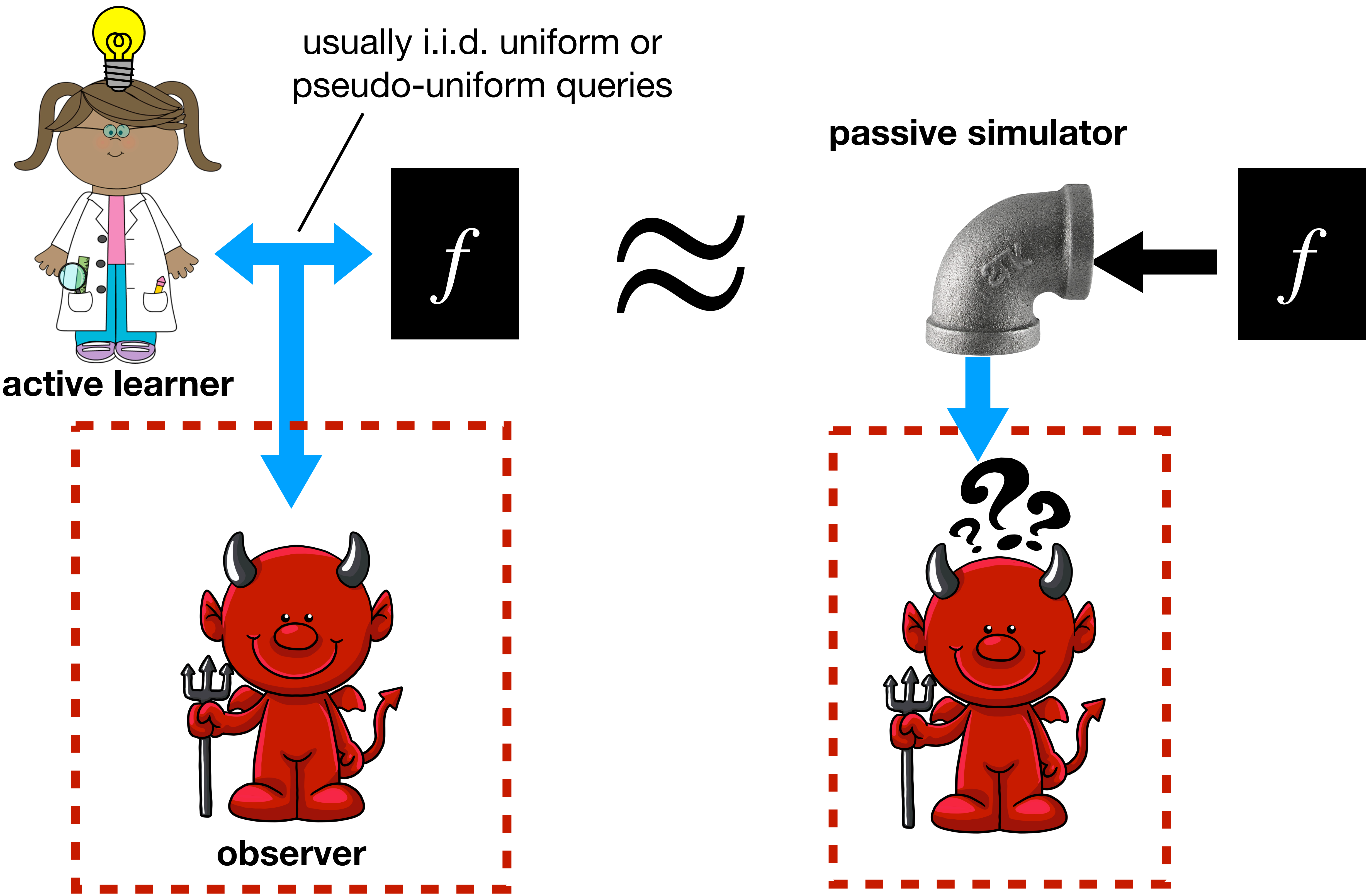
Covert Learning

[Canetti-Karchmer ’21, IKOS ’19]



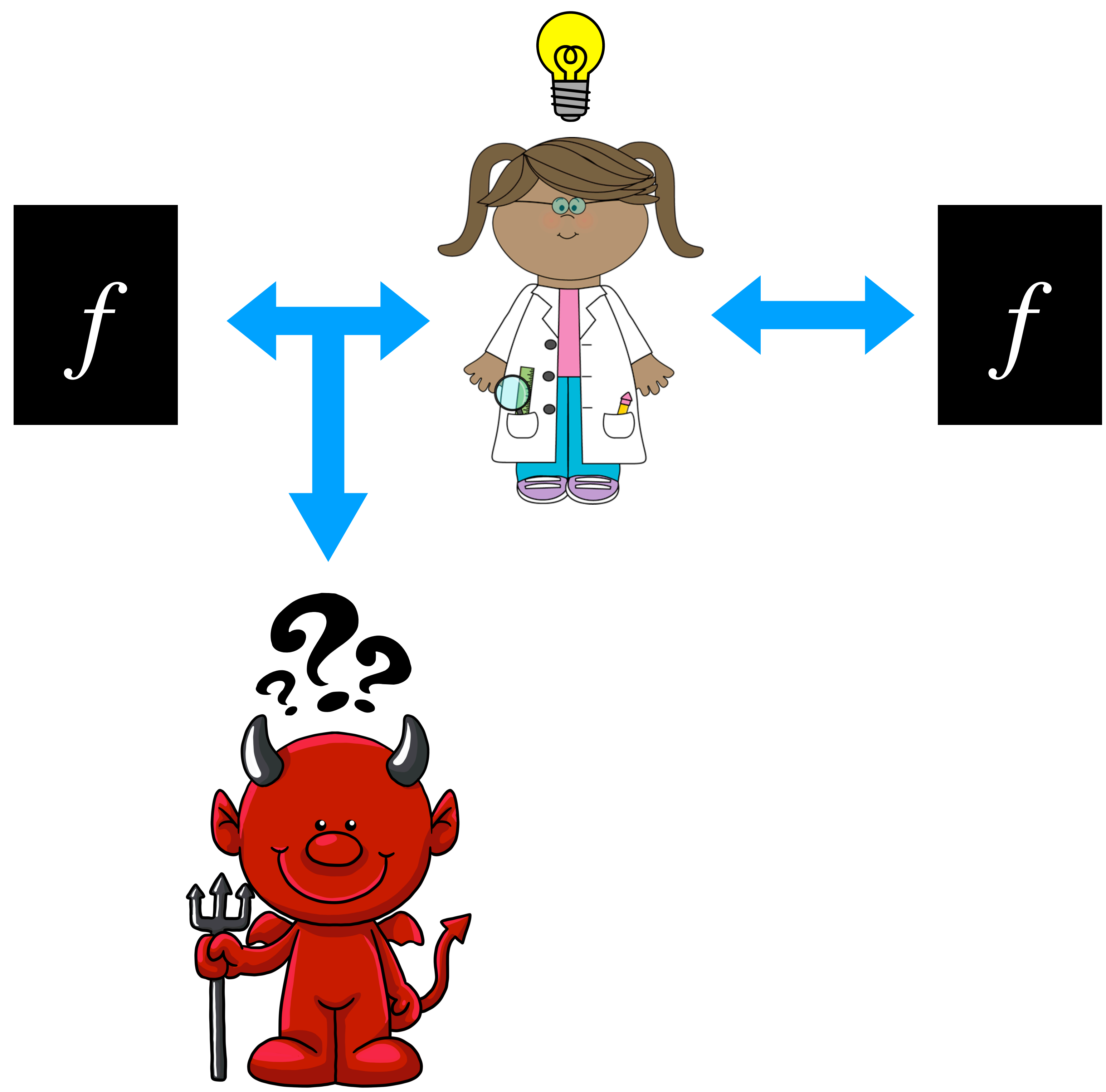
Covert Learning

[Canetti-Karchmer '21, IKOS '19]



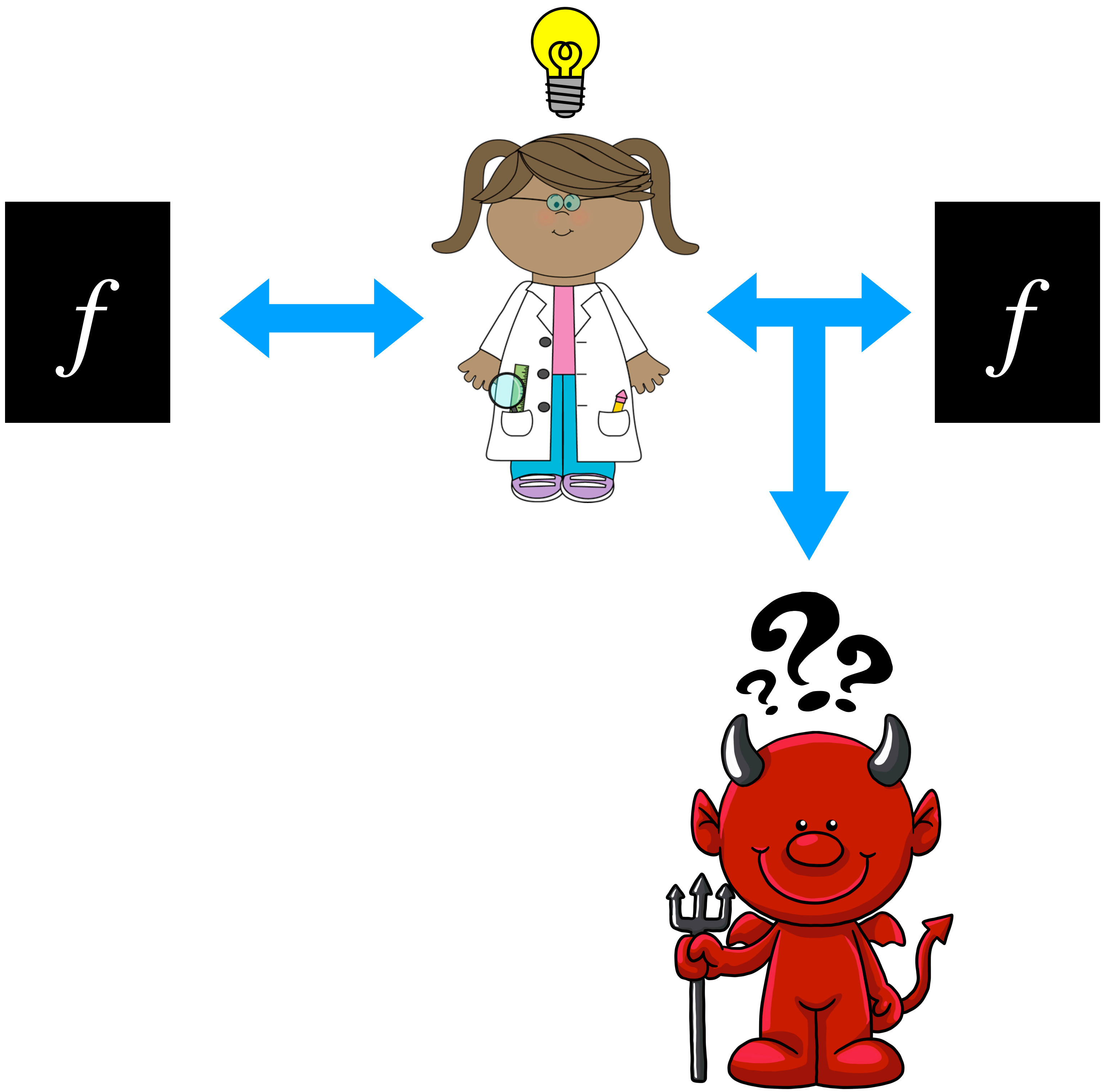
(1-of-2) Locally Covert Learning

[IKOS19]



(1-of-2) Locally Covert Learning

[IKOS19]



Why Study Covert Learning?
Scenario 1: Delegating Scientific Discovery

[Canetti-Karchmer21]

Why Study Covert Learning?
Scenario 1: Delegating Scientific Discovery

[Canetti-Karchmer21]

Plan: Learn a function f for which:
random examples are cheap / useless
queries are relatively useful but expensive,

Why Study Covert Learning?
Scenario 1: Delegating Scientific Discovery
[Canetti-Karchmer21]

Plan: Learn a function f for which:
random examples are cheap / useless
queries are relatively useful but expensive,

For example, an organism's genome → phenome map

Why Study Covert Learning?
Scenario 1: Delegating Scientific Discovery
[Canetti-Karchmer21]

Plan: Learn a function f for which:
random examples are cheap / useless
queries are relatively useful but expensive,

For example, an organism's genome → phenome map

Problem: Want to delegate to specialists, but …

Why Study Covert Learning?
Scenario 1: Delegating Scientific Discovery
[Canetti-Karchmer21]

Plan: Learn a function f for which:
random examples are cheap / useless
queries are relatively useful but expensive,

For example, an organism's genome → phenome map

Problem: Want to delegate to specialists, but …
what if they sell resulting data to your competitors?

