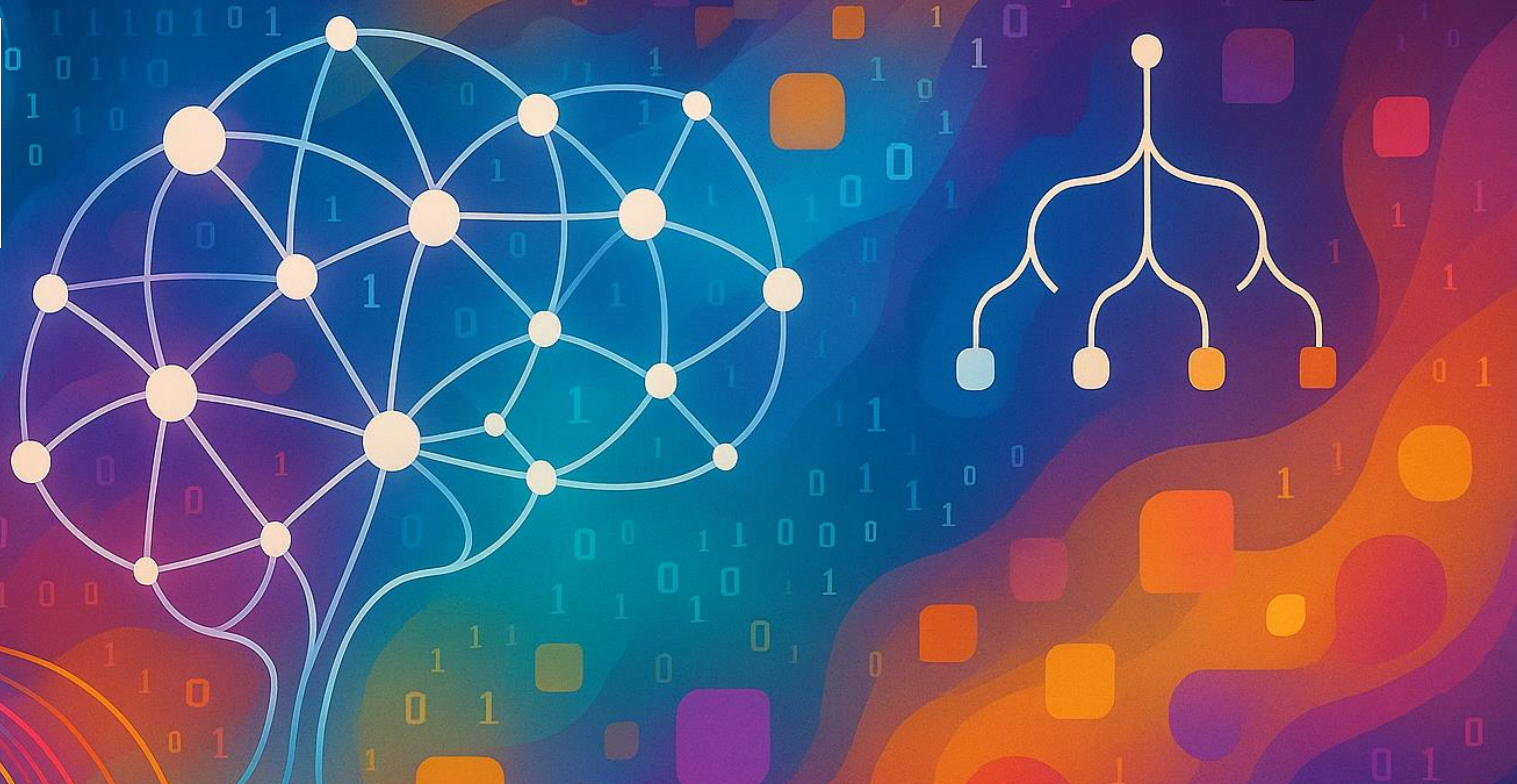


Tutorial on Language Generation

Organizers: *Moses Charikar, Anay Mehrotra, Charlotte Peale, Chirag Pabbaraju, Grigoris Velegkas*

Diverse and Robust Generation

Charlotte Peale
Stanford University

COLT Tutorial on Language Generation – June 30, 2025

This Talk:

Two Extensions of the Language Generation Model

1. Generating with **diversity** constraints

   Representative Language Generation, [CP, Vinod Raman, Omer Reingold]

2. Generating from **noisy data**

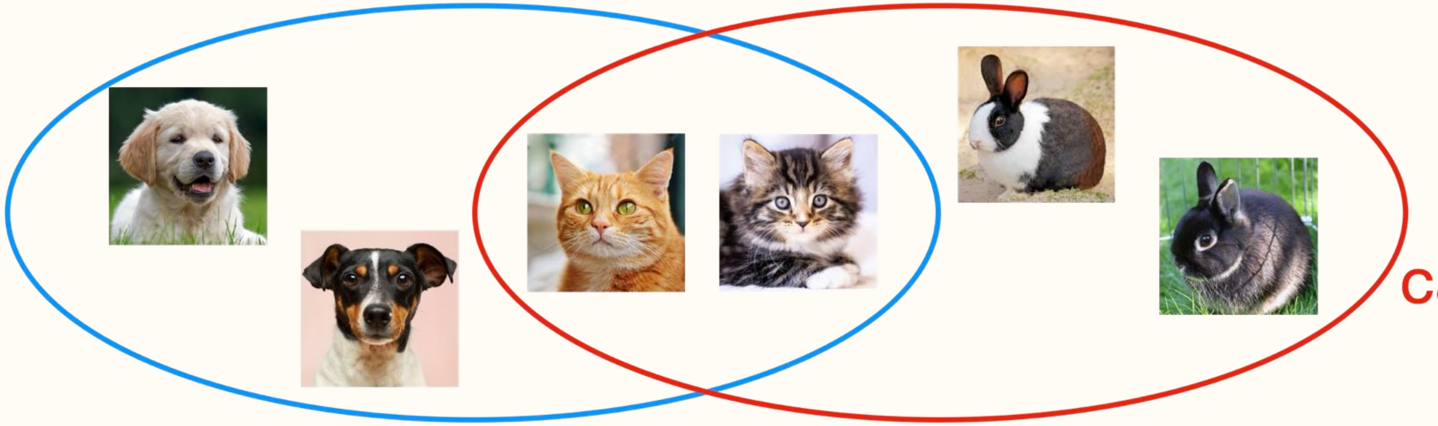
   Generation from Noisy Examples, [Ananth Raman, Vinod Raman]

