# Self-Supervised Fine-Tuning

| Input: | | Decode: | | Label: |
|---|---|---|---|---|
| CS165 | → | | vs | is |
| is | → | | vs | an |
| an | → | | vs | awesome |
| awesome | → | | vs | class |
| class | → | | vs | EOS |

**loss function**

## Likelihood

$$p(\text{is, an}, \ldots, \text{EOS}) = p(x_1, x_2, \ldots, x_N) = \prod_{i=1}^{N} p(x_i \mid x_1, \ldots, x_{i-1})$$

## Cross-Entropy

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \log p(x_i)$$

## Perplexity

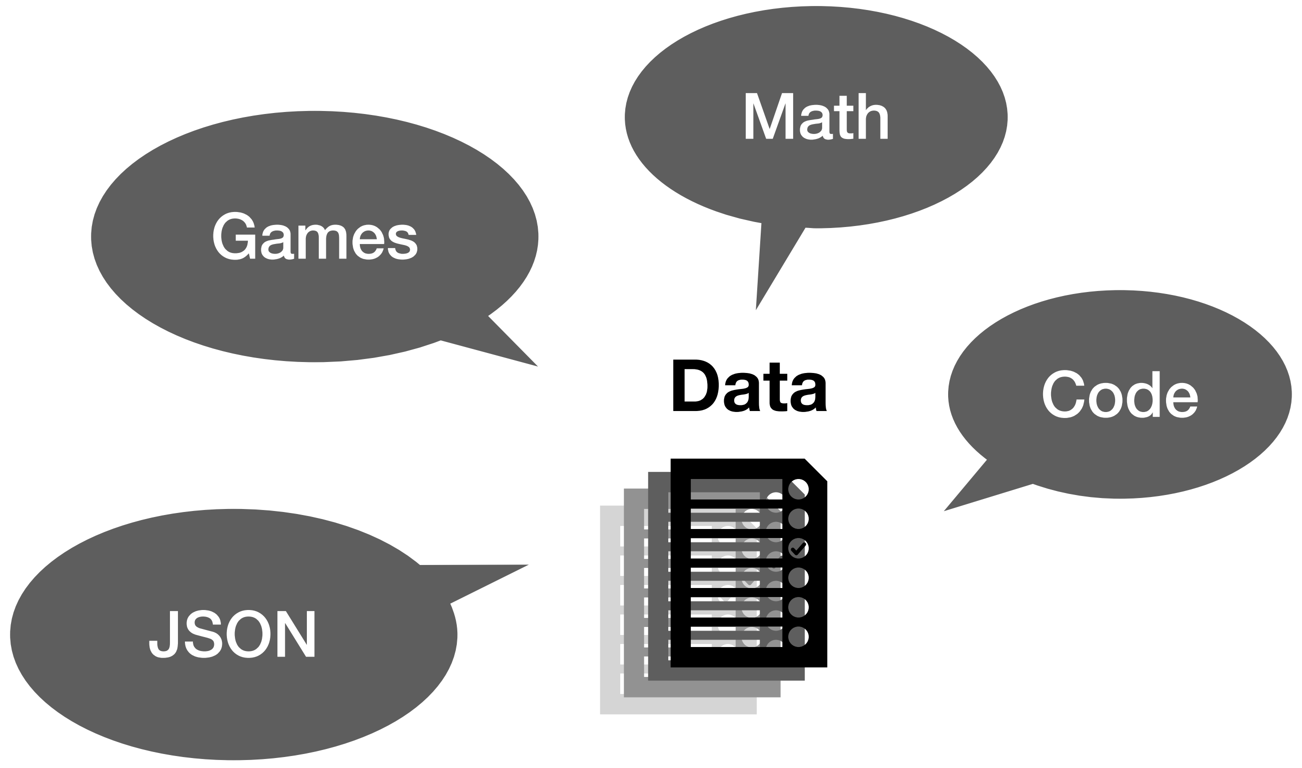
$$e^{\mathcal{L}} = \exp\left(-\frac{1}{N} \sum_{i=1}^{N} \log p(x_i)\right)$$

pre-training


Supervised Learning
Reinforcement Learning
Learning Paradigms
Objective ?
distance minimization
reward maximization
Labels from…
Reward from…
internal structure of data
external annotations
environment
human preferences
Self-Supervised Fine-Tuning
pre-training
Supervised Fine-Tuning (SFT)
Reinforcement Fine-Tuning (RFT)
Reinforcement Learning with Human Feedback (RLHF)
DASlab
@ Harvard SEAS


