CS11-711 Advanced NLP

Ensembling and Mixture of Experts

Graham Neubig

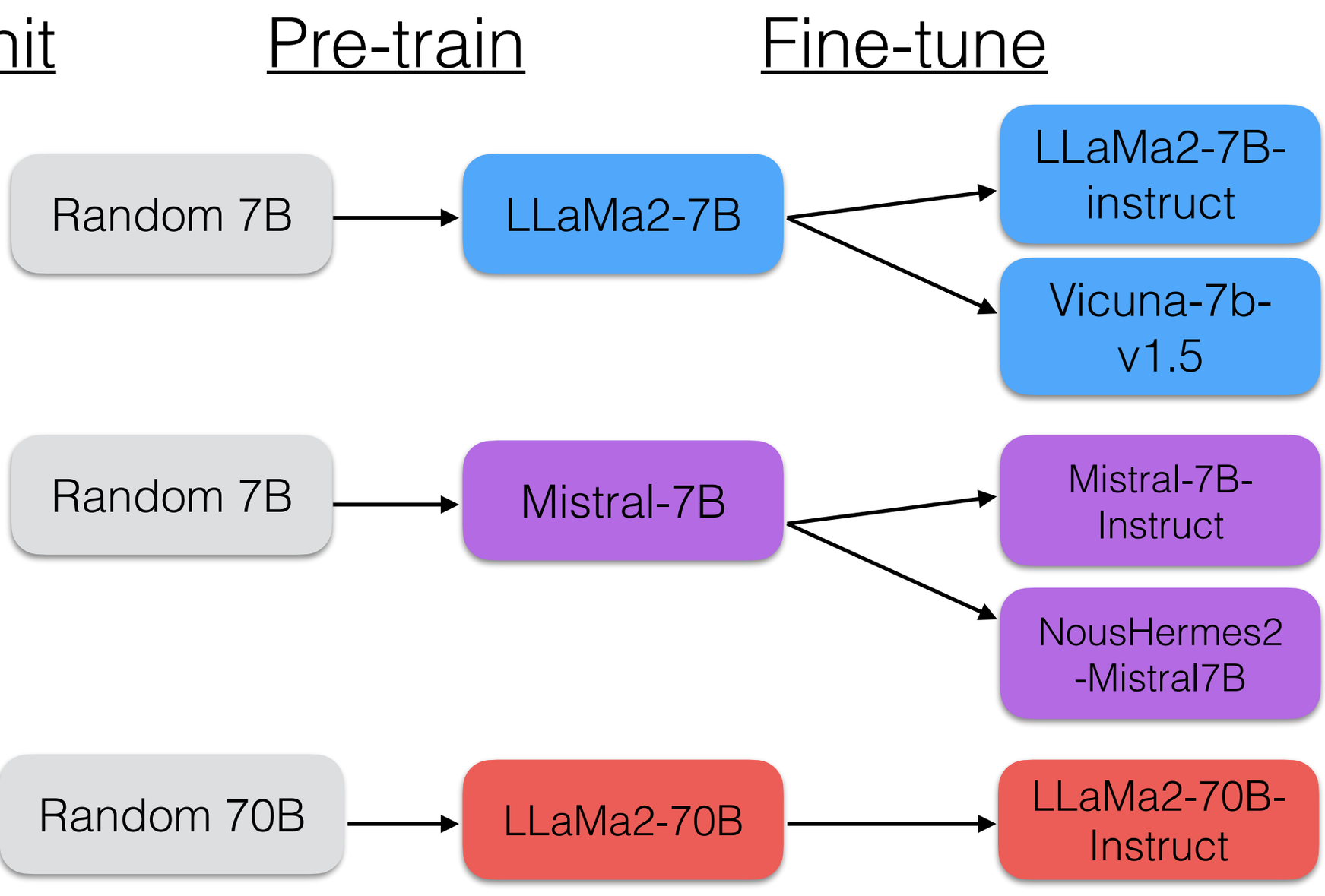
Carnegie Mellon University

Language Technologies Institute

https://phontron.com/class/anlp-fall2024/

Many Models Exist!