Representative Generation

CP, Vinod Raman, Omer Reingold (ICML, 2025)

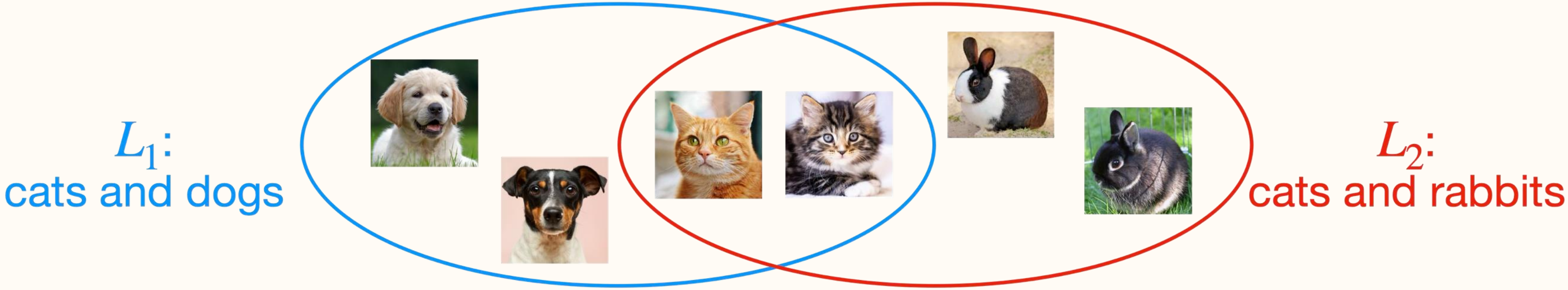
Revisiting Breadth

L_1:
cats and dogs

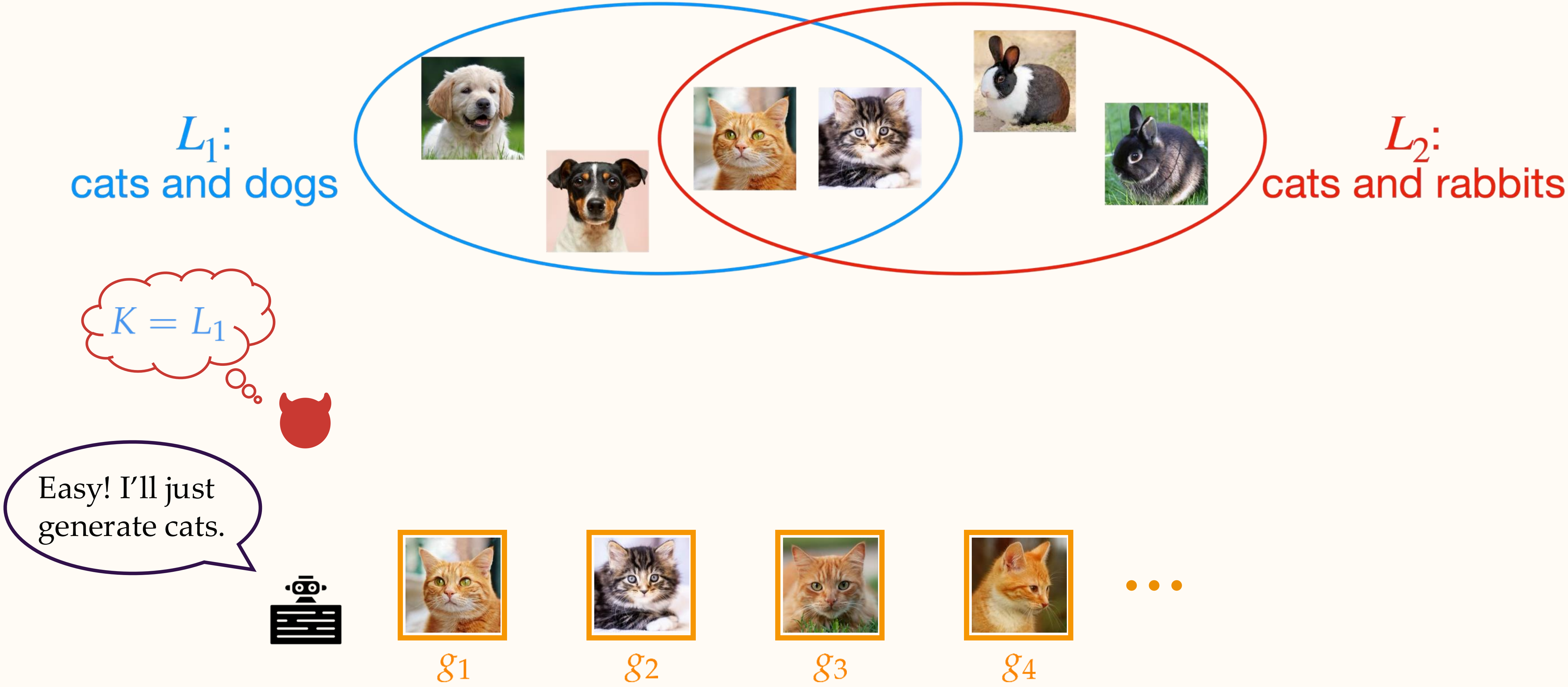


L_2:
cats and rabbits

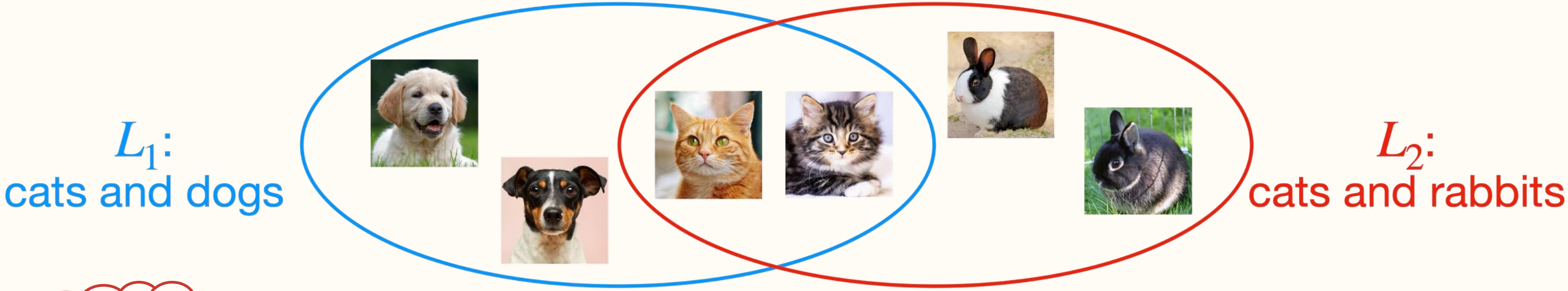
Revisiting Breadth

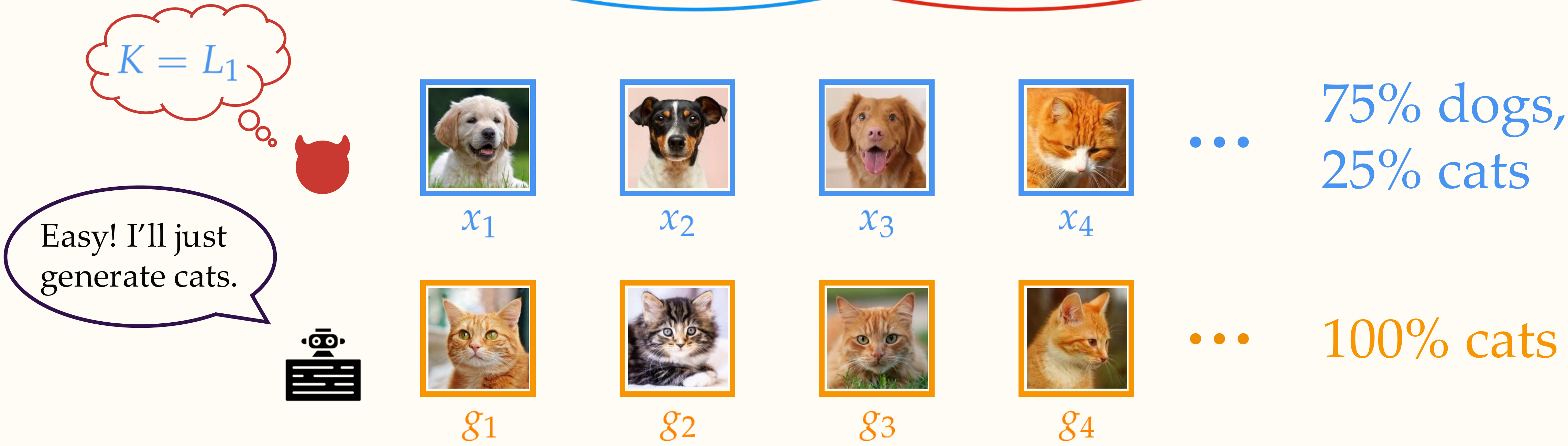


Revisiting Breadth

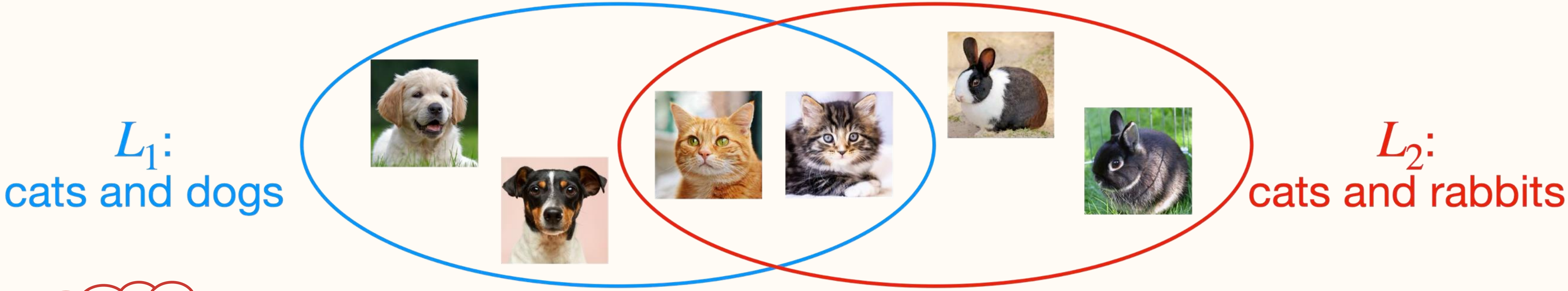


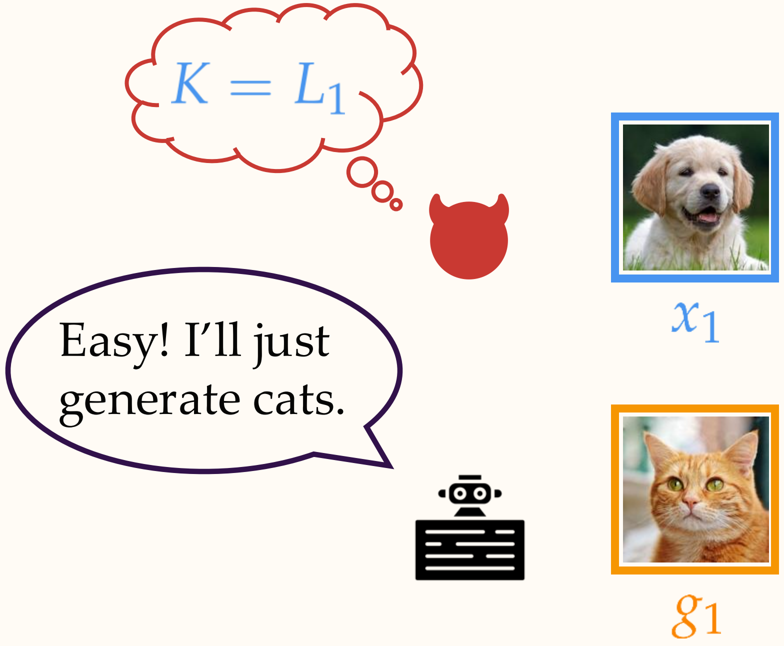
Revisiting Breadth





Revisiting Breadth





Even when generations are consistent, they may not meaningfully resemble the data stream.

Representative Generation

- We require the generator's stream match the data stream wrt *proportions of various subpopulations*.

Representative Generation