Why Study Covert Learning?
Scenario 1: Delegating Scientific Discovery
[Canetti-Karchmer21]

Plan: Learn a function f for which:
random examples are cheap / useless
queries are relatively useful but expensive,

For example, an organism's genome $\to$ phenome map

Problem: Want to delegate to specialists, but …
what if they sell resulting data to your competitors?

Solution: use covert learning $\implies$ their data has no resale value

Scenario 2: Verifiable Learning

Scenario 2: Verifiable Learning

Same Plan: Delegate the query-learning of a function f; assume cheap but useless random examples for f.

Scenario 2: Verifiable Learning

Same Plan: Delegate the query-learning of a function f;
assume cheap but useless random examples for f.

New Problem [Goldwasser-Rothblum-Shafer-Yehudayoff '21]**:** How to ensure we receive a near-optimal circuit?

Scenario 2: Verifiable Learning

Same Plan: Delegate the query-learning of a function f;
assume cheap but useless random examples for f.

New Problem [Goldwasser-Rothblum-Shafer-Yehudayoff '21]**:** How to ensure we receive a near-optimal circuit?

One Approach: Tell learner what queries to make (following a covert learning algorithm). Hide "test queries" (using random examples)

Scenario 2: Verifiable Learning

Same Plan: Delegate the query-learning of a function f;
assume cheap but useless random examples for f.

New Problem [Goldwasser-Rothblum-Shafer-Yehudayoff '21]**:** How to ensure we receive a near-optimal circuit?

One Approach: Tell learner what queries to make (following a covert learning algorithm). Hide "test queries" (using random examples)

➡ If test queries are correct, most others must be as well.

Scenario 2: Verifiable Learning

Same Plan: Delegate the query-learning of a function f;
assume cheap but useless random examples for f.

New Problem [Goldwasser-Rothblum-Shafer-Yehudayoff '21]**:** How to ensure we receive a near-optimal circuit?

One Approach: Tell learner what queries to make (following a covert learning algorithm). Hide "test queries" (using random examples)

➡ If test queries are correct, most others must be as well.

➡ If learning algorithm is also "robust" then a few incorrect query answers can't ruin the output.

Scenario 3: Model Extraction

[Canetti-Karchmer 21]

Scenario 3: Model Extraction

[Canetti-Karchmer 21]

Plan: Sell AI as a service (e.g. chat GPT)

Scenario 3: Model Extraction

[Canetti-Karchmer 21]

Plan: Sell AI as a service (e.g. chat GPT)

- Generally trained on random data (more scalable)

Scenario 3: Model Extraction

[Canetti-Karchmer 21]

Plan: Sell AI as a service (e.g. chat GPT)

- Generally trained on random data (more scalable)

Problem: Can competitor use queries to clone the model?

Scenario 3: Model Extraction

[Canetti-Karchmer 21]

Plan: Sell AI as a service (e.g. chat GPT)

- Generally trained on random data (more scalable)

Problem: Can competitor use queries to clone the model?

Defense?? Block users who make weird query patterns

Scenario 3: Model Extraction

[Canetti-Karchmer 21]

Plan: Sell AI as a service (e.g. chat GPT)

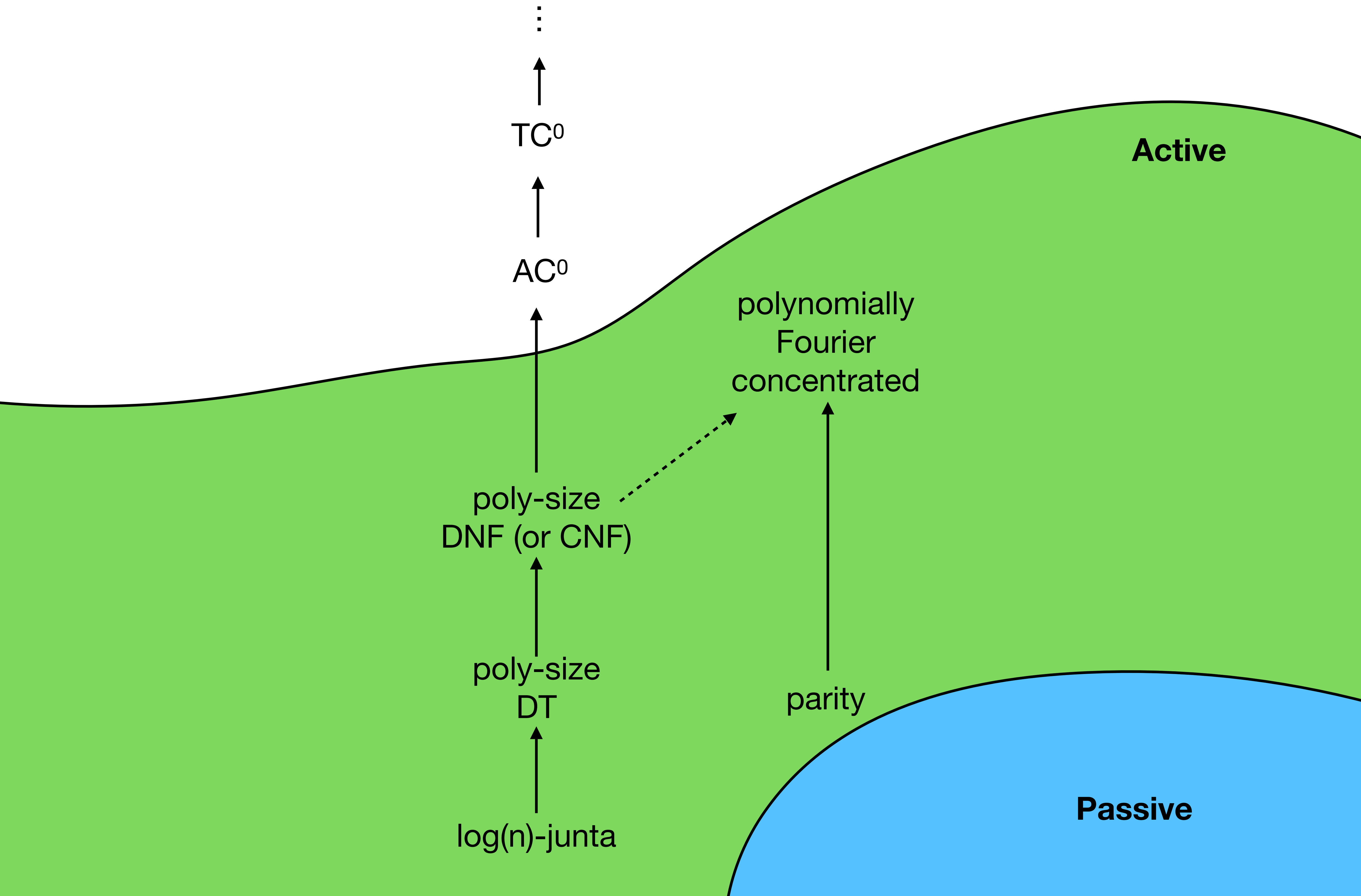
- Generally trained on random data (more scalable)

Problem: Can competitor use queries to clone the model?

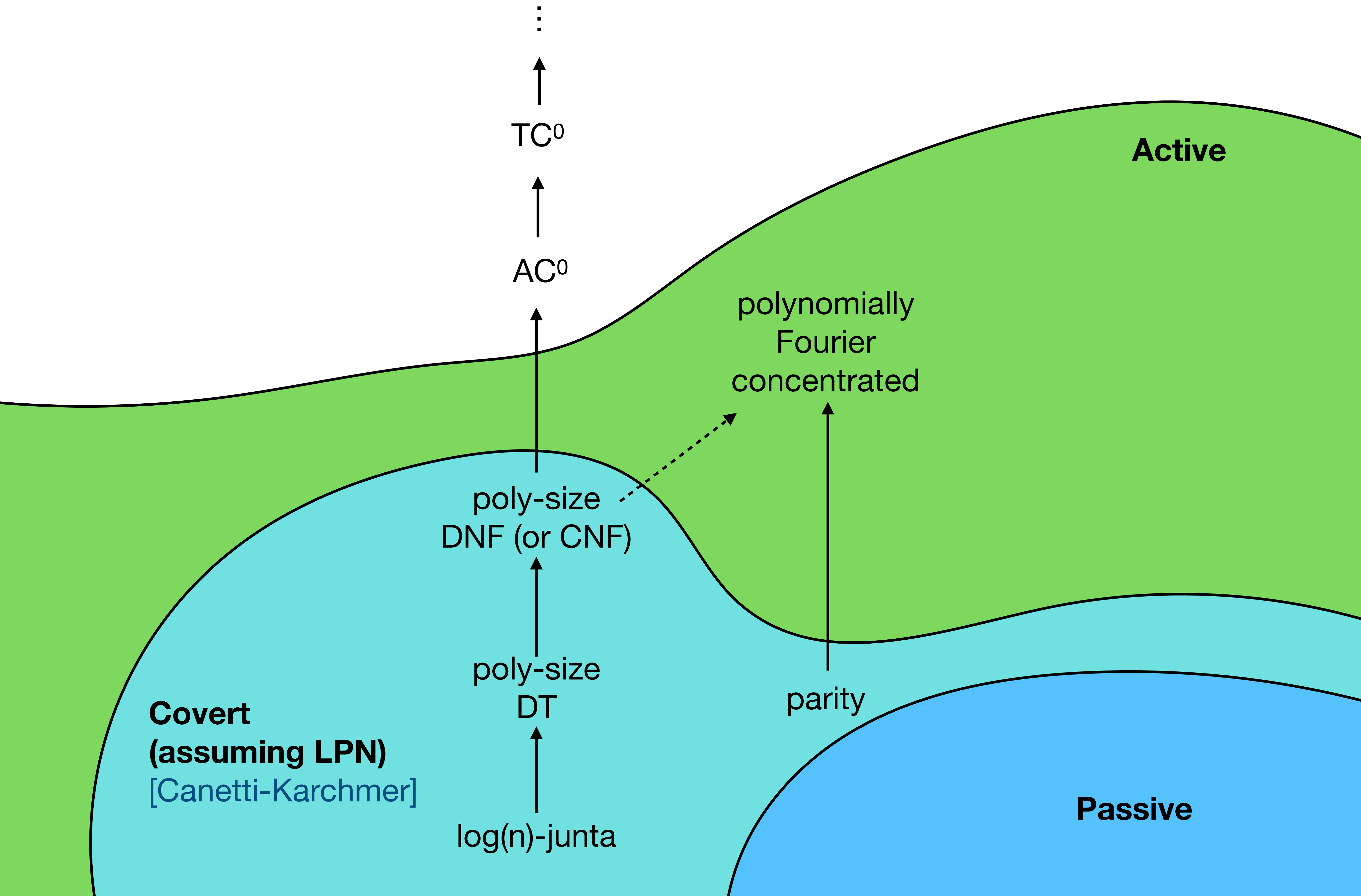
Defense?? Block users who make weird query patterns