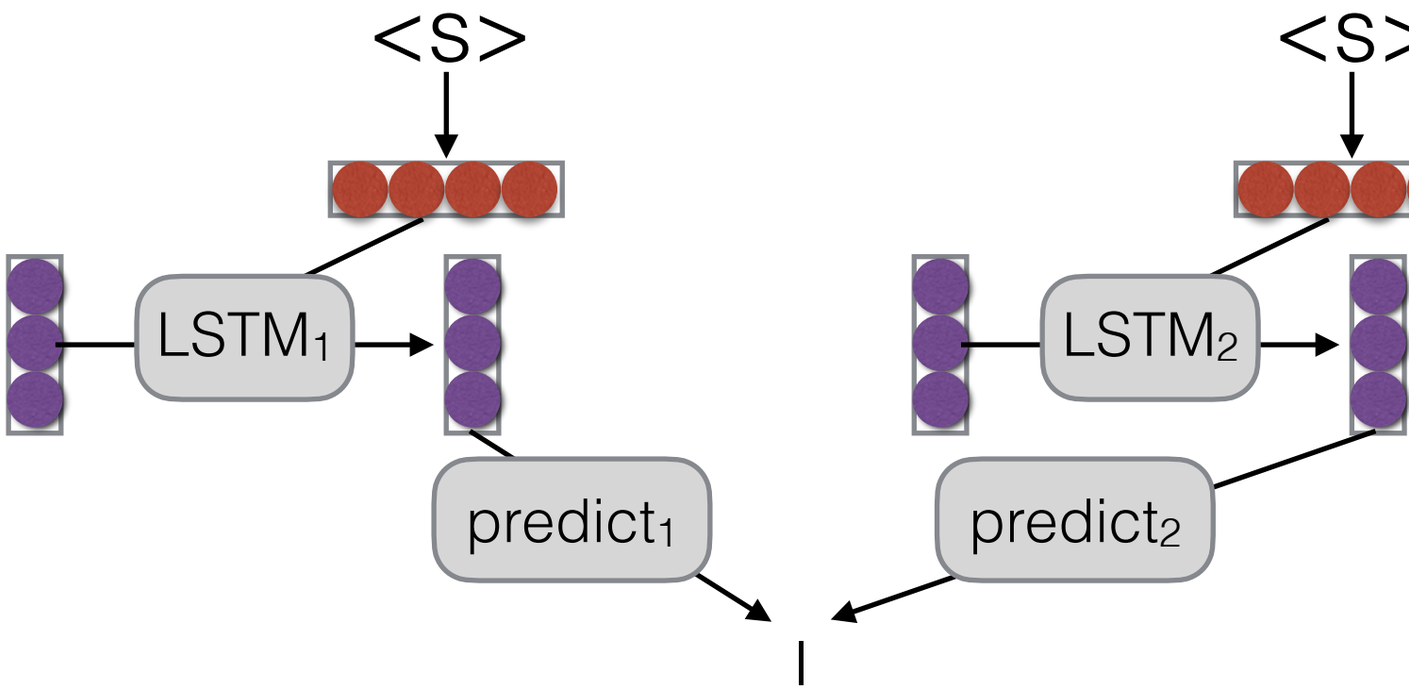
- Different architectures or training result in different P(Y|X)



Model Ensembling

Ensembling

- Combine predictions from multiple models



- Why?
  - Multiple models make somewhat uncorrelated errors
  - Models tend to be more uncertain when they are about to make errors
  - Smooths over idiosyncrasies of the model

Linear Interpolation

- Take a weighted average of the M model probabilities

$$P(y_j \mid X, y_1, \ldots, y_{j-1}) = \sum_{m=1}^{M} \underbrace{P_m(y_j \mid X, y_1, \ldots, y_{j-1})}_{\text{Probability according to model } m} \underbrace{P(m \mid X, y_1, \ldots, y_{j-1})}_{\text{Probability of model } m}$$

- Second term often set to a constant, independent of context

Log-linear Interpolation

- Weighted combination of log probabilities, normalize

$$P(y_j \mid X, y_1, \ldots, y_{j-1}) =$$

$$\text{softmax}\left(\sum_{m=1}^{M} \lambda_m(X, y_1, \ldots, y_{j-1}) \log P_m(y_j \mid X, y_1, \ldots, y_{j-1})\right)$$

Normalize

Interpolation coefficient for model *m*

Log probability of model *m*

- Interpolation coefficient often set to a constant

Linear or Log Linear?

- Think of it in logic!
- **Linear:** “Logical OR”
  - the interpolated model likes any choice that a model gives a high probability
  - use models with models that capture different traits
  - necessary when any model can assign zero probability
- **Log Linear:** “Logical AND”
  - interpolated model only likes choices where all models agree
  - use when you want to restrict possible answers

Using Models as Negative Evidence

- It is also possible to use models as *negative* evidence that you want to remove

$$\text{logit} = \underbrace{\log P_{\text{core}}(y_t|X, y_{<t})}_{\text{Core}} + \lambda(\underbrace{\log P_{+}(y_t|X, y_{<t})}_{\text{Positive}} - \underbrace{\log P_{-}(y_t|X, y_{<t})}_{\text{Negative}})$$

- e.g. Domain differential adaptation (Dou et al. 2019)
  - core = MT model, negative = out-of-domain LM, positive = in-domain LM
- e.g. DExperts (Liu et al. 2021)
  - core = strong LM, negative = weak toxic LM, positive = weak nontoxic LM
- (This is just a special case of log-linear interpolation)