DASlab @ Harvard SEAS

Self-Supervised Fine-Tuning vs Supervised Fine-Tuning (SFT)




Decoder Architecture

## Causal Language Modeling

(predict next token left→right)

| | | | | | |
|---|---|---|---|---|---|
| Input: | CS165 | is | an | awesome | class |
| Label: | is | an | awesome | class | EOS |

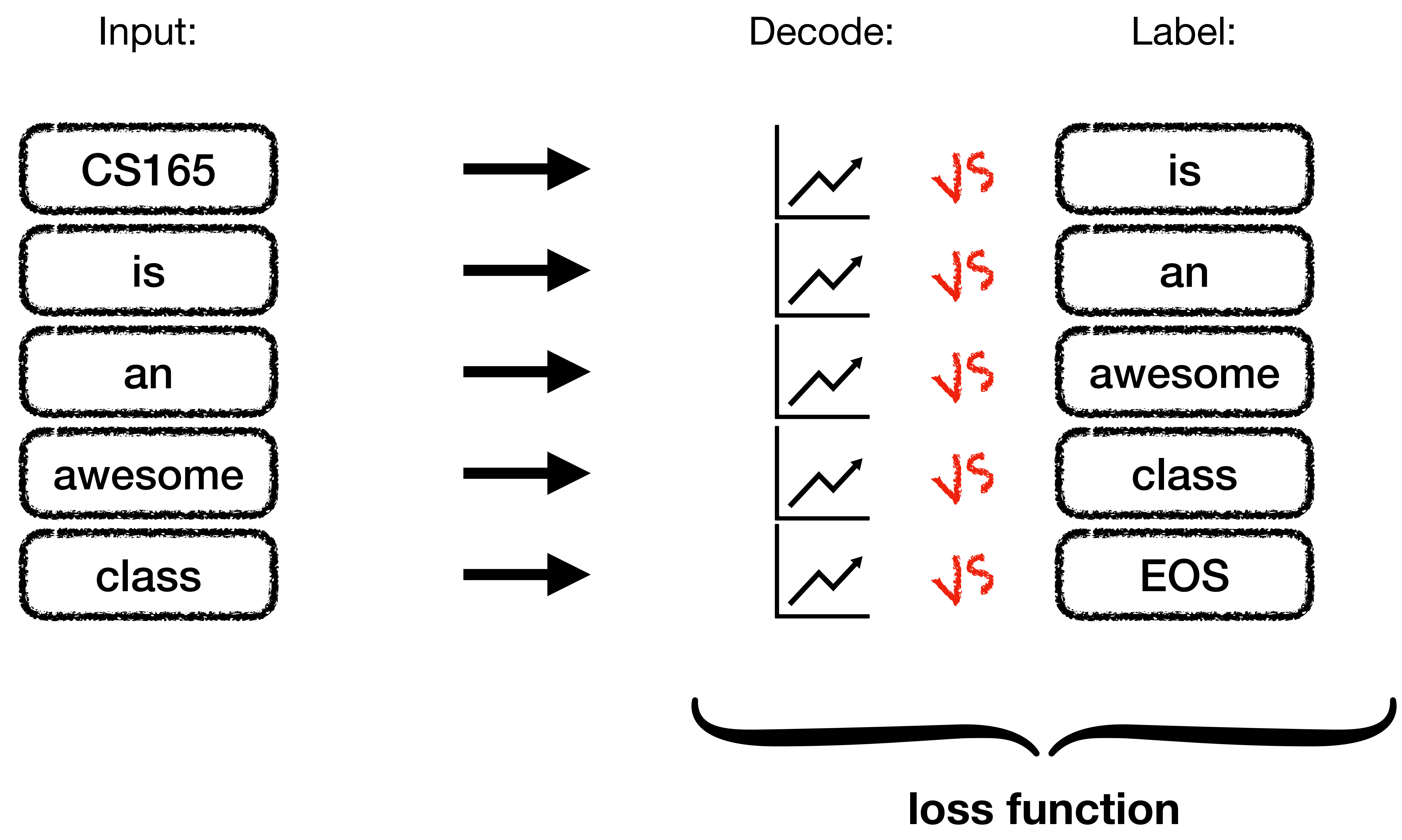
Cross-Entropy

$$\mathscr{L} = -\frac{1}{N}\sum_{i=1}^{N} \log p(x_i)$$

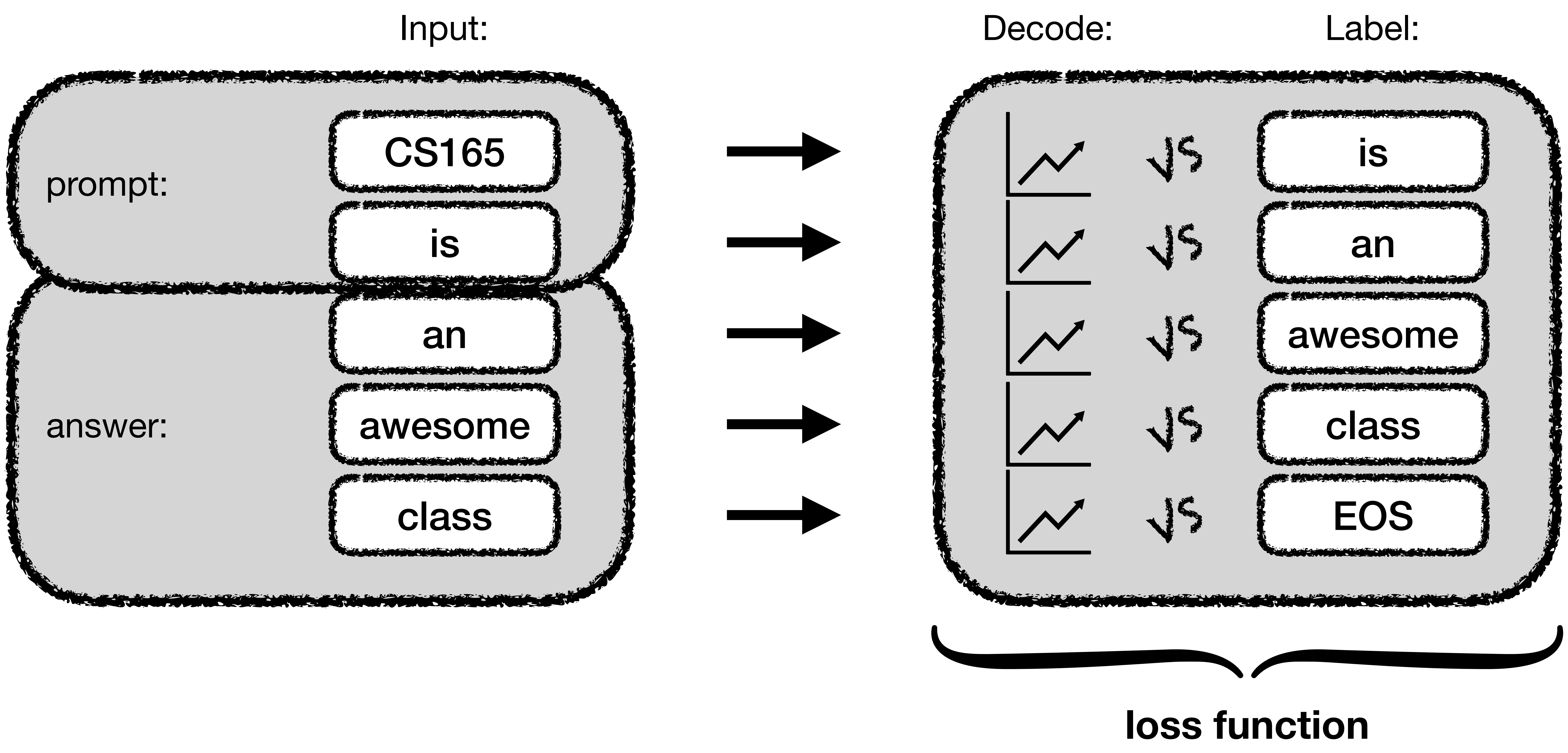
Perplexity

$$e^{\mathscr{L}} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N} \log p(x_i)\right)$$