Can’t really work against a covert learner 😢

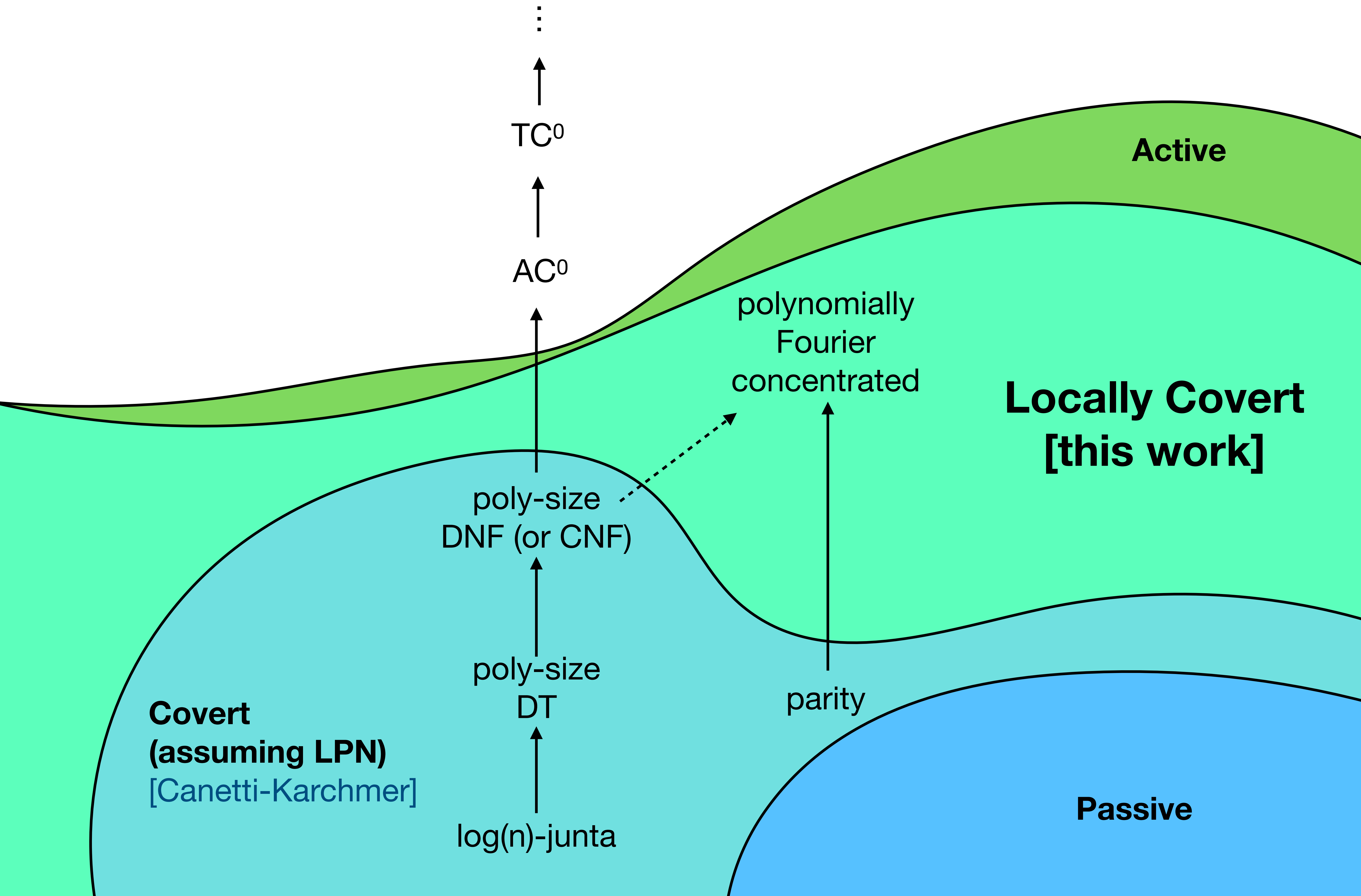
Passive $\leq$ Covert $\leq$ Active



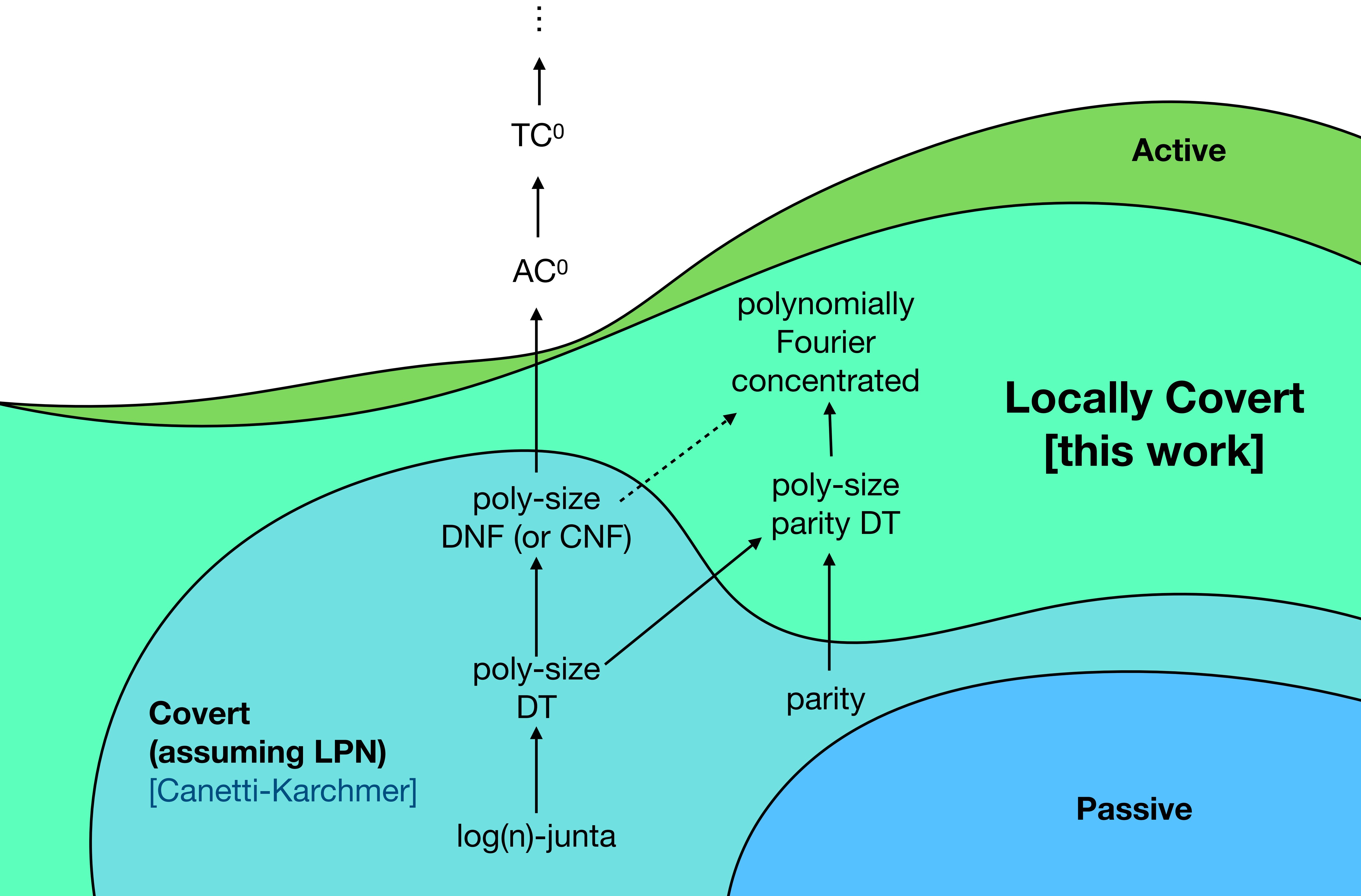
Passive $\leq$ Covert $\leq$ Active



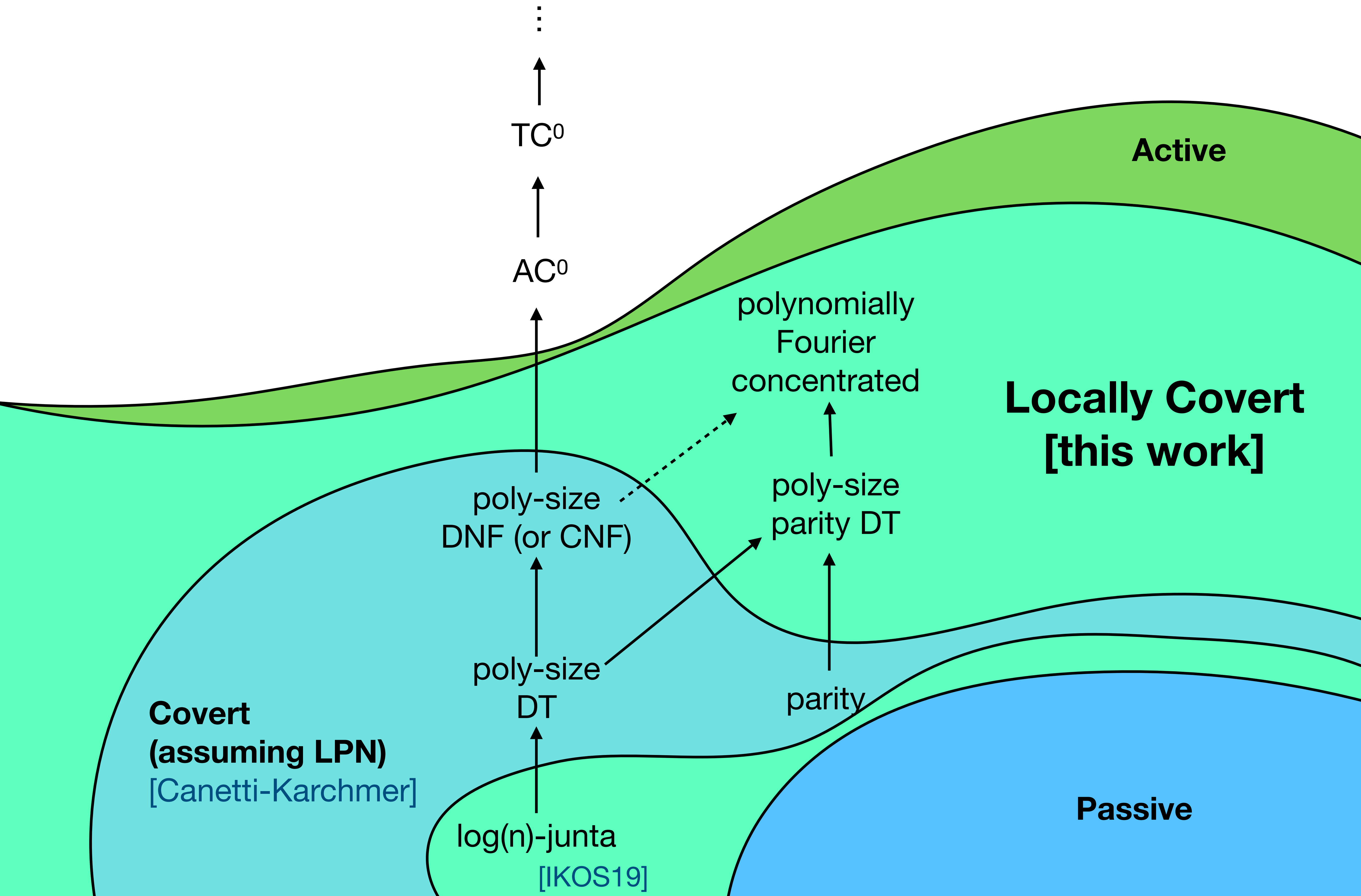
Passive $\leq$ Covert $\leq$ Active



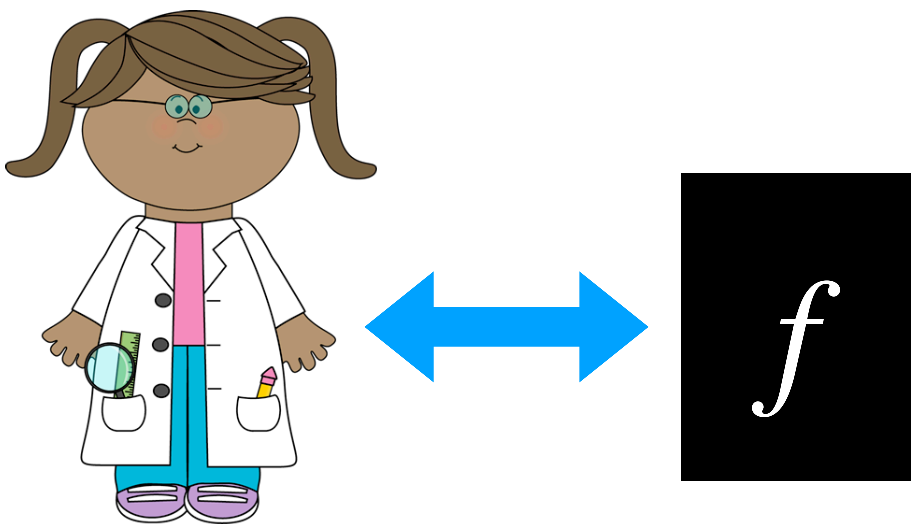
Passive $\leq$ Covert $\leq$ Active



Passive $\leq$ Covert $\leq$ Active



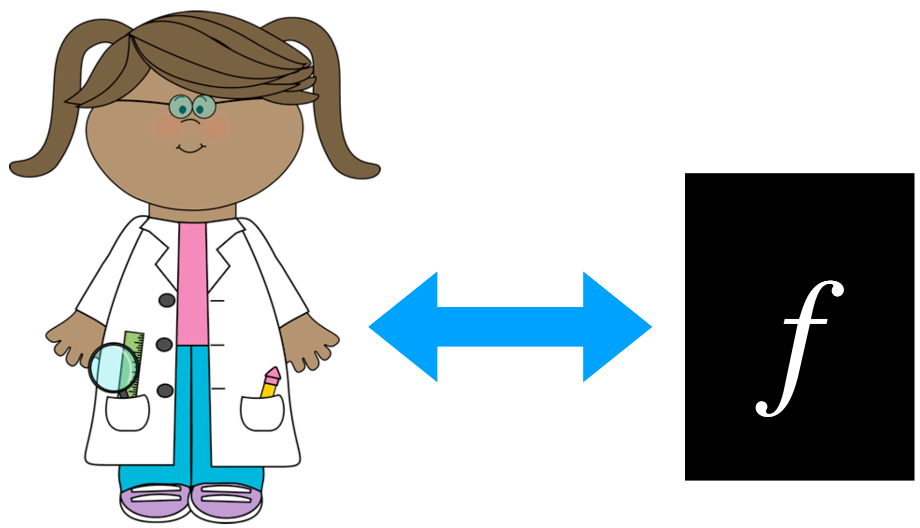
The Goldreich-Levin Theorem



Learning Version:

Given oracle access to f, one can efficiently find all *parity functions* γ that are even weakly correlated with f

The Goldreich-Levin Theorem



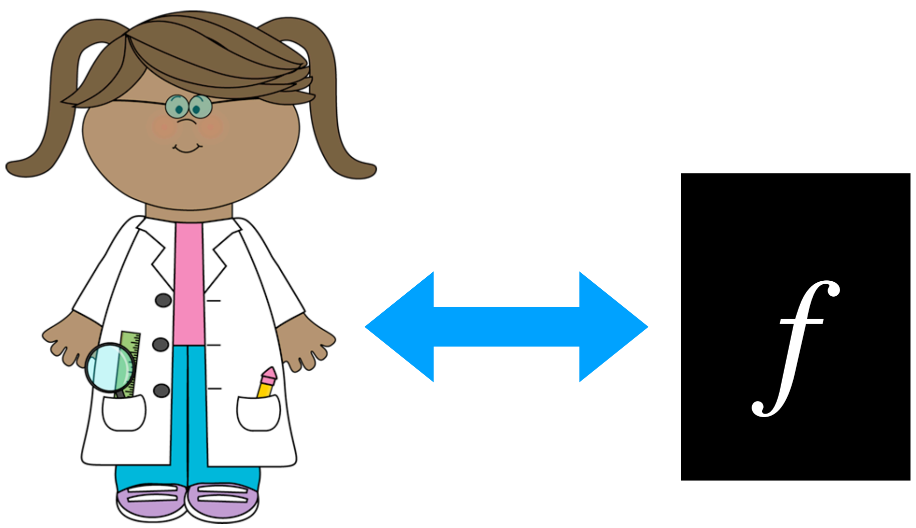
Learning Version:

Given oracle access to f, one can efficiently find all *parity functions* γ that are even weakly correlated with f

Crypto Version:

Let g be a OWF. Then $\langle \mathbf{x}, \mathbf{r} \rangle \pmod 2$ is *hard-core* for $(g(\mathbf{x}), \mathbf{r})$.

The Goldreich-Levin Theorem



Learning Version:

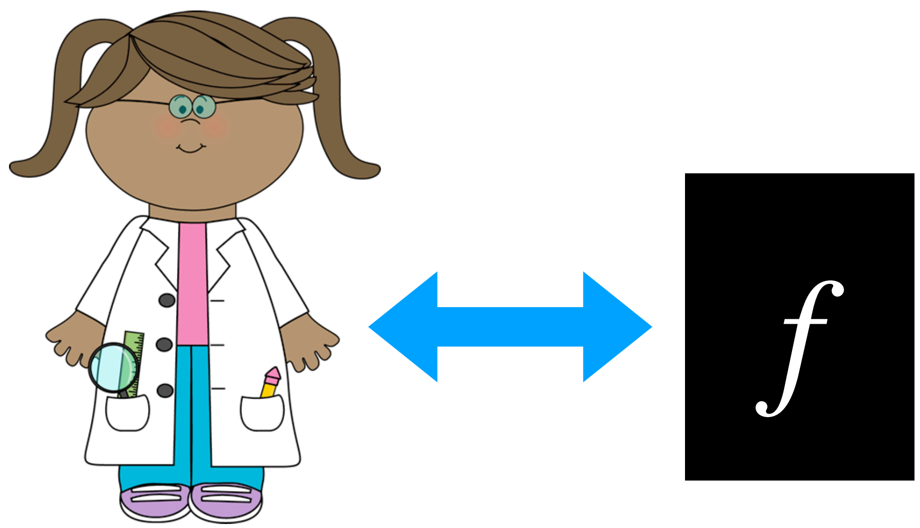
Given oracle access to f, one can efficiently find all *parity functions* γ that are even weakly correlated with f

Crypto Version:

Let g be a OWF. Then $\langle \mathbf{x}, \mathbf{r} \rangle \pmod 2$ is *hard-core* for $(g(\mathbf{x}), \mathbf{r})$.

Proof assuming Learning Version:

The Goldreich-Levin Theorem



Learning Version:

Given oracle access to f, one can efficiently find all *parity functions* γ that are even weakly correlated with f

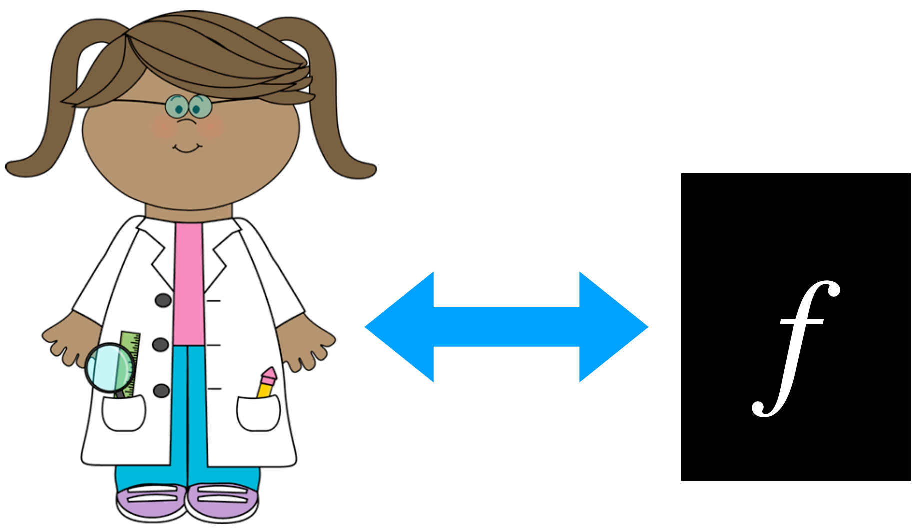
Crypto Version:

Let g be a OWF. Then $\langle \mathbf{x}, \mathbf{r} \rangle \pmod 2$ is *hard-core* for $(g(\mathbf{x}), \mathbf{r})$.

Proof assuming Learning Version:

1. $\langle \mathbf{x}, \cdot \rangle \pmod 2$ is a parity function.

The Goldreich-Levin Theorem



Learning Version:

Given oracle access to f, one can efficiently find all *parity functions* γ that are even weakly correlated with f

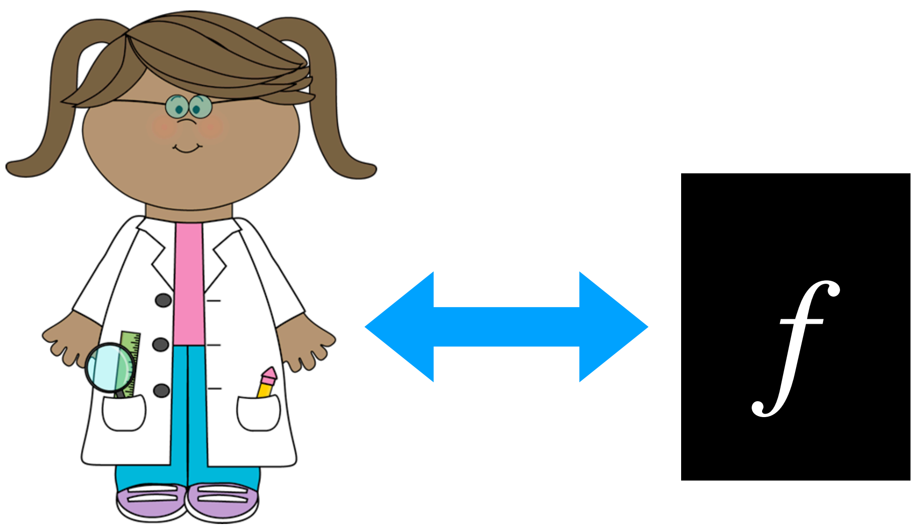
Crypto Version:

Let g be a OWF. Then $\langle \mathbf{x}, \mathbf{r} \rangle \pmod 2$ is *hard-core* for $(g(\mathbf{x}), \mathbf{r})$.

Proof assuming Learning Version:

1. $\langle \mathbf{x}, \cdot \rangle \pmod 2$ is a parity function.
2. If not hard-core, then an adversary $\mathscr{A}(g(\mathbf{x}), \cdot\,)$ weakly predicts $\langle \mathbf{x}, \mathbf{r} \rangle$.

The Goldreich-Levin Theorem



Learning Version:

Given oracle access to f, one can efficiently find all *parity functions* γ that are even weakly correlated with f

Crypto Version:

Let g be a OWF. Then $\langle \mathbf{x}, \mathbf{r} \rangle \pmod 2$ is *hard-core* for $(g(\mathbf{x}), \mathbf{r})$.

Proof assuming Learning Version:

1. $\langle \mathbf{x}, \cdot\, \rangle \pmod 2$ is a parity function.
2. If not hard-core, then an adversary $\mathscr{A}(g(\mathbf{x}), \cdot\,)$ weakly predicts $\langle \mathbf{x}, \mathbf{r} \rangle$.
3. $\mathrm{GL}^{\mathscr{A}(g(\mathbf{x}), \cdot)}$ outputs a list containing $\mathbf{x} \Longrightarrow$ contradicts that g is a OWF.

Which “Goldreich-Levin Algorithm”?

Which "Goldreich-Levin Algorithm"?

Rackoff's Algorithm

Which "Goldreich-Levin Algorithm"?

Rackoff's Algorithm

- Uses derandomization (querying all subset sums of $\approx \log(n)$ random vectors in $\mathbb{F}_2^n$)

Which "Goldreich-Levin Algorithm"?

Rackoff's Algorithm

- Uses derandomization (querying all subset sums of $\approx \log(n)$ random vectors in $\mathbb{F}_2^n$)

  ➡ Queries are not statistically uniform

Which “Goldreich-Levin Algorithm”?

Rackoff’s Algorithm

- Uses derandomization (querying all subset sums of $\approx \log(n)$ random vectors in $\mathbb{F}_2^n$)

  ➡ Queries are not statistically uniform

The Original [Goldreich-Levin]

Which "Goldreich-Levin Algorithm"?

Rackoff's Algorithm

- Uses derandomization (querying all subset sums of $\approx \log(n)$ random vectors in $\mathbb{F}_2^n$)

➡ Queries are not statistically uniform

The Original [Goldreich-Levin]

- Uses Fourier analysis

Which "Goldreich-Levin Algorithm"?

Rackoff's Algorithm

- Uses derandomization (querying all subset sums of $\approx \log(n)$ random vectors in $\mathbb{F}_2^n$)

➡ Queries are not statistically uniform

The Original [Goldreich-Levin]

- Uses Fourier analysis
- Well-known in learning theory

Which "Goldreich-Levin Algorithm"?

Rackoff's Algorithm

- Uses derandomization (querying all subset sums of $\approx \log(n)$ random vectors in $\mathbb{F}_2^n$)

➡ Queries are not statistically uniform

The Original [Goldreich-Levin]

- Uses Fourier analysis
- Well-known in learning theory

This Work:

Original algorithm is basically already 1-out-of-2 covert.

Which “Goldreich-Levin Algorithm”?

Rackoff’s Algorithm

- Uses derandomization (querying all subset sums of $\approx \log(n)$ random vectors in $\mathbb{F}_2^n$)

➡ Queries are not statistically uniform

The Original [Goldreich-Levin]

- Uses Fourier analysis
- Well-known in learning theory

This Work:

Original algorithm is basically already 1-out-of-2 covert.

Small modification gives $(k-1)$-out-of-k covertness.