Efficient Methods for Using Multiple Models

Problem with Ensembling: Cost

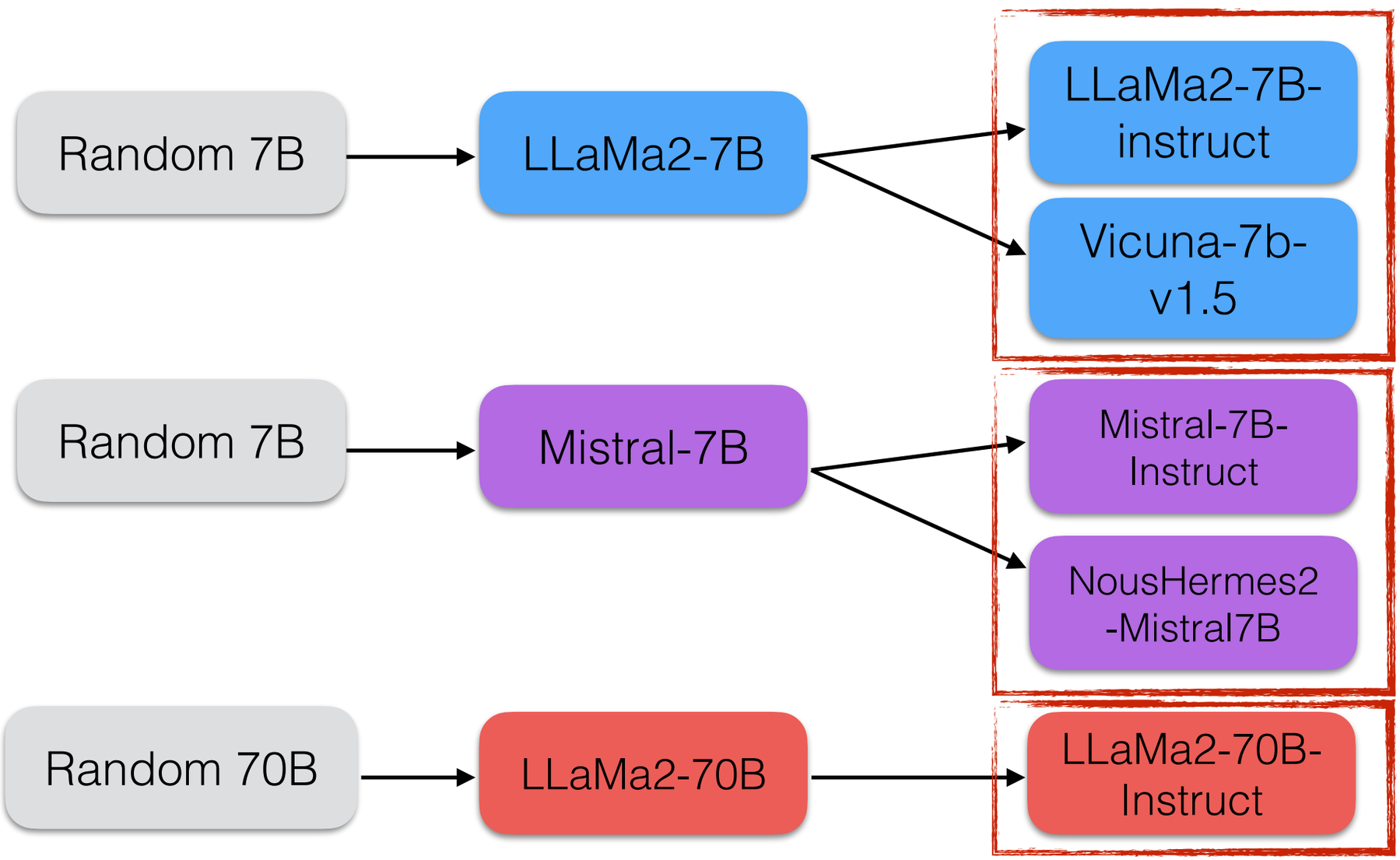
- Simple ensembling is expensive: it requires running multiple models in parallel
  - N times the computation
  - N times the memory (!)
- Is there any way we can more easily combine together two models?

Parameter Averaging
(e.g. Utans 1996)

- Parameter averaging is a cheap way to get some good effects of ensembling
- Basically, average the parameters of multiple models
  - **Checkpoint averaging:** write out models several times near the end of training, and take the average of parameters
  - **Fine-tuned model merging:** fine tune in several different ways, then average

Can only Average Related Models

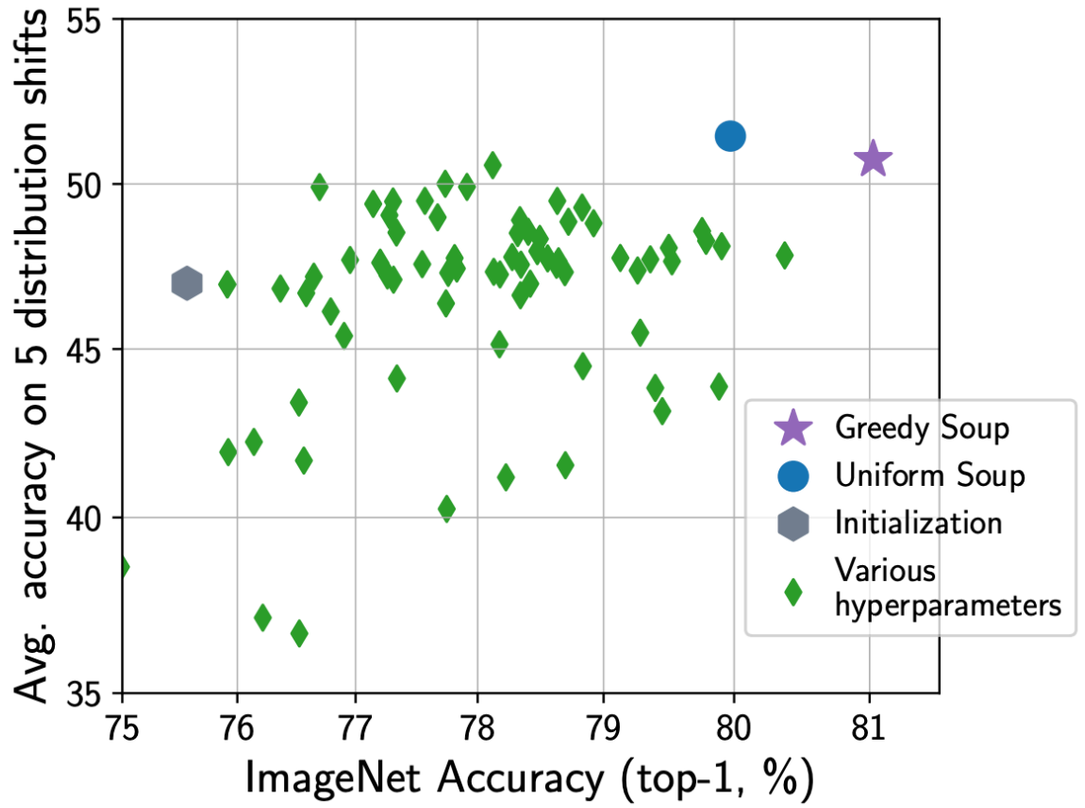
- Models must originate from the same pre-trained checkpoint