- We require the generator's stream match the data stream wrt *proportions of various subpopulations*.
- Randomized generator outputs $\mu_t \in \Delta\mathcal{X}$ at each step.

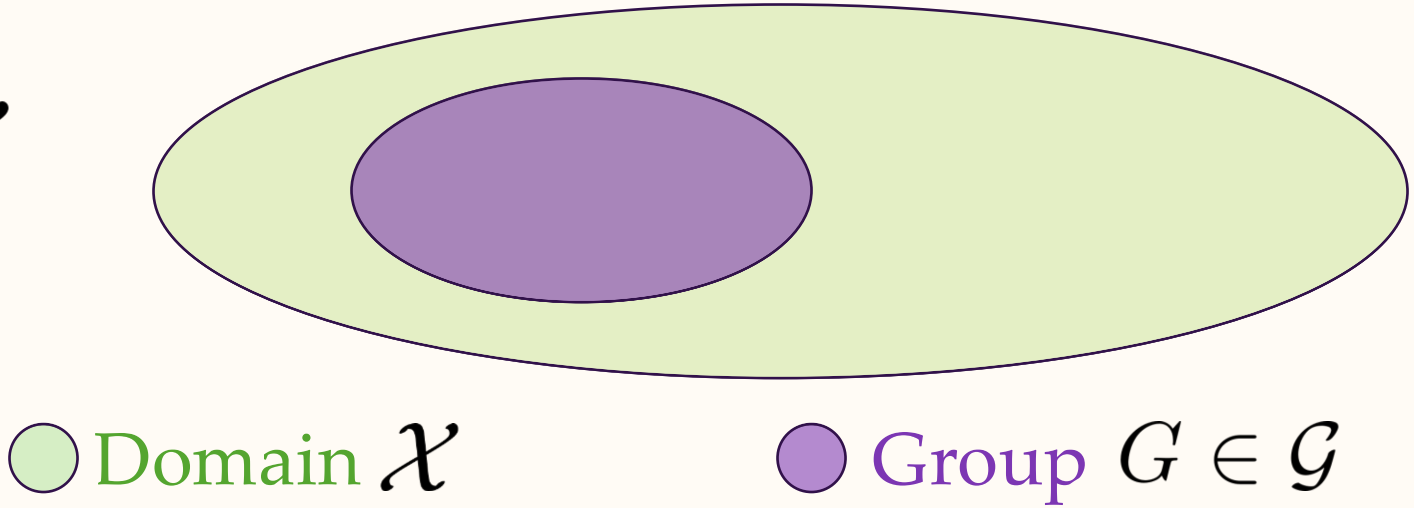
Representative Generation

- We require the generator's stream match the data stream wrt *proportions of various subpopulations*.
- Randomized generator outputs $\mu_t \in \Delta\mathcal{X}$ at each step.
- Introduce an additional constraint given a collection of groups

Representative Generation

- We require the generator's stream match the data stream wrt *proportions of various subpopulations*.
- Randomized generator outputs $\mu_t \in \Delta\mathcal{X}$ at each step.
- Introduce an additional constraint given a collection of groups

$$\mathcal{G} \subseteq 2^{\mathcal{X}}$$



Representative Generation

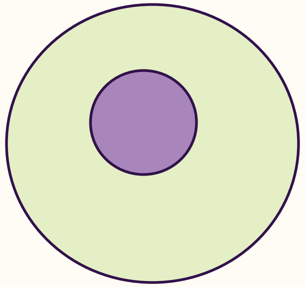
- We require the generator's stream match the data stream wrt *proportions of various subpopulations*.
- Randomized generator outputs $\mu_t \in \Delta\mathcal{X}$ at each step.
- Collection of groups $\mathcal{G} \subseteq 2^{\mathcal{X}}$

For any group $G \in \mathcal{G}$, at every timestep t, μ_t must satisfy:

Representative Generation

- We require the generator's stream match the data stream wrt *proportions of various subpopulations*.
- Randomized generator outputs $\mu_t \in \Delta\mathcal{X}$ at each step.
- Collection of groups $\mathcal{G} \subseteq 2^{\mathcal{X}}$

For any group $G \in \mathcal{G}$, at every timestep t, μ_t must satisfy:

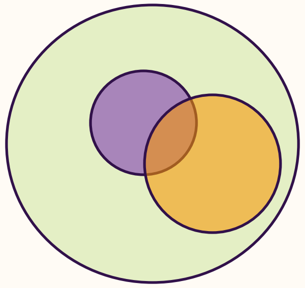


$\mathcal{X}$ Group $G \in \mathcal{G}$

Representative Generation

- We require the generator's stream match the data stream wrt *proportions of various subpopulations*.
- Randomized generator outputs $\mu_t \in \Delta\mathcal{X}$ at each step.
- Collection of groups $\mathcal{G} \subseteq 2^{\mathcal{X}}$

For any group $G \in \mathcal{G}$, at every timestep t, μ_t must satisfy:

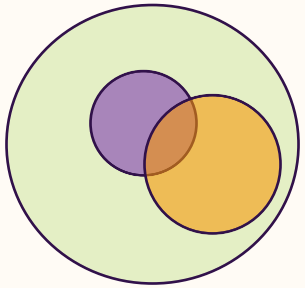


μ_t $\mathcal{X}$ Group $G \in \mathcal{G}$

Representative Generation

- We require the generator's stream match the data stream wrt *proportions of various subpopulations*.
- Randomized generator outputs $\mu_t \in \Delta\mathcal{X}$ at each step.
- Collection of groups $\mathcal{G} \subseteq 2^{\mathcal{X}}$

For any group $G \in \mathcal{G}$, at every timestep t, μ_t must satisfy:



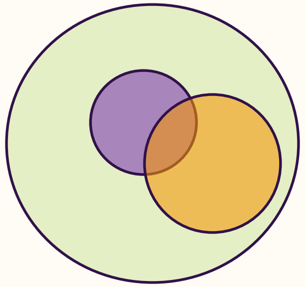
$$\Pr_{x \sim \mu_t} [x \in G]$$

μ_t $\mathcal{X}$ Group $G \in \mathcal{G}$

Representative Generation

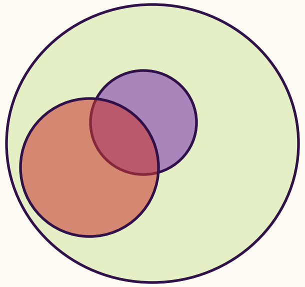
- We require the generator's stream match the data stream wrt *proportions of various subpopulations*.
- Randomized generator outputs $\mu_t \in \Delta\mathcal{X}$ at each step.
- Collection of groups $\mathcal{G} \subseteq 2^{\mathcal{X}}$

For any group $G \in \mathcal{G}$, at every timestep t, μ_t must satisfy:



$$\Pr_{x\sim\mu_t}[x \in G] \approx$$

Proportion of G in adversary stream thus far.

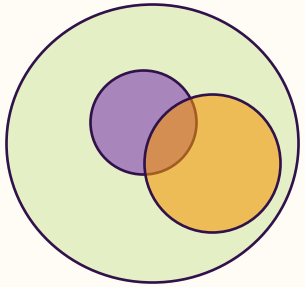


μ_t $\mathcal{X}$ Group $G \in \mathcal{G}$

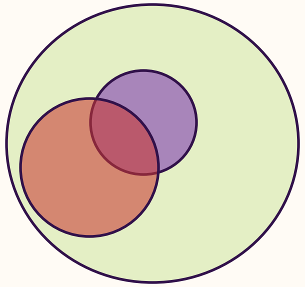
Representative Generation

- We require the generator's stream match the data stream wrt *proportions of various subpopulations*.
- Randomized generator outputs $\mu_t \in \Delta\mathcal{X}$ at each step.
- Collection of groups $\mathcal{G} \subseteq 2^{\mathcal{X}}$

For any group $G \in \mathcal{G}$, at every timestep t, μ_t must satisfy:



$$\Pr_{x\sim\mu_t}[x \in G] \approx \frac{1}{|x_{1:t-1}|}\sum_{x_i \in x_{1:t-1}} \mathbf{1}[x_i \in G]$$



μ_t $\mathcal{X}$ Group $G \in \mathcal{G}$

Representative Generation

- We require the generator's stream match the data stream wrt *proportions of various subpopulations*.

For which group collections and language classes can we representatively generate?

Results

Informal Theorem Any countable language class and countable group collection (subject to a minor assumption) can be representatively generated in the limit.

Results

Informal Theorem Any countable language class and countable group collection (subject to a minor assumption) can be representatively generated in the limit.

But, not possible with only membership queries to languages and groups…

Results

Informal Theorem Any countable language class and countable group collection (subject to a minor assumption) can be representatively generated in the limit.

But, not possible with only membership queries to languages and groups…

Also…

- Characterize representative uniform and non-uniform generatability for finite, disjoint, groups.

Idea Behind Representative Generation Algorithm

Idea Behind Representative Generation Algorithm

Informal Theorem Any countable language class and countable group collection (subject to a minor assumption) can be representatively generated in the limit.

Idea Behind Representative Generation Algorithm

Informal Theorem Any countable language class and countable group collection (subject to a minor assumption) can be representatively generated in the limit.

Approach: Modify critical languages generation in the limit algorithm of [KM'24]

Idea Behind Representative Generation Algorithm

Informal Theorem Any countable language class and countable group collection (subject to a minor assumption) can be representatively generated in the limit.

Approach: Modify critical languages generation in the limit algorithm of [KM'24]

Generate from right-most critical language at each step, but first filter out critical languages that do not allow generator to satisfy representation constraints.

Idea Behind Representative Generation Algorithm

Informal Theorem Any countable language class and countable group collection (subject to a minor assumption) can be representatively generated in the limit.

Approach: Modify critical languages generation in the limit algorithm of [KM'24]

Generate from right-most critical language at each step, but first filter out critical languages that do not allow generator to satisfy representation constraints.

Problem: Even when it becomes critical, the true language isn't guaranteed to allow the generator to pass group tests while playing only from the language support.

Idea Behind Representative Generation Algorithm

Informal Theorem Any countable language class and countable group collection (subject to a minor assumption) can be representatively generated in the limit.

Approach: Modify critical languages generation in the limit algorithm of [KM'24]

Generate from right-most critical language at each step, but first filter out critical languages that do not allow generator to satisfy representation constraints.

Problem: Even when it becomes critical, the true language isn't guaranteed to allow the generator to pass group tests while playing only from the language support.

We show: Eventually, the generator *can* satisfy all group constraints while generating only from the support of the true language. Thus, the true language will not get filtered out!

Future Directions

Future Directions

- Uniform/non-uniform representative generatability: going beyond finite, disjoint, groups.

Future Directions

- Uniform/non-uniform representative generatability: going beyond finite, disjoint, groups.
- Understanding representative generation in the limit beyond countable classes.

Future Directions

- Uniform/non-uniform representative generatability: going beyond finite, disjoint, groups.
- Understanding representative generation in the limit beyond countable classes.
- How do we deal with groups that may change and shift over time?

This Talk:

Two Extensions of the Language Generation Model

1. Generating with **diversity** constraints

   Representative Language Generation, [CP, Vinod Raman, Omer Reingold]

2. Generating from **noisy data**

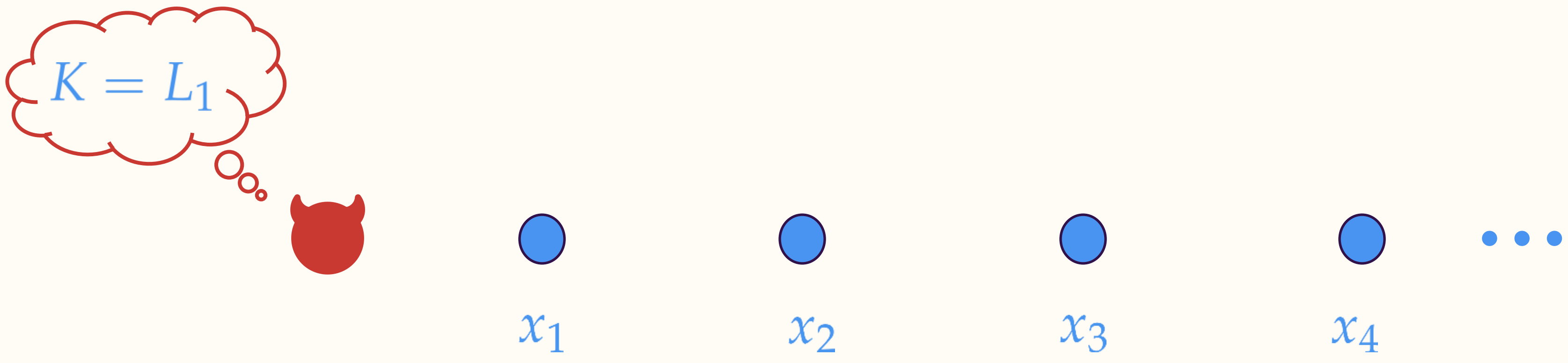
   Generation from Noisy Examples, [Ananth Raman, Vinod Raman]

   a. Model
   b. Results
   c. Future Directions

Generating from Noisy Examples

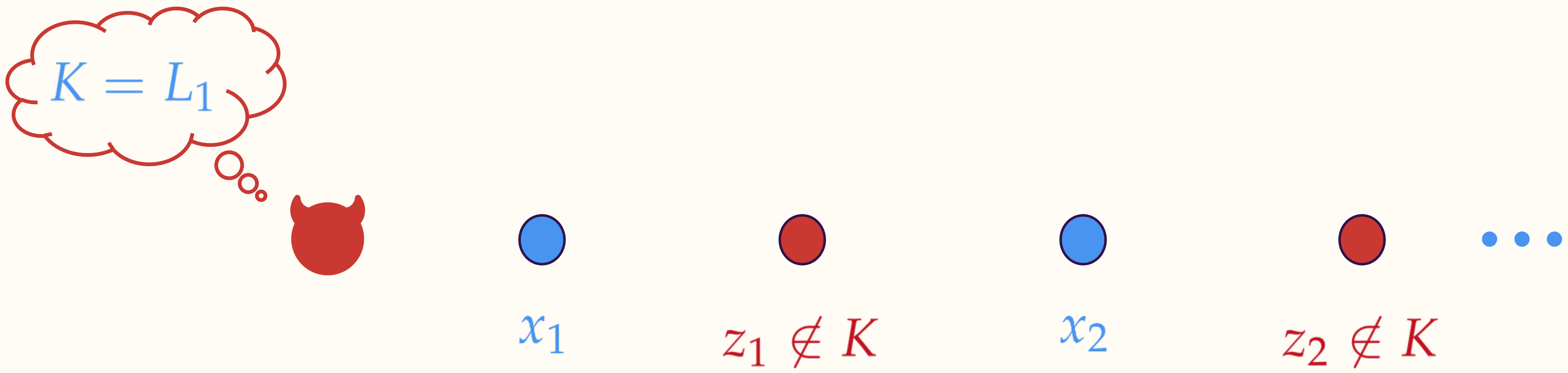
Ananth Raman, Vinod Raman (ICML, 2025)