(Locally) Covert Goldreich-Levin Algorithms

(Locally) Covert Goldreich-Levin Algorithms

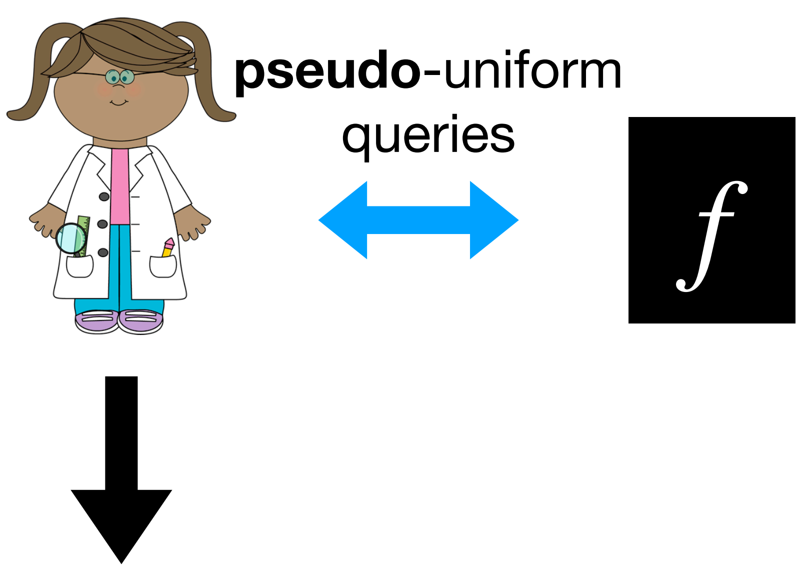
Previous Lemma: [Canetti-Karchmer]
(following [Rackoff])

Assuming LPN is subexponentially hard, there is a **computationally** covert algorithm for **low-degree** Goldreich-Levin learning

(Locally) Covert Goldreich-Levin Algorithms

Previous Lemma: [Canetti-Karchmer]
(following [Rackoff])

Assuming LPN is subexponentially hard, there is a **computationally** covert algorithm for **low-degree** Goldreich-Levin learning

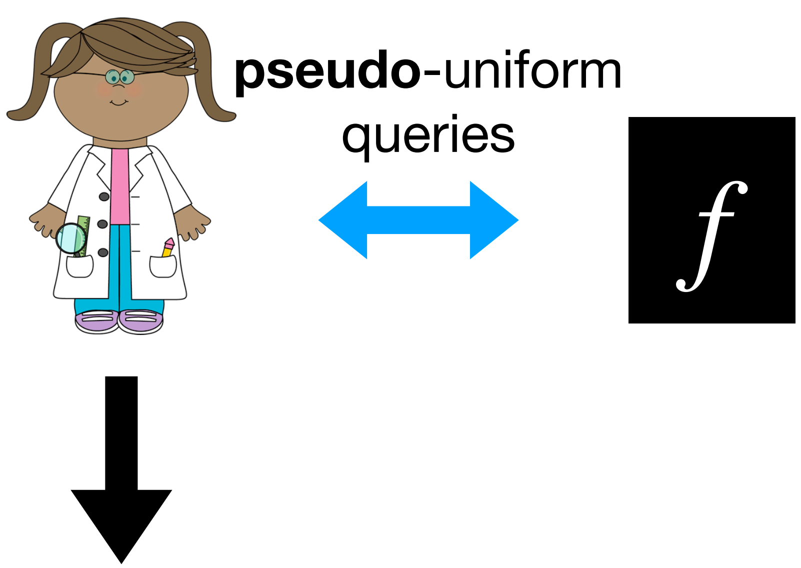


all **log(n)-variable** γ s.t. $|\hat{f}(\gamma)| \geq \epsilon$
(except with δ probability)

(Locally) Covert Goldreich-Levin Algorithms

Previous Lemma: [Canetti-Karchmer] (following [Rackoff])

Assuming LPN is subexponentially hard, there is a **computationally** covert algorithm for **low-degree** Goldreich-Levin learning



all **log(n)-variable** γ s.t. $|\hat{f}(\gamma)| \geq \epsilon$

(except with δ probability)

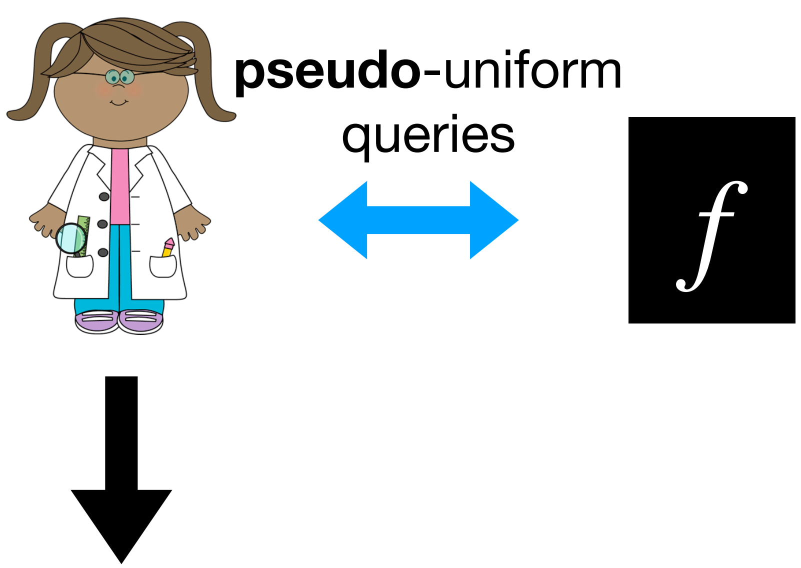
Our Main Theorem (following [Goldreich-Levin]):

For any constant k, there is a **perfectly** $(k-1)$-out-of-k covert algorithm for Goldreich-Levin learning

(Locally) Covert Goldreich-Levin Algorithms

Previous Lemma: [Canetti-Karchmer]
(following [Rackoff])

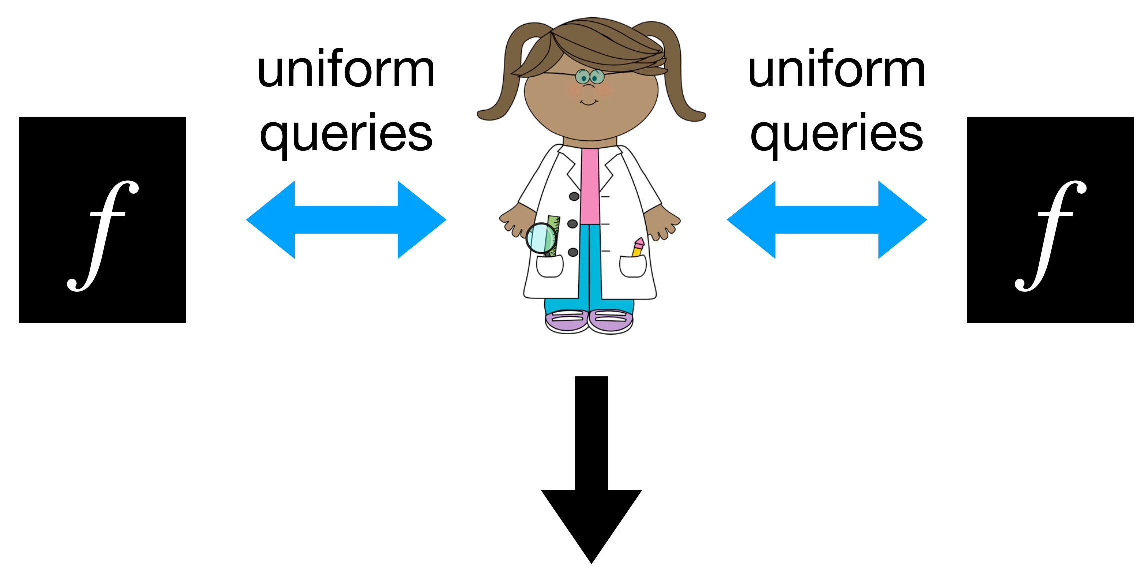
Assuming LPN is subexponentially hard, there is a **computationally** covert algorithm for **low-degree** Goldreich-Levin learning



all **log(n)-variable** γ s.t. $|\hat{f}(\gamma)| \geq \epsilon$
(except with δ probability)

Our Main Theorem
(following [Goldreich-Levin]):

For any constant k, there is a **perfectly** $(k-1)$-out-of-k covert algorithm for Goldreich-Levin learning



all γ s.t. $|\hat{f}(\gamma)| \geq \epsilon$
(except with δ probability)

Fourier Analysis Essentials

Fourier Analysis Essentials

For $\gamma \in \mathbb{F}_2^n$, let $\hat{f}(\gamma) \in [-1,1]$ denote the *correlation* of f with the parity function $\langle \gamma, \cdot \rangle$. Call $\hat{f}(\gamma)^2$ the *weight* of γ.

Fourier Analysis Essentials

For $\gamma \in \mathbb{F}_2^n$, let $\hat{f}(\gamma) \in [-1,1]$ denote the *correlation* of f with the parity function $\langle \gamma, \cdot \rangle$. Call $\hat{f}(\gamma)^2$ the *weight* of γ.

Fact: $\sum_{\gamma \in \mathbb{F}_2^n} \hat{f}(\gamma)^2 = 1.$

Fourier Analysis Essentials

For $\gamma \in \mathbb{F}_2^n$, let $\hat{f}(\gamma) \in [-1,1]$ denote the *correlation* of f with the parity function $\langle \gamma, \cdot \rangle$. Call $\hat{f}(\gamma)^2$ the *weight* of γ.

Fact: $\sum_{\gamma \in \mathbb{F}_2^n} \hat{f}(\gamma)^2 = 1.$

Not many heavy parities.

Fourier Analysis Essentials

For $\gamma \in \mathbb{F}_2^n$, let $\hat{f}(\gamma) \in [-1,1]$ denote the *correlation* of f with the parity function $\langle \gamma, \cdot \rangle$. Call $\hat{f}(\gamma)^2$ the *weight* of γ.

Fact: $\displaystyle\sum_{\gamma \in \mathbb{F}_2^n} \hat{f}(\gamma)^2 = 1.$ Not many heavy parities.

Lemma: With queries to f, one can efficiently estimate $\text{weight}(p) := \displaystyle\sum_{s \in \mathbb{F}_2^{n-k}} \hat{f}(p \circ s)^2$ for *any "prefix"* $p \in \mathbb{F}_2^k$,

Fourier Analysis Essentials

For $\gamma \in \mathbb{F}_2^n$, let $\hat{f}(\gamma) \in [-1,1]$ denote the *correlation* of f with the parity function $\langle \gamma, \cdot \rangle$. Call $\hat{f}(\gamma)^2$ the *weight* of γ.

Fact: $\sum_{\gamma \in \mathbb{F}_2^n} \hat{f}(\gamma)^2 = 1.$

Not many heavy parities.

Lemma: With queries to f, one can efficiently estimate

$\text{weight}(p) := \sum_{s \in \mathbb{F}_2^{n-k}} \hat{f}(p \circ s)^2$ for *any "prefix"* $p \in \mathbb{F}_2^k$,

concatenation

Fourier Analysis Essentials

For $\gamma \in \mathbb{F}_2^n$, let $\hat{f}(\gamma) \in [-1,1]$ denote the *correlation* of f with the parity function $\langle \gamma, \cdot \rangle$. Call $\hat{f}(\gamma)^2$ the *weight* of γ.

Fact: $\sum_{\gamma \in \mathbb{F}_2^n} \hat{f}(\gamma)^2 = 1.$

Not many heavy parities.

Not many heavy prefixes.

Lemma: With queries to f, one can efficiently estimate

$$\text{weight}(p) := \sum_{s \in \mathbb{F}_2^{n-k}} \hat{f}(p \circ s)^2$$ for *any "prefix"* $p \in \mathbb{F}_2^k$,

concatenation

Fourier Analysis Essentials

For $\gamma \in \mathbb{F}_2^n$, let $\hat{f}(\gamma) \in [-1,1]$ denote the *correlation* of f with the parity function $\langle \gamma, \cdot \rangle$. Call $\hat{f}(\gamma)^2$ the *weight* of γ.

Fact: $\displaystyle\sum_{\gamma \in \mathbb{F}_2^n} \hat{f}(\gamma)^2 = 1.$

Not many heavy parities.

Not many heavy prefixes.

and 1-of-2 covertly

Lemma: With queries to f, one can efficiently estimate

$\mathrm{weight}(p) := \displaystyle\sum_{s \in \mathbb{F}_2^{n-k}} \hat{f}(p \circ s)^2$ for *any "prefix"* $p \in \mathbb{F}_2^k$,

concatenation

Fourier Analysis Essentials

For $\gamma \in \mathbb{F}_2^n$, let $\hat{f}(\gamma) \in [-1,1]$ denote the *correlation* of f with the parity function $\langle \gamma, \cdot \rangle$. Call $\hat{f}(\gamma)^2$ the *weight* of γ.