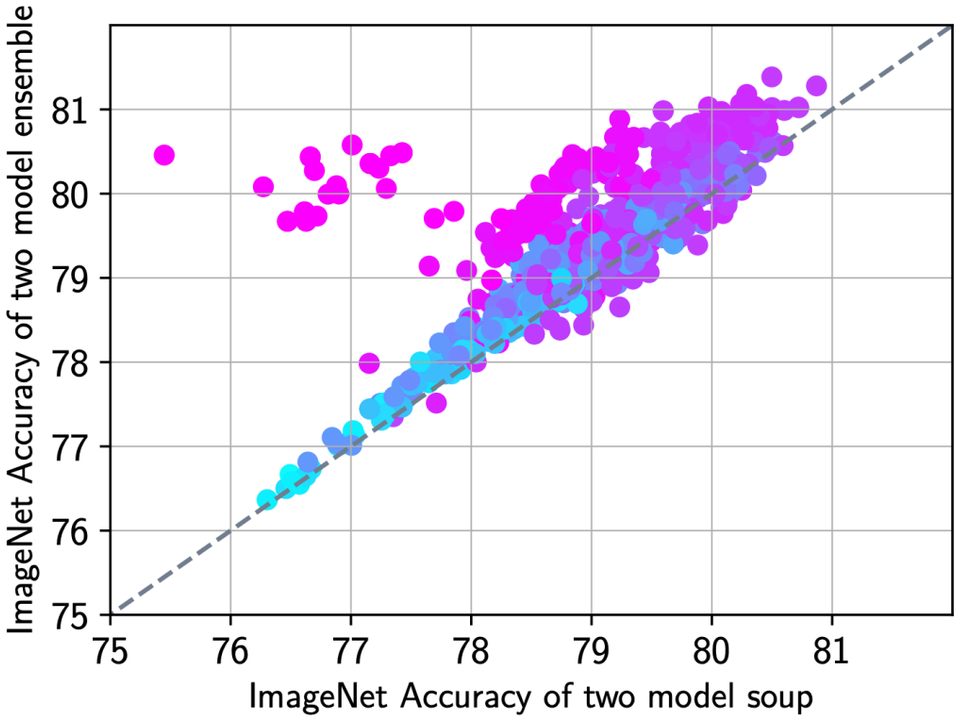
- **Quiz:** why is this?

Model Soups (Wortsman et al. 2022)

- Examines two strategies:
  - Uniform averaging
  - Greedy averaging (add one, and keep if it improves)