post-training

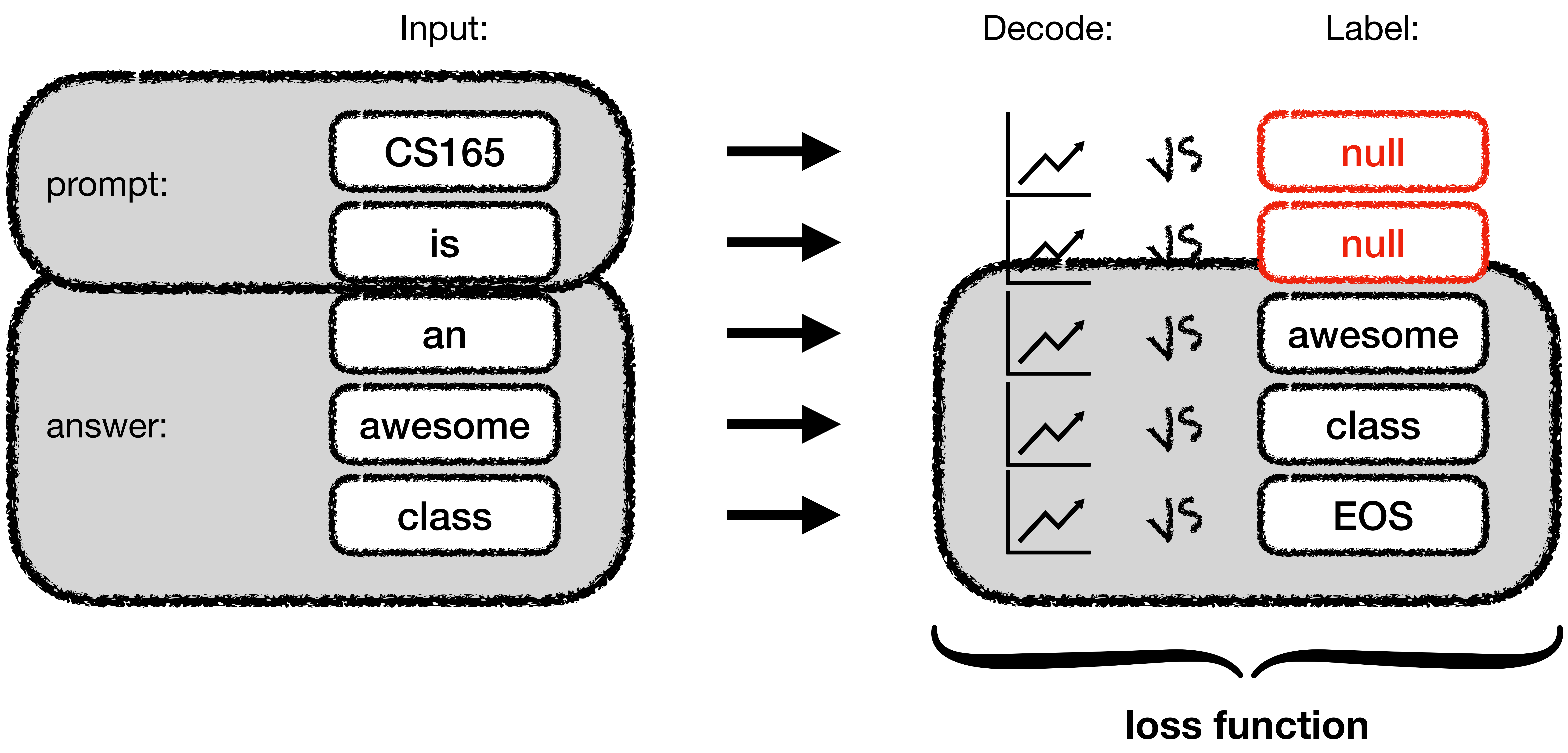

Input:
Decode:
Label:
CS165
is
an
awesome
class
vs
is
an
awesome
class
EOS
loss function
post-training

# Instruction Tuning




post-training

# Instruction Tuning




post-training

DASlab @ Harvard SEAS

# Instruction Tuning

## Evaluation

**Perplexity**

Input:

CS165

is

an → vs awesome

awesome → vs class

class → vs EOS

Decode:

Step 1:

**loss function**

**Accuracy**

Input: CS165 is

Step 1: → an

Step 2: → awesome

Step 3: → class → EOS

Generated Answer: class

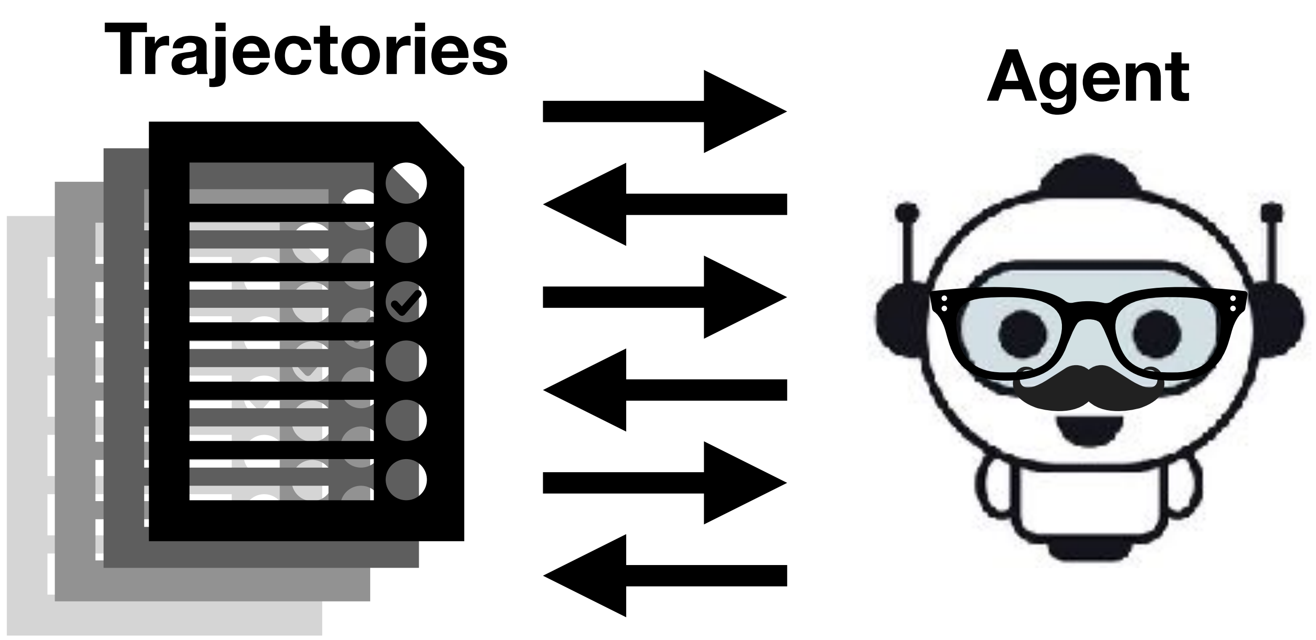
Golden Answer: class

post-training

DASlab @ Harvard SEAS

## Supervised Learning

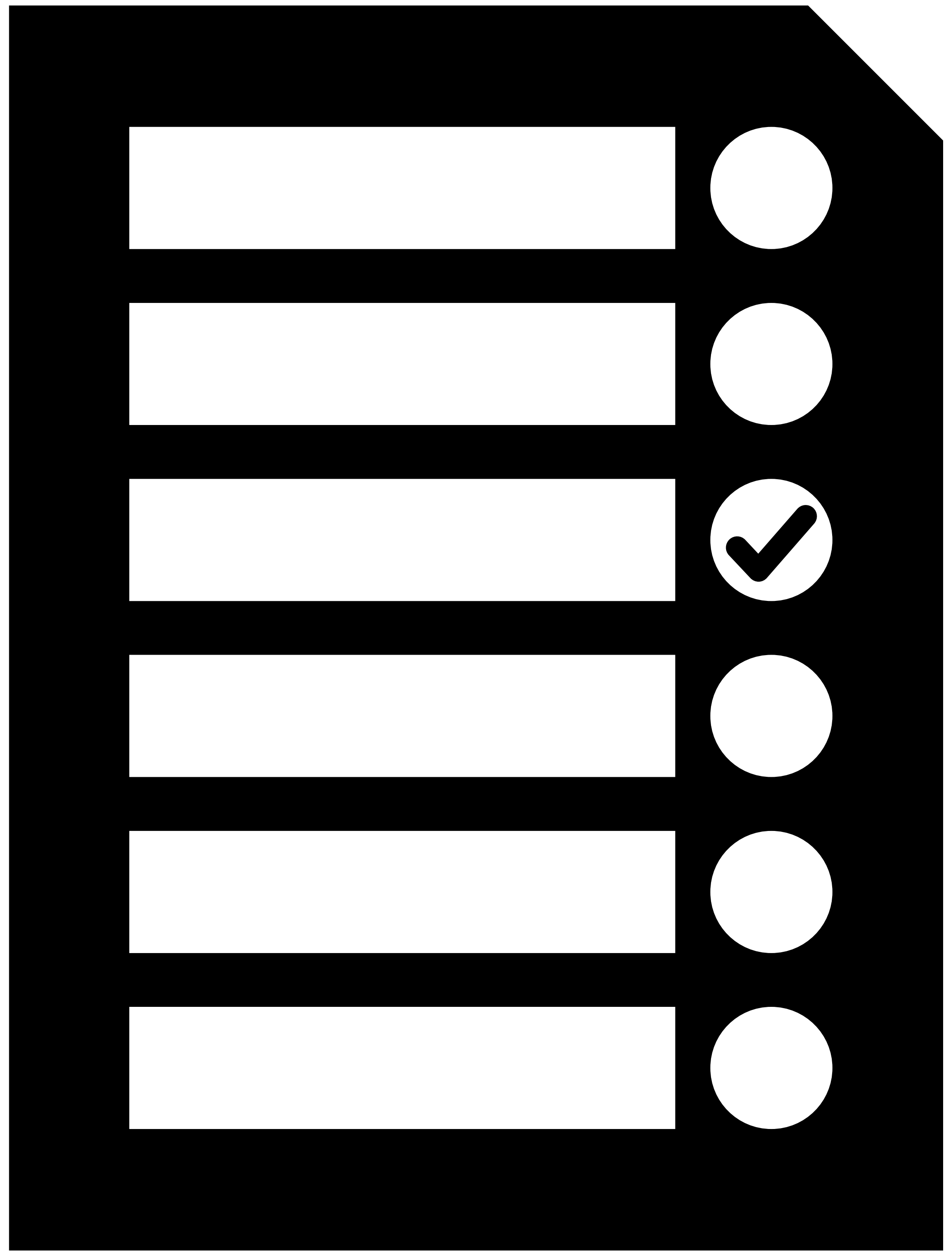
**Training**




How to Train your… AGENT!

**ReAct**
*Yao et al. 2022*

**Trajectory**



**User:** Find the population of Tokyo and compare it to New York City.

**Thought:** I should search for the current population of Tokyo first.
**Action:** search("population of Tokyo 2024")
**Observation:** Tokyo metropolitan area population is approximately 37.4 million.

**Thought:** Now I need the population of New York City.
**Action:** search("population of New York City 2024")
**Observation:** New York City population is approximately 8.3 million.

**Thought:** I have both numbers. I can now compute the comparison.
**Final Answer:** Tokyo (37.4M) is approximately 4.5 times larger than New York City (8.3M).


Supervised Learning
Training
Trajectories
Agent
Evaluation
Agent
Environment
How to Train your… AGENT!
Reinforcement Learning
Prob
Agent
Environment
Reward

## 1. Quality Assessment of Agent Fine-Tuning Benchmarks

# PROBLEM

# Agent fine-tuning rests on benchmarks the quality of which has not been systematically assessed

*If the scores cannot be trusted — the conclusions built on top of them cannot be trusted either*

## 1. Quality Assessment of Agent Fine-Tuning Benchmarks

# SOLUTION

# Build a unified quality framework to characterize what agent benchmarks actually measure

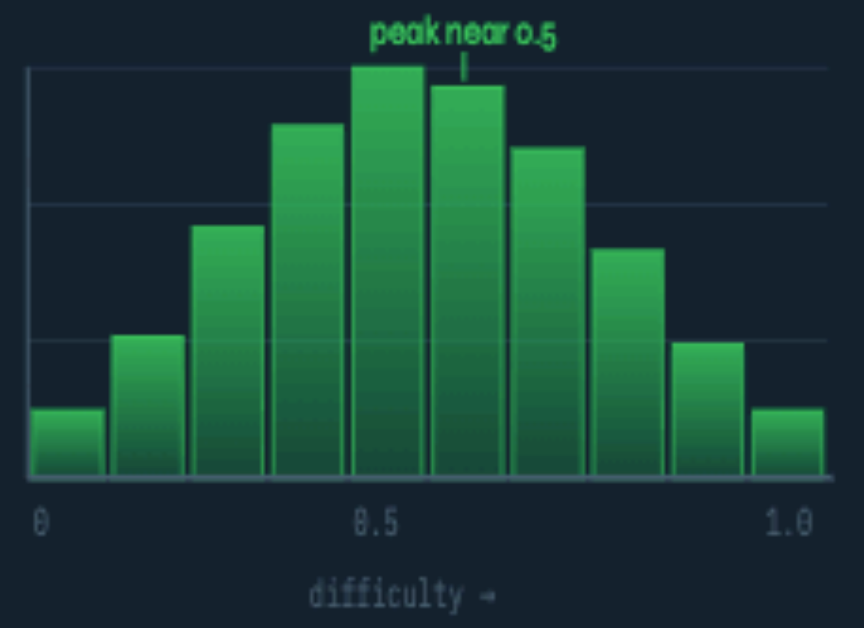
*Propose new benchmarks to fill in the gaps*

# Benchmark Difficulty Curve

Each bar is a bucket of tasks. Height = number of tasks in that bucket. X-axis = empirical difficulty (fraction of model-run pairs that fail the task). Shape reveals whether a benchmark discriminates capability.

## Smooth unimodal spread

GOOD




- **Tasks span the full range.** Every difficulty tier is represented — easy, medium, hard.
- **Score differences are meaningful.** A better model shifts the boundary rightward; you can detect that.
- **Good for both SFT and RL.** Mid-range tasks provide demonstrations and reward signal.

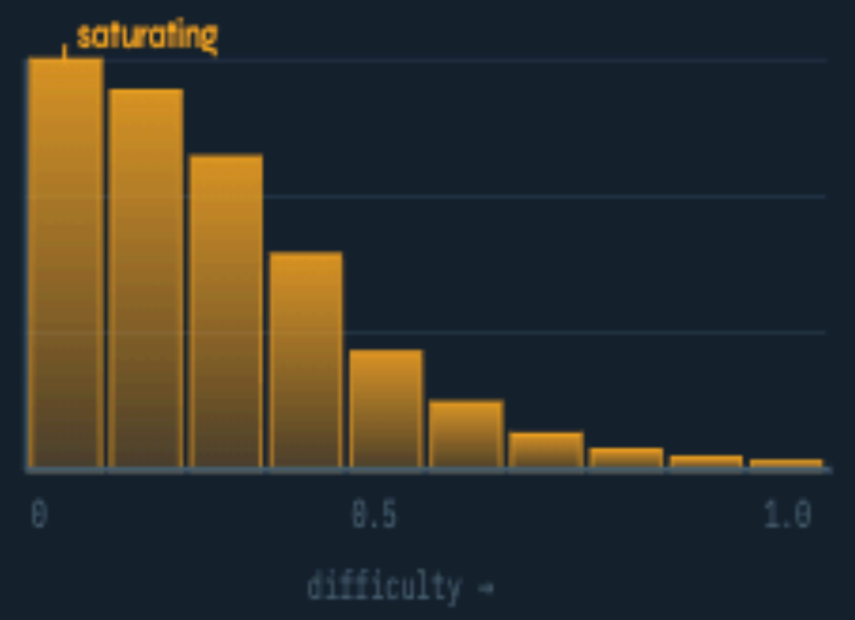
## Bimodal / U-shaped

BAD


- **Measures one threshold, not a range.** Tasks are trivial or impossible — nothing in between.
- **Training improvements are invisible.** A better model still fails the hard tasks and already solved the easy ones.
- **RL gets no gradient signal.** Reward variance near zero in both tails — nothing to optimize against.

## Left-skewed — too easy

BAD




- **Scores cluster near the ceiling.** Most models solve most tasks — no room to improve. AgentGym shows this pattern.
- **Cannot distinguish strong models.** A fine-tuned model looks roughly the same as the baseline.
- **SFT overfits to easy patterns.** Training on these tasks doesn't generalize; it just memorizes solved trajectories.

## Right-skewed — too hard

BAD


- **Scores cluster near zero.** The benchmark was designed for a capability level the field hasn't reached yet.
- **RL starves.** Successes are too rare for stable reward signal — gradients are noisy and training diverges.
- **SFT has no demonstrations.** If the teacher model also fails, there are no successful trajectories to imitate.