Fact: $\sum_{\gamma \in \mathbb{F}_2^n} \hat{f}(\gamma)^2 = 1.$

Not many heavy parities.

Not many heavy prefixes.

and 1-of-2 covertly

Lemma: With queries to f, one can efficiently estimate

$$\mathrm{weight}(p) := \sum_{s \in \mathbb{F}_2^{n-k}} \hat{f}(p \circ s)^2$$ for *any "prefix"* $p \in \mathbb{F}_2^k$,

concatenation

will prove this later

Weighing Parity Prefixes $\Longrightarrow$ Goldreich-Levin

Weighing Parity Prefixes $\Longrightarrow$ Goldreich-Levin

Basic Idea: Maintain a list of candidate *prefixes* of heavy parities γ (those with $\hat{f}(\gamma)^2 \geq \epsilon$), starting with 1-bit prefixes $\{0,1\}$.

Weighing Parity Prefixes $\implies$ Goldreich-Levin

Basic Idea: Maintain a list of candidate *prefixes* of heavy parities γ (those with $\hat{f}(\gamma)^2 \geq \epsilon$), starting with 1-bit prefixes $\{0,1\}$.

1. Weigh each prefix in the list and throw away light prefixes (those with weight $< \epsilon$)

   ➡ At most $1/\epsilon$ prefixes.

Weighing Parity Prefixes $\Longrightarrow$ Goldreich-Levin

Basic Idea: Maintain a list of candidate *prefixes* of heavy parities γ (those with $\hat{f}(\gamma)^2 \geq \epsilon$), starting with 1-bit prefixes $\{0,1\}$.

1. Weigh each prefix in the list and throw away light prefixes (those with weight $< \epsilon$)
   - ➡ At most $1/\epsilon$ prefixes.
2. Replace remaining prefixes p by $p \circ 0$ and $p \circ 1$
   - ➡ At most $2/\epsilon$ prefixes

Weighing Parity Prefixes $\implies$ Goldreich-Levin

Basic Idea: Maintain a list of candidate *prefixes* of heavy parities γ (those with $\hat{f}(\gamma)^2 \geq \epsilon$), starting with 1-bit prefixes $\{0,1\}$.

1. Weigh each prefix in the list and throw away light prefixes (those with weight $< \epsilon$)

   ➡ At most $1/\epsilon$ prefixes.

2. Replace remaining prefixes p by $p \circ 0$ and $p \circ 1$

   ➡ At most $2/\epsilon$ prefixes

3. Repeat until prefixes are n-bit strings.

How To Weigh Prefixes

How To Weigh Prefixes

Lemma:

With queries to f, one can efficiently estimate

$\mathrm{weight}(p) := \sum_{s \in \mathbb{F}_2^{n-k}} \hat{f}(p \circ s)^2$ for *any "prefix"* $p \in \mathbb{F}_2^k$.

How To Weigh Prefixes

Lemma:

With queries to f, one can efficiently estimate

weight$(p) := \sum_{s \in \mathbb{F}_2^{n-k}} \hat{f}(p \circ s)^2$ for *any "prefix"* $p \in \mathbb{F}_2^k$.

More Generally:

With queries to f, one can efficiently estimate

weight$(A) := \sum_{\gamma \in A} \hat{f}(\gamma)^2$ for *any affine subspace* $A \subseteq \mathbb{F}_2^n$

How To Weigh ~~Prefixes~~

Affine Spaces

Lemma:

With queries to f, one can efficiently estimate

$$\text{weight}(p) := \sum_{s \in \mathbb{F}_2^{n-k}} \hat{f}(p \circ s)^2$$ for *any "prefix"* $p \in \mathbb{F}_2^k$.

More Generally:

With queries to f, one can efficiently estimate

$$\text{weight}(A) := \sum_{\gamma \in A} \hat{f}(\gamma)^2$$ for *any affine subspace* $A \subseteq \mathbb{F}_2^n$

How To Weigh ~~Prefixes~~

Affine Spaces

Lemma:

and 1-of-2 covertly

With queries to f, one can efficiently estimate

$$\text{weight}(p) := \sum_{s \in \mathbb{F}_2^{n-k}} \hat{f}(p \circ s)^2$$ for *any "prefix"* $p \in \mathbb{F}_2^k$.

More Generally:

and 1-of-2 covertly

With queries to f, one can efficiently estimate

$$\text{weight}(A) := \sum_{\gamma \in A} \hat{f}(\gamma)^2$$ for *any affine subspace* $A \subseteq \mathbb{F}_2^n$

How To Weigh Affine Spaces

Lemma:

For *any affine subspace* $A \subseteq \mathbb{F}_2^n, \quad A = \gamma^\star + V$,

one can efficiently estimate $\mathsf{weight}(A) := \sum_{\gamma \in A} \hat{f}(\gamma)^2$

How To Weigh Affine Spaces

Lemma:

For *any affine subspace* $A \subseteq \mathbb{F}_2^n, \quad A = \gamma^\star + V$,

one can efficiently estimate $\text{weight}(A) := \sum_{\gamma \in A} \hat{f}(\gamma)^2$

Formula:

$$\sum_{\gamma \in A} \hat{f}(\gamma)^2 = \mathop{\mathbb{E}}_{x_1 + x_2 \in V^\perp} \left[f(x_1) \cdot f(x_2) \cdot \gamma^\star(x_1 + x_2) \right]$$

How To Weigh Affine Spaces

Lemma:

For *any affine subspace* $A \subseteq \mathbb{F}_2^n, \quad A = \gamma^\star + V,$

one can efficiently estimate $\text{weight}(A) := \sum_{\gamma \in A} \hat{f}(\gamma)^2$

Formula:

$$\sum_{\gamma \in A} \hat{f}(\gamma)^2 = \mathop{\mathbb{E}}_{x_1 + x_2 \in V^\perp} \left[\, f(x_1) \cdot f(x_2) \cdot \gamma^\star(x_1 + x_2) \right]$$

Expectation can be directly empirically estimated

How To Weigh Affine Spaces

Lemma:

For *any affine subspace* $A \subseteq \mathbb{F}_2^n$, $\quad A = \gamma^\star + V$,

one can efficiently estimate $\text{weight}(A) := \sum_{\gamma \in A} \hat{f}(\gamma)^2$

Formula:

x_1 and x_2 individually uniformly random

$$\sum_{\gamma \in A} \hat{f}(\gamma)^2 = \mathop{\mathbb{E}}_{x_1 + x_2 \in V^\perp} \left[f(x_1) \cdot f(x_2) \cdot \gamma^\star(x_1 + x_2) \right]$$

Expectation can be directly empirically estimated

How To Weigh Affine Spaces

Lemma:

For *any affine subspace* $A \subseteq \mathbb{F}_2^n, \quad A = \gamma^\star + V,$

one can efficiently estimate $\mathrm{weight}(A) := \sum_{\gamma \in A} \hat{f}(\gamma)^2$

1-of-2 covertly

Formula:

x_1 and x_2 individually uniformly random

$$\sum_{\gamma \in A} \hat{f}(\gamma)^2 = \mathop{\mathbb{E}}_{x_1 + x_2 \in V^\perp} \left[f(x_1) \cdot f(x_2) \cdot \gamma^\star(x_1 + x_2) \right]$$

Expectation can be directly empirically estimated

$(k-1)$-of-k Covertness

$(k-1)$-of-k Covertness

Previous formula naturally generalizes:

If $A = \gamma^\star + V$ is an affine subspace of $\mathbb{F}_2^n$, then

$$\sum_{\gamma \in A} \hat{f}(\gamma)^k = \mathop{\mathbb{E}}_{x_1+\cdots+x_k \in V^\perp}\left[\, f(x_1)\cdots f(x_k) \cdot \gamma^\star(x_1 + \cdots x_k)\right]$$

$(k-1)$-of-k Covertness

Previous formula naturally generalizes:

If $A = \gamma^\star + V$ is an affine subspace of $\mathbb{F}_2^n$, then

$$\sum_{\gamma \in A} \hat{f}(\gamma)^k = \mathop{\mathbb{E}}_{x_1+\cdots+x_k \in V^\perp} \left[f(x_1)\cdots f(x_k) \cdot \gamma^\star(x_1 + \cdots x_k) \right]$$

"k-weight of A"

$(k-1)$-of-k Covertness

Previous formula naturally generalizes:

If $A = \gamma^\star + V$ is an affine subspace of $\mathbb{F}_2^n$, then

$$\sum_{\gamma \in A} \hat{f}(\gamma)^k = \mathop{\mathbb{E}}_{x_1 + \cdots + x_k \in V^\perp} \left[f(x_1) \cdots f(x_k) \cdot \gamma^\star(x_1 + \cdots x_k) \right]$$

"k-weight of A"

- To get $(k-1)$-of-k covert GL, apply same strategy, using k-weight instead of 2-weight

$(k-1)$-of-k Covertness

Previous formula naturally generalizes:

If $A = \gamma^\star + V$ is an affine subspace of $\mathbb{F}_2^n$, then

$$\sum_{\gamma \in A} \hat{f}(\gamma)^k = \mathop{\mathbb{E}}_{x_1 + \cdots + x_k \in V^\perp} \left[f(x_1) \cdots f(x_k) \cdot \gamma^\star(x_1 + \cdots x_k) \right]$$

"k-weight of A"

$(k-1)$-wise uniform

- To get $(k-1)$-of-k covert GL, apply same strategy, using k-weight instead of 2-weight

Goldreich-Levin with k-weights
(k even WLOG)

Goldreich-Levin with k-weights
(k even WLOG)

Strategy: Maintain a list of candidate l*-bit prefixes* of heavy parities γ (those with $\hat{f}(\gamma)^2 \geq \epsilon$), starting with $\{0,1\}$.

Goldreich-Levin with k-weights (k even WLOG)

Strategy: Maintain a list of candidate l*-bit prefixes* of heavy parities γ (those with $\hat{f}(\gamma)^2 \geq \epsilon$), starting with $\{0,1\}$.

1. Weigh each prefix in the list and throw away light prefixes (those with k-weight $< \epsilon^{k/2}$)

Goldreich-Levin with k-weights (k even WLOG)

Strategy: Maintain a list of candidate l*-bit prefixes* of heavy parities γ (those with $\hat{f}(\gamma)^2 \geq \epsilon$), starting with $\{0,1\}$.

1. Weigh each prefix in the list and throw away light prefixes (those with k-weight $< \epsilon^{k/2}$)
   - ➡ At most $1/\epsilon^{k/2}$ prefixes because 2-weight > k-weight

Goldreich-Levin with k-weights
(k even WLOG)

Strategy: Maintain a list of candidate l*-bit prefixes* of heavy parities γ (those with $\hat{f}(\gamma)^2 \geq \epsilon$), starting with $\{0,1\}$.

1. Weigh each prefix in the list and throw away light prefixes (those with k-weight $< \epsilon^{k/2}$)
   - ➡ At most $1/\epsilon^{k/2}$ prefixes because 2-weight > k-weight
2. Replace remaining prefixes p by $p \circ 0$ and $p \circ 1$

Goldreich-Levin with k-weights (k even WLOG)

Strategy: Maintain a list of candidate l*-bit prefixes* of heavy parities γ (those with $\hat{f}(\gamma)^2 \geq \epsilon$), starting with $\{0,1\}$.

1. Weigh each prefix in the list and throw away light prefixes (those with k-weight $< \epsilon^{k/2}$)
   - ➡ At most $1/\epsilon^{k/2}$ prefixes because 2-weight > k-weight
2. Replace remaining prefixes p by $p \circ 0$ and $p \circ 1$
   - ➡ At most $2/\epsilon^{k/2}$ prefixes

Goldreich-Levin with k-weights (k even WLOG)

Strategy: Maintain a list of candidate *l-bit prefixes* of heavy parities γ (those with $\hat{f}(\gamma)^2 \geq \epsilon$), starting with $\{0,1\}$.