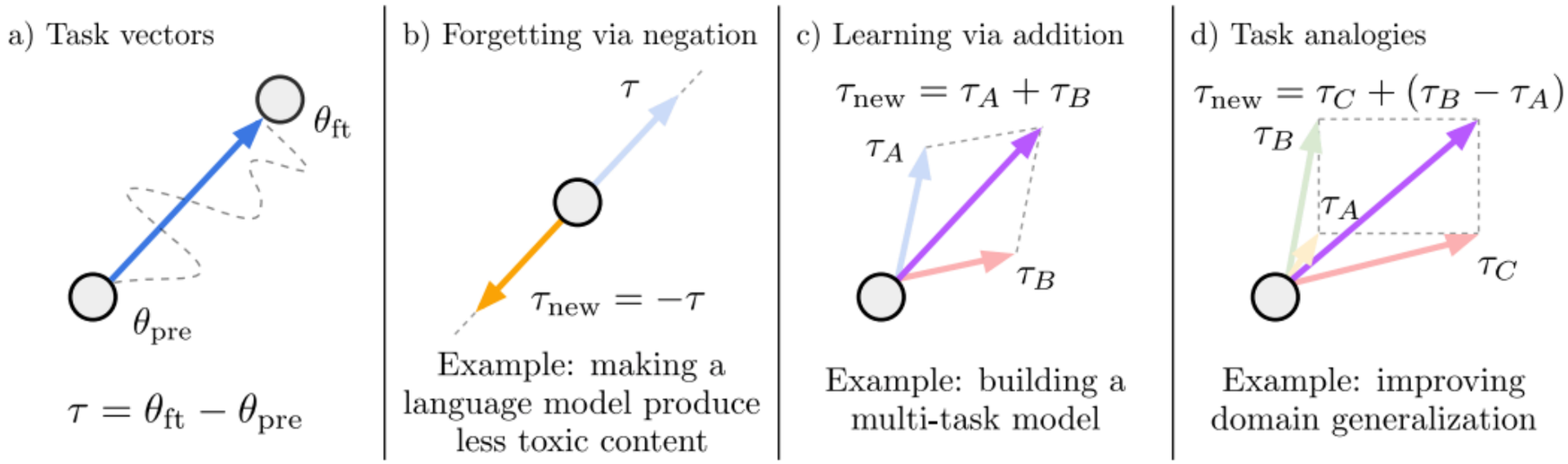


- Demonstrates that averaging is correlated with resembling

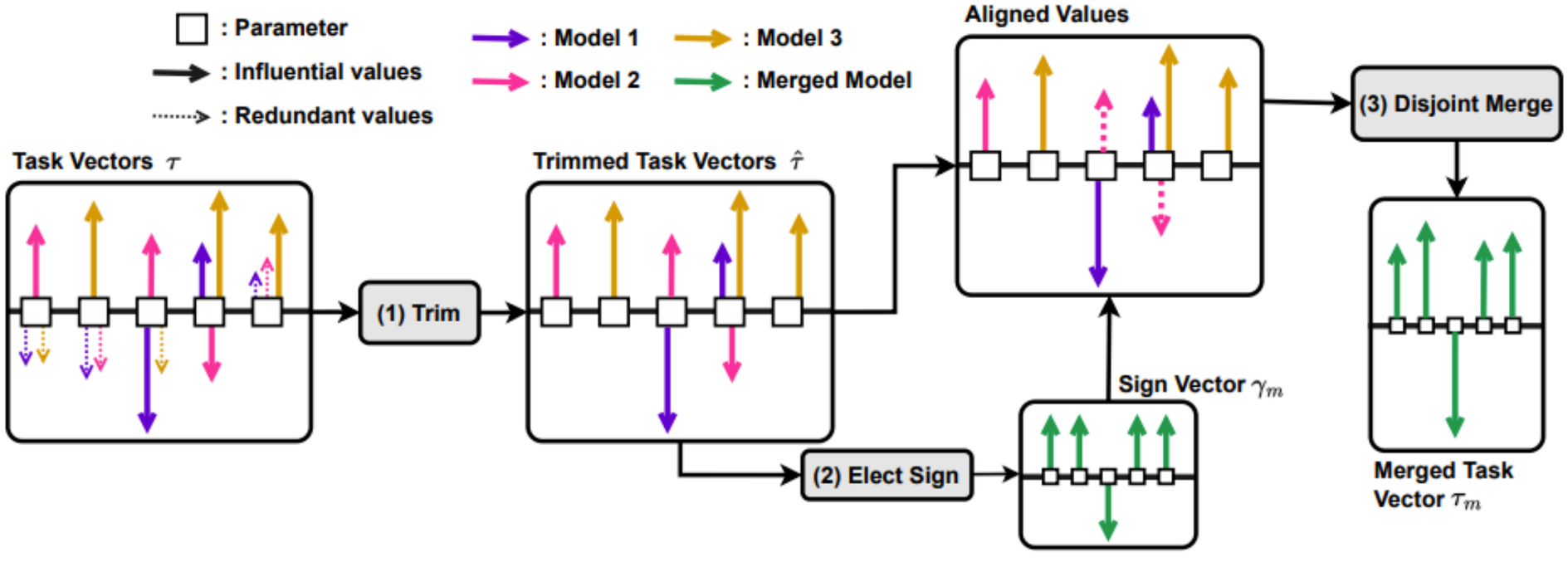


Task Vectors

- Quantify changes from a base models through “task vectors” (Ilharco et al. 2022)



- TIES: resolves conflicts through max and sign (Yadav et al. 2023)



Software: mergekit

- https://github.com/arcee-ai/mergekit
- Implements a number of different methods for model merging

Method	`merge_method` value	Multi-Model	Uses base model
Linear (Model Soups)	`linear`	✅	❌
SLERP	`slerp`	❌	✅
Task Arithmetic	`task_arithmetic`	✅	✅
TIES	`ties`	✅	✅
DARE TIES	`dare_ties`	✅	✅
DARE Task Arithmetic	`dare_linear`	✅	✅
Passthrough	`passthrough`	❌	❌

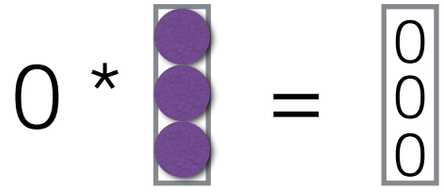
Ensemble Distillation (e.g. Hinton et al. 2015)

- **Problem:** parameter averaging only works for models within the same run
- Knowledge distillation trains a model to **copy the ensemble**
  - Specifically, it tries to match the distribution over predicted words
  - Why? We want the model to make the same mistakes as an ensemble
- Shown to increase accuracy notably