Model



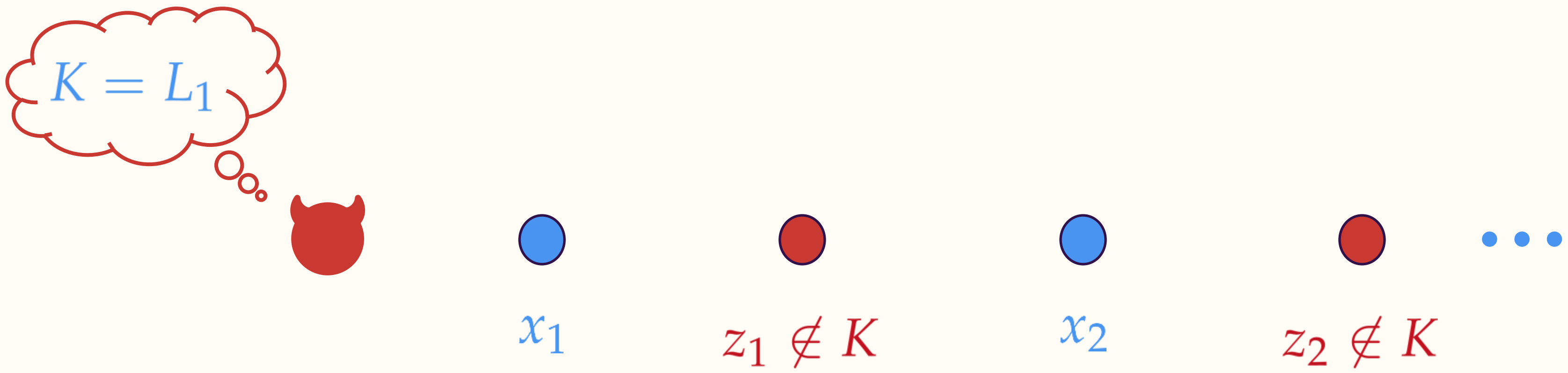
Thus far, adversaries are guaranteed to always output datapoints from the true language.

Model



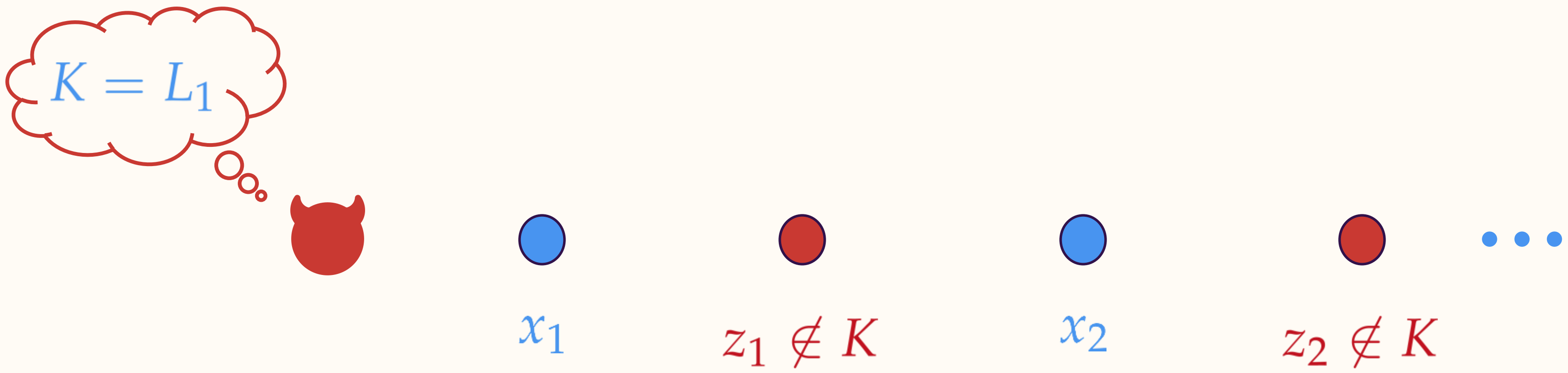
- What if some points are noisy, and not from the true language?

Model



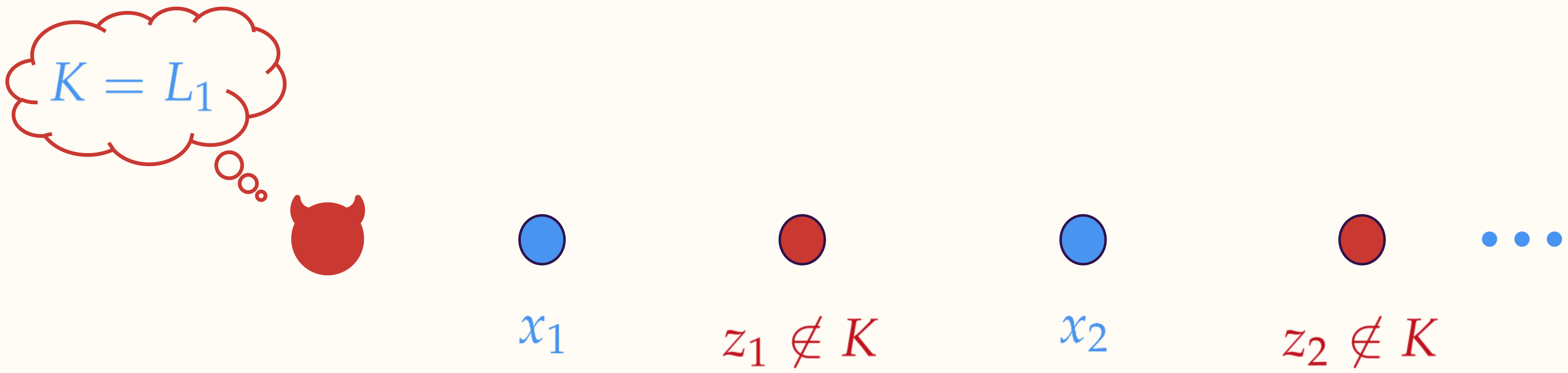
- What if some points are noisy, and not from the true language?
- Important for capturing real-world data settings

Model



- What if some points are noisy, and not from the true language?
- Important for capturing real-world data settings
  - E.g., internet data containing hallucinations of other LLMs

Model



New Generation Setting: We allow the adversary to insert a *finite* number of noisy datapoints into the stream.