1. Weigh each prefix in the list and throw away light prefixes (those with k-weight $< \epsilon^{k/2}$)

   ➡ At most $1/\epsilon^{k/2}$ prefixes because 2-weight > k-weight

2. Replace remaining prefixes p by $p \circ 0$ and $p \circ 1$

   ➡ At most $2/\epsilon^{k/2}$ prefixes

3. Repeat until prefixes are n-bit strings.

Goldreich-Levin with k-weights (k even WLOG)

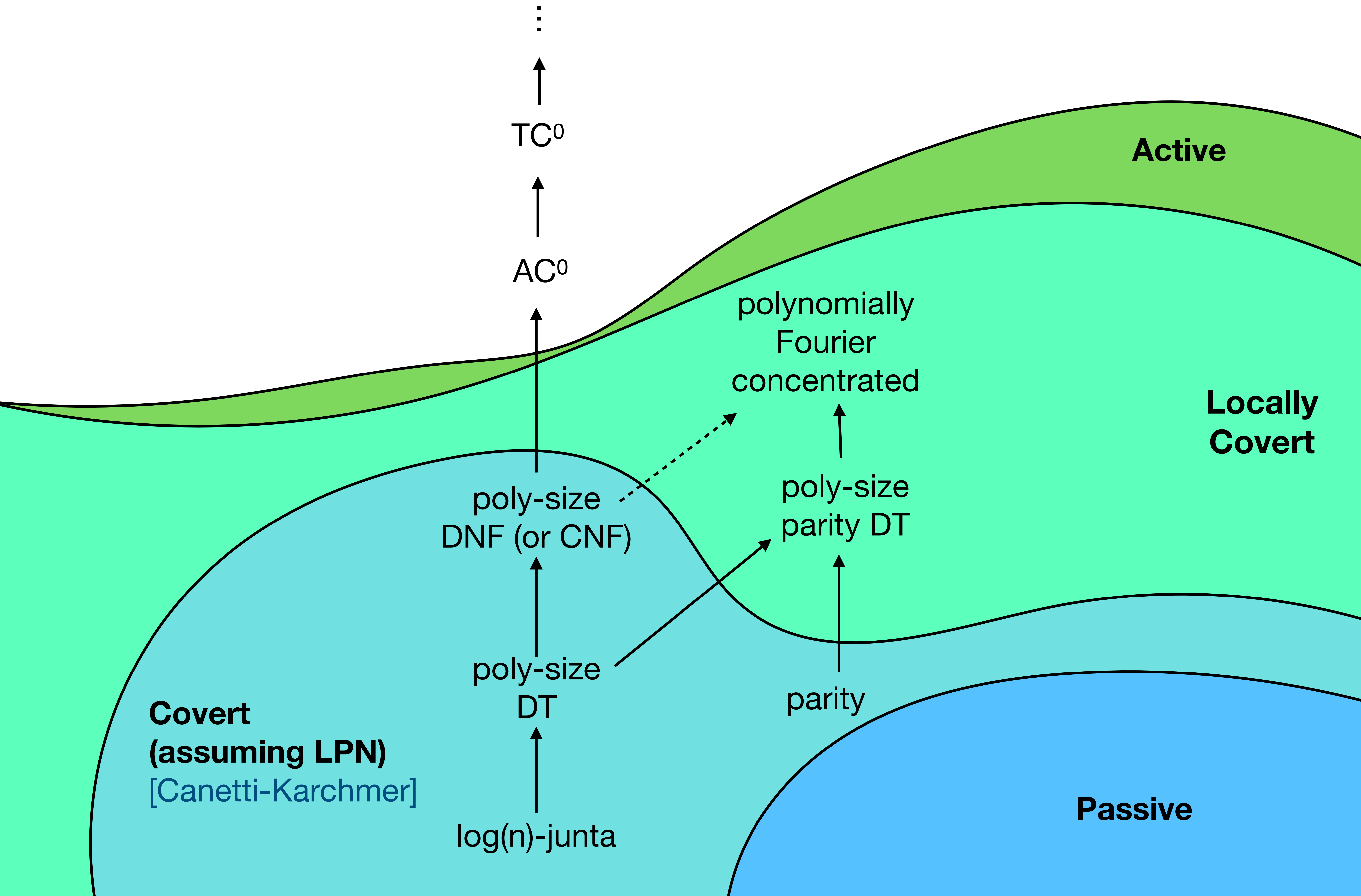
Strategy: Maintain a list of candidate l-*bit prefixes* of heavy parities γ (those with $\hat{f}(\gamma)^2 \geq \epsilon$), starting with $\{0,1\}$.

1. Weigh each prefix in the list and throw away light prefixes (those with k-weight $< \epsilon^{k/2}$)
   - ➡ At most $1/\epsilon^{k/2}$ prefixes because 2-weight > k-weight
2. Replace remaining prefixes p by $p \circ 0$ and $p \circ 1$
   - ➡ At most $2/\epsilon^{k/2}$ prefixes

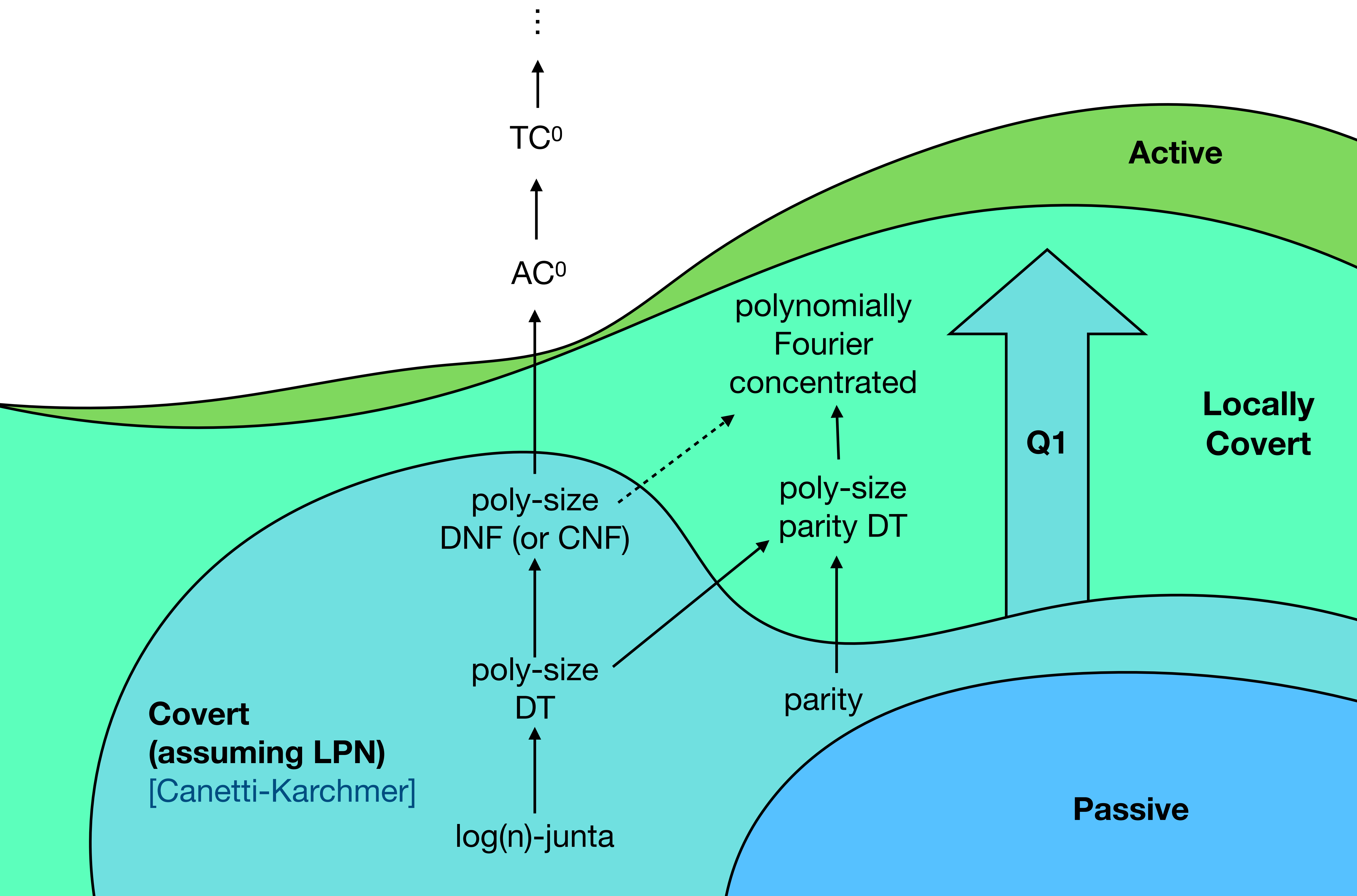
running time is exponential in k

3. Repeat until prefixes are n-bit strings.

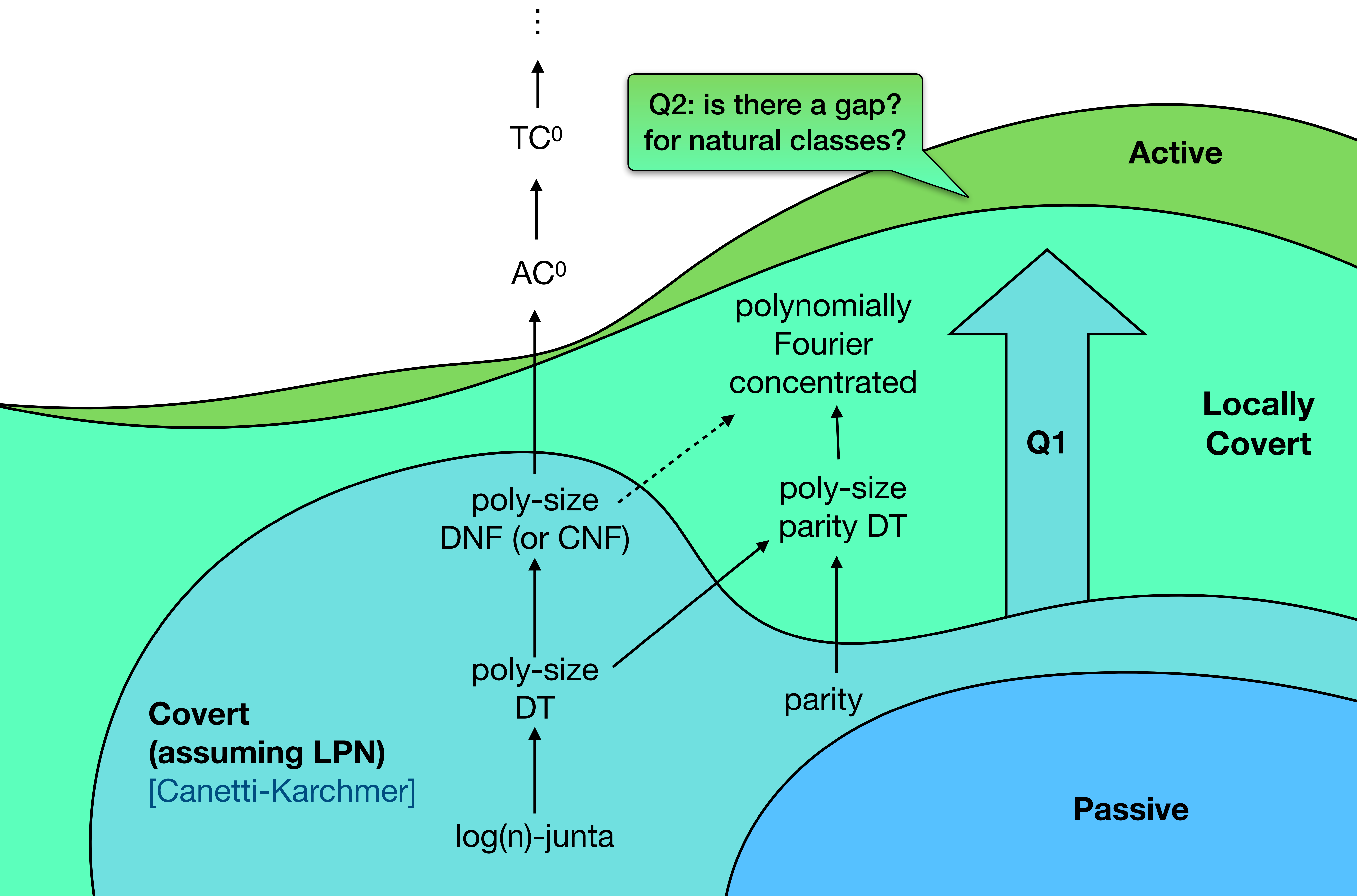
Future Work?


⋮
TC⁰
AC⁰
polynomially
Fourier
concentrated
Active
Locally
Covert
poly-size
DNF (or CNF)
poly-size
parity DT
poly-size
DT
parity
Covert
(assuming LPN)
[Canetti-Karchmer]
log(n)-junta
Passive

Future Work?