Sparse Mixture of Experts Models

Sparse Computation

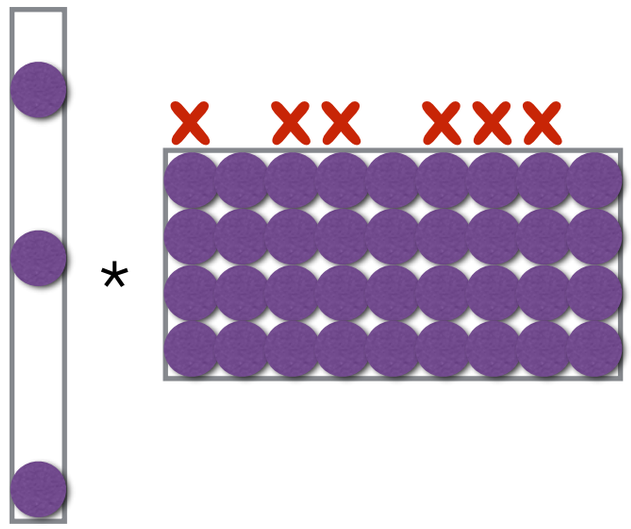
- What happens when a scalar-tensor multiplication is zero?
  - Result is guaranteed to be zero! No computation needed



- This can happen in many parts of a model:
  - Single rows in a matrix multiply → optimized by GPU
  - Larger tensors → sparse MoE models
  - Whole models in an ensemble → just don't use that model

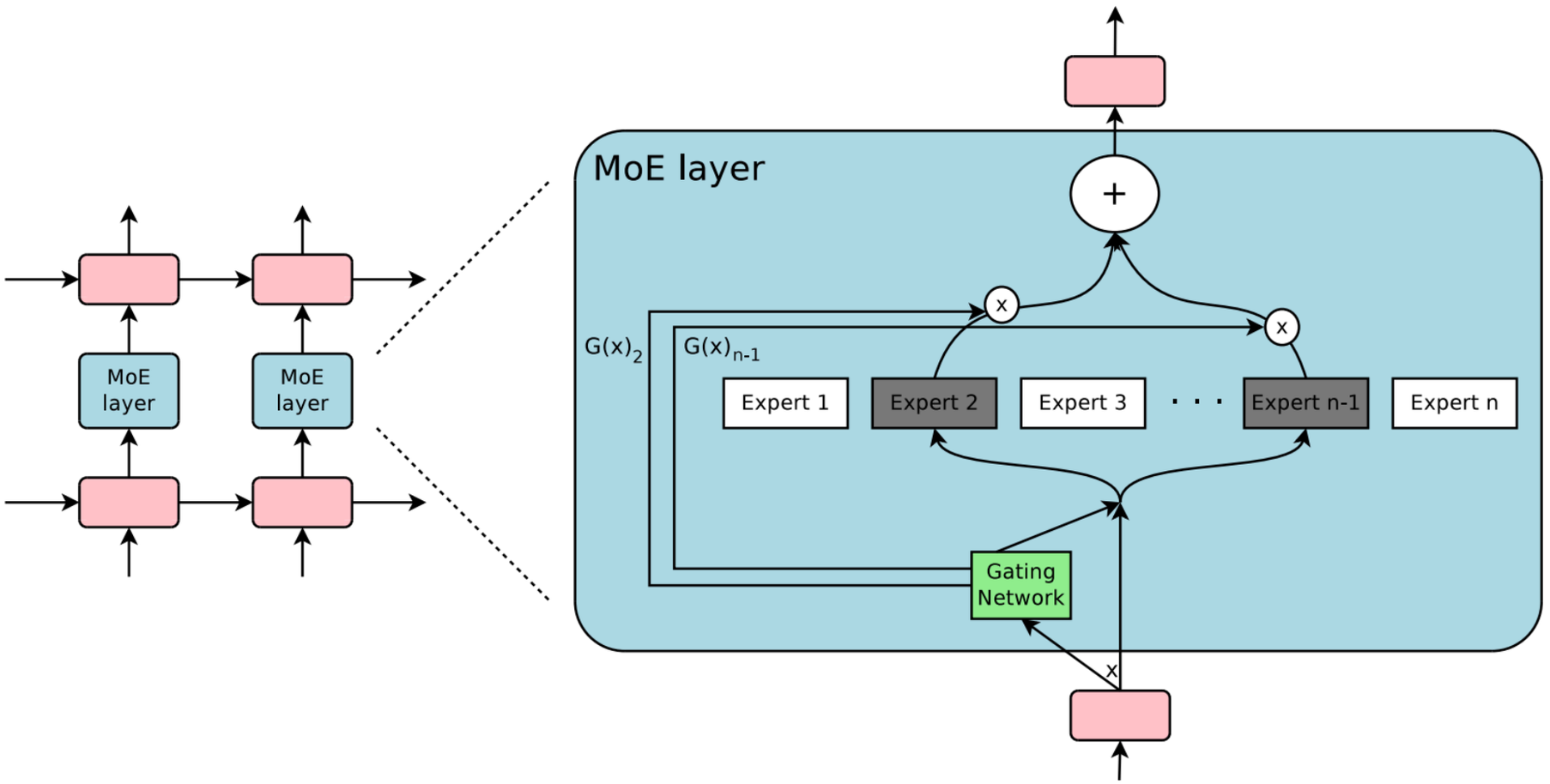
GPU-level Sparsity

- NVIDIA GPUs support various types of sparsity through the cuSPARSE library and tensor cores
- Examples, vector-matrix multiply with sparse vector (e.g. one that comes from ReLU activation)



Sparsely Gated Mixture of Experts Layer (Shazeer+ 2017)

- Select a subset of FFNs to actually execute

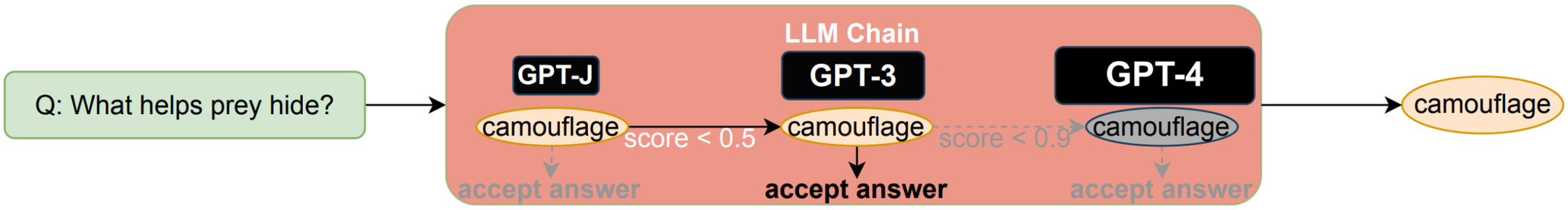


$$g(x) = \text{softmax}(\text{keep\_top\_k}(f_{\text{gating}}(x), k))$$

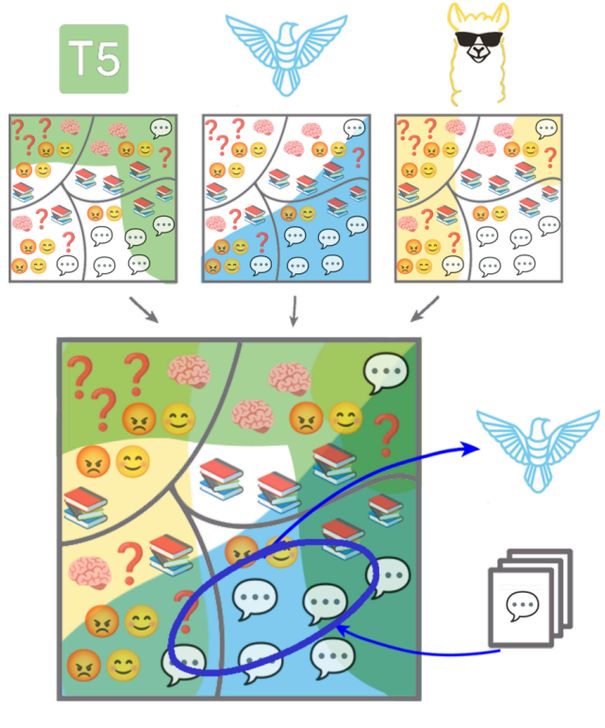
$$\text{keep\_top\_k}(v, k)_i = \begin{cases} v_i & \text{if } v_i \text{ is in the top } k \text{ elements of } v. \\ -\infty & \text{otherwise.} \end{cases}$$

Model Selection

- An alternative: select which of multiple models to use
- **Model cascades:** try increasingly acceptable models until one reaches acceptable accuracy (Chen et al. 2023)



- **Model routing:** try to choose the most appropriate model a-priori (e.g. Schnitzer et al. 2023)



Pipeline Systems

Cascaded Systems

- In many cases, we hook the input of one system to the output of another systems

$$\hat{Y}_1 = \underset{Y}{\operatorname{argmax}} P(Y|X;\theta_1)$$

$$\hat{Y}_2 = \underset{Y}{\operatorname{argmax}} P(Y|Y_1;\theta_2)$$

- Example: speech translation:
  - speech -> ASR -> text
  - text -> MT -> text in another language
- Why?
  - Data availability
  - Interpretability

Stacking

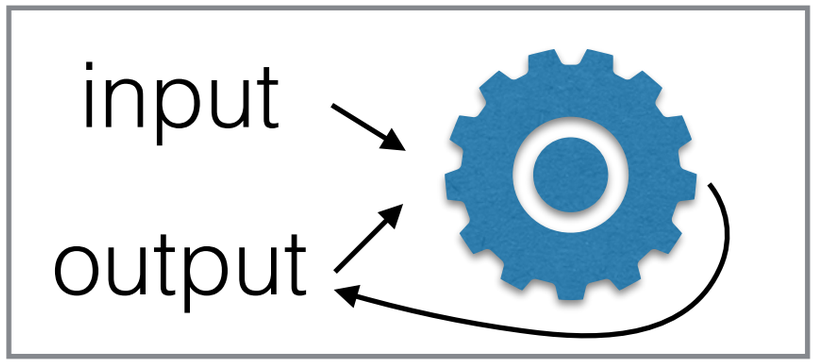
- What if we have two very different models for the *same* task but predictions done in different ways
- e.g. a phrase-based translation model and a neural MT model (Niehues et al. 2017)
- Stacking uses the **output of one system in calculating features for another** system

$$\hat{Y}_1 = \underset{Y}{\operatorname{argmax}} P(Y|X;\theta_1)$$

$$\hat{Y}_2 = \underset{Y}{\operatorname{argmax}} P(Y|X, Y_1;\theta_2)$$

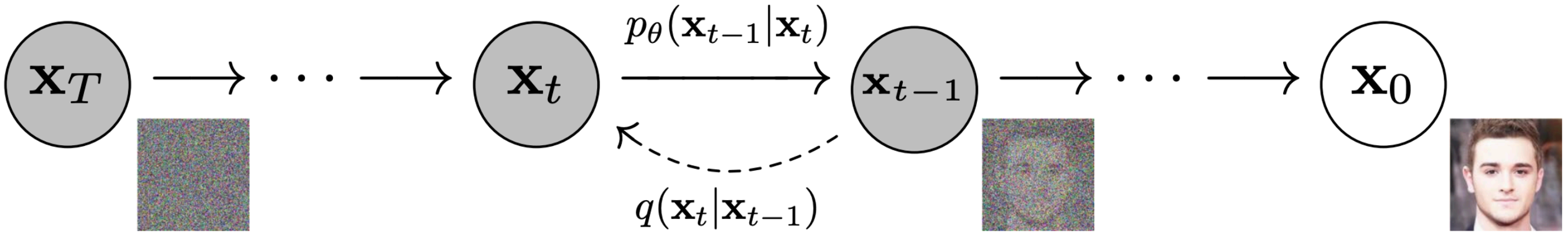
Iterative Refinement

- Like cascade/stacking, but done multiple times with the same model



Diffusion: Basic Idea
(Ho et al. 2020)

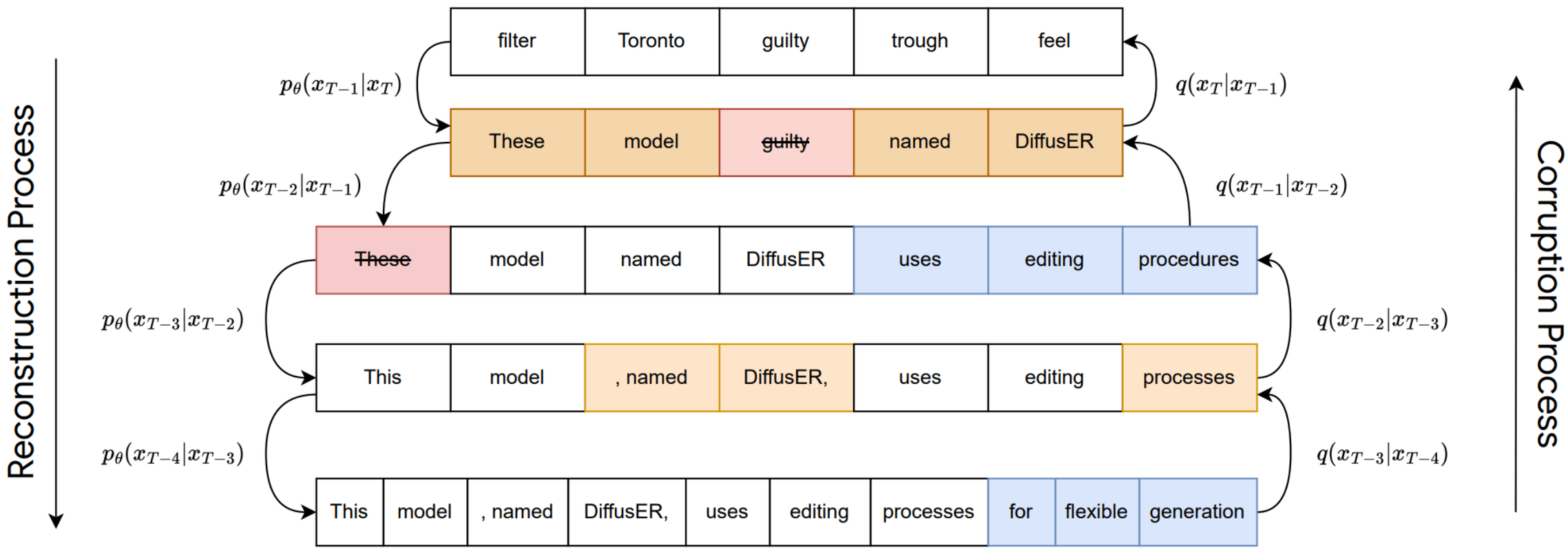
- Add noise to the data *x* using a noising process *q*
- Learn a model *p* to denoise



- For images, noise can be, e.g. Gaussian noise

Diffusion: Applications to Language

- How do we define noising/reconstruction
- *Simple (Austin et al. 2021):* replace individual words
- *More effective (Reid et al. 2022):* noise using insert/delete/replace operations, and reconstruct with editing model



Self-Refinement

(Madaan et al. 2023)

- After generating output, have LM critique and improve

(a) **Dialogue:** x, $\mathbf{y_t}$

```
User: I am interested
in playing Table
tennis.

Response: I'm sure
it's a great way to
socialize, stay active
```

(b) **FEEDBACK** **fb**

```
Engaging: Provides no
information about table
tennis or how to play it.

User understanding: Lacks
understanding of user's
needs and state of mind.
```

(c) **REFINE** $\mathbf{y_{t+1}}$

```
Response (refined): That's
great to hear (...) ! It's
a fun sport requiring
quick reflexes and good
hand-eye coordination.
Have you played before, or
are you looking to learn?
```

(d) **Code optimization:** x, $\mathbf{y_t}$

```
Generate sum of 1, ..., N
def sum(n):
    res = 0
    for i in range(n+1):
        res += i
    return res
```

(e) **FEEDBACK** **fb**

```
This code is slow as
it uses brute force.
A better approach is
to use the formula
... (n(n+1))/2.
```

(f) **REFINE** $\mathbf{y_{t+1}}$

```
Code (refined)

def sum_faster(n):
  return (n*(n+1))//2
```

Questions?