Model

1. Adversary picks target $\boldsymbol{K = L_{i^\star}}$ and enumeration $z_1, z_2, \ldots$
2. Adversary picks $n^* \in \mathbb{N}$, and inserts at most n^* negative examples into the enumeration to obtain stream $x_1, x_2, \ldots$.
3. Game continues as normal.

Model

1. Adversary picks target $K = L_{i^\star}$ and enumeration $z_1, z_2, \ldots$
2. Adversary picks $n^* \in \mathbb{N}$, and inserts at most n^* negative examples into enumeration to obtain stream $x_1, x_2, \ldots$.
3. Game continues as normal.

The same noise model studied in works on language *identification* in the limit [Schäfer, 1985; Fulk & Jain, 1989; Baliga et al., 1992; ...]

Model

1. Adversary picks target $K = L_{i^\star}$ and enumeration $z_1, z_2, \ldots$
2. Adversary picks $n^* \in \mathbb{N}$, and inserts at most n^* negative examples into enumeration to obtain stream $x_1, x_2, \ldots$.
3. Game continues as normal.

Noisy Generation in the Limit: Generator wins if all guesses are eventually in K after some $t < \infty$.

Model

1. Adversary picks target $\boldsymbol{K = L_{i^\star}}$ and enumeration $z_1, z_2, \ldots$
2. Adversary picks $n^* \in \mathbb{N}$, and inserts at most n^* negative examples into enumeration to obtain stream $x_1, x_2, \ldots$.
3. Game continues as normal.

Noisy Generation in the Limit: Generator wins if all guesses are eventually in K after some $t < \infty$.

Can also define notions of uniform, non-uniform noisy generatability (see paper for details)

Results

Informal Theorem If a class $\mathcal{L}$ is (noiseless) non-uniformly generatable, then it is noisily generatable in the limit.

Results

Informal Theorem If a class $\mathcal{L}$ is (noiseless) non-uniformly generatable, then it is noisily generatable in the limit.

Also…

- Characterize noisy uniform generatability.

Results

Informal Theorem If a class $\mathcal{L}$ is (noiseless) non-uniformly generatable, then it is noisily generatable in the limit.

Also…

- Characterize noisy uniform generatability.
- Provide a sufficient condition for noisy non-uniform generatability.

Results

Informal Theorem If a class $\mathcal{L}$ is (noiseless) non-uniformly generatable, then it is noisily generatable in the limit.

Also…

- Characterize noisy uniform generatability.
- Provide a sufficient condition for noisy non-uniform generatability.

Taste of the Theorem Proof

Informal Theorem If a class $\mathcal{L}$ is (noiseless) non-uniformly generatable, then it is noisily generatable in the limit.

Goal: Use a blackbox noiseless non-uniform generator to generate in the limit with noise.

$$Q(x_1, x_2, \ldots, x_{t-1}) \to g_t$$

Non-uniform generator

In true language after d^* unique elements

Taste of the Theorem Proof

Informal Theorem If a class $\mathcal{L}$ is (noiseless) non-uniformly generatable, then it is noisily generatable in the limit.

Goal: Use a blackbox noiseless non-uniform generator to generate in the limit with noise.

$$Q(x_1, x_2, \ldots, x_{t-1}) \to g_t$$

Non-uniform generator

~~In true language after d^* unique elements~~

Problem: guarantee breaks down when data stream corrupted.

Taste of the Theorem Proof

Informal Theorem If a class $\mathcal{L}$ is (noiseless) non-uniformly generatable, then it is noisily generatable in the limit.

Goal: Use a blackbox noiseless non-uniform generator to generate in the limit with noise.

Taste of the Theorem Proof

Informal Theorem If a class $\mathcal{L}$ is (noiseless) non-uniformly generatable, then it is noisily generatable in the limit.

Goal: Use a blackbox noiseless non-uniform generator to generate in the limit with noise.

Idea: Apply Q on a *sliding window* of the data stream.

Taste of the Theorem Proof

Informal Theorem If a class $\mathcal{L}$ is (noiseless) non-uniformly generatable, then it is noisily generatable in the limit.

Goal: Use a blackbox noiseless non-uniform generator to generate in the limit with noise.

Idea: Apply Q on a *sliding window* of the data stream.

$$Q(x_{r_t}, x_{r_t+1}, \ldots, x_{t-1}) \to g_t$$

Taste of the Theorem Proof

Informal Theorem If a class $\mathcal{L}$ is (noiseless) non-uniformly generatable, then it is noisily generatable in the limit.

Goal: Use a blackbox noiseless non-uniform generator to generate in the limit with noise.

Idea: Apply Q on a *sliding window* of the data stream.

$$Q(x_{r_t}, x_{r_t+1}, \ldots, x_{t-1}) \to g_t$$

As long as the number of unique elements and lower threshold of the window continue to grow, at some point the input to Q will always be clean data.

Taste of the Theorem Proof

Informal Theorem If a class $\mathcal{L}$ is (noiseless) non-uniformly generatable, then it is noisily generatable in the limit.

Goal: Use a blackbox noiseless non-uniform generator to generate in the limit with noise.

Idea: Apply Q on a *sliding window* of the data stream.

$$Q(x_{r_t}, x_{r_t+1}, \ldots, x_{t-1}) \to g_t$$

As long as the number of unique elements and lower threshold of the window continue to grow, at some point the input to Q will always be clean data.

With a few tweaks to make sure no duplicates are generated, this approach generates in the limit!

Future Directions

Future Directions

- What are complete characterizations of noisy non-uniform generation and generation in the limit?

Future Directions

- What are complete characterizations of noisy non-uniform generation and generation in the limit?
- Can noisy generation in the limit be achieved with only membership oracle access? Cf. [KM'24]

Future Directions

- What are complete characterizations of noisy non-uniform generation and generation in the limit?
- Can noisy generation in the limit be achieved with only membership oracle access? Cf. [KM'24]
- Is there a reasonable definition of agnostic generatability for language generation?