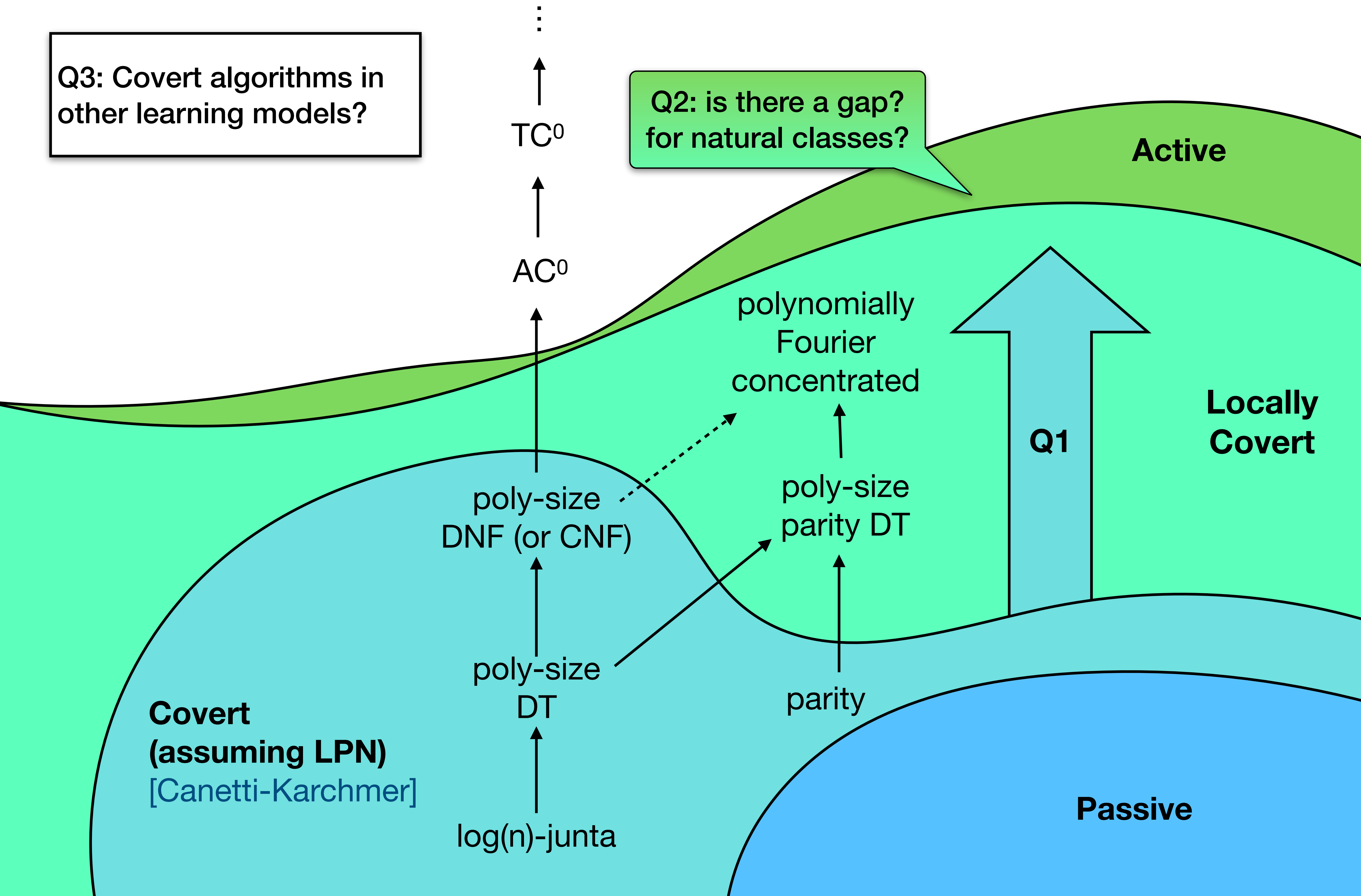

⋮
TC^0
AC^0
Active
polynomially Fourier concentrated
Locally Covert
Q1
poly-size DNF (or CNF)
poly-size parity DT
poly-size DT
parity
Covert (assuming LPN)
[Canetti-Karchmer]
log(n)-junta
Passive

Future Work?

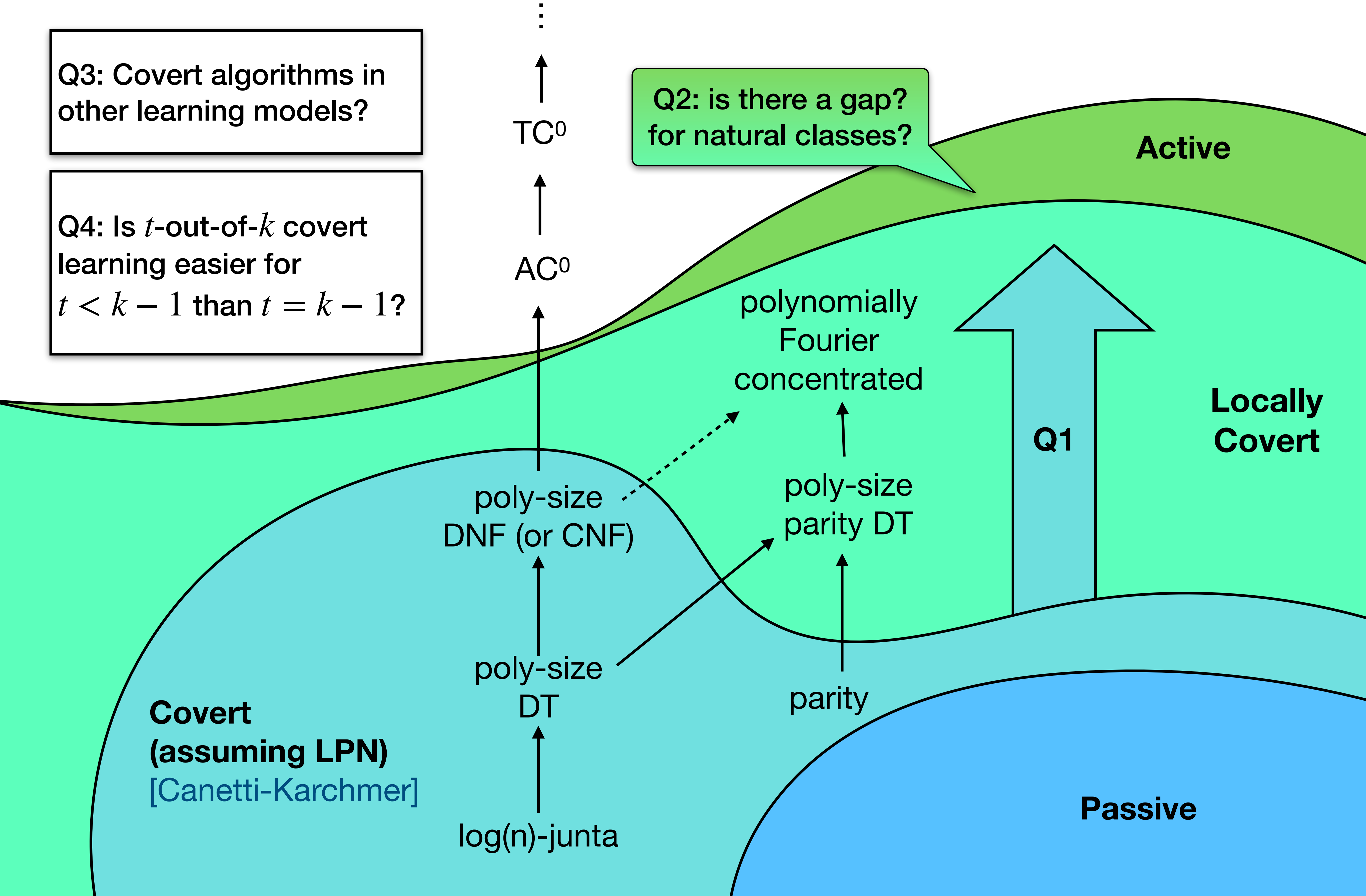


⋮
TC^0
AC^0
Q2: is there a gap? for natural classes?
Active
polynomially Fourier concentrated
Locally Covert
Q1
poly-size DNF (or CNF)
poly-size parity DT
poly-size DT
parity
Covert (assuming LPN)
[Canetti-Karchmer]
log(n)-junta
Passive

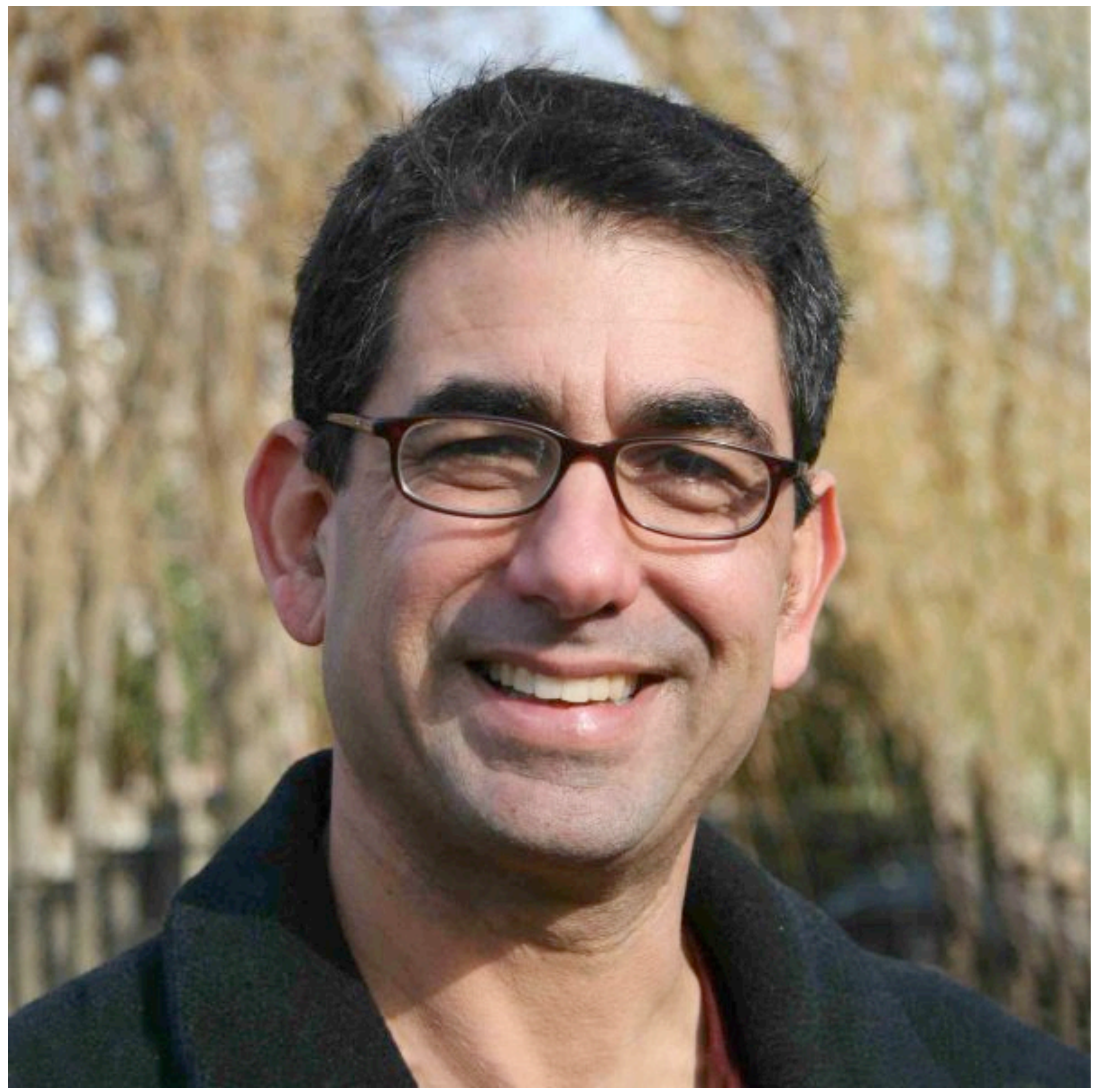
Future Work?


Q3: Covert algorithms in other learning models?
Q2: is there a gap? for natural classes?
Active
TC^0
AC^0
polynomially Fourier concentrated
Locally Covert
Q1
poly-size DNF (or CNF)
poly-size parity DT
poly-size DT
parity
Covert (assuming LPN)
[Canetti-Karchmer]
log(n)-junta
Passive

Future Work?


Q3: Covert algorithms in other learning models?
Q4: Is t-out-of-k covert learning easier for $t < k - 1$ than $t = k - 1$?
⋮
TC^0
AC^0
Q2: is there a gap? for natural classes?
Active
polynomially Fourier concentrated
Q1
Locally Covert
poly-size DNF (or CNF)
poly-size parity DT
poly-size DT
parity
Covert (assuming LPN) [Canetti-Karchmer]
log(n)-junta
Passive

Thanks & Happy Birthday!



ia.cr/2023/392

Thanks & Happy Birthday!

ia.cr/2023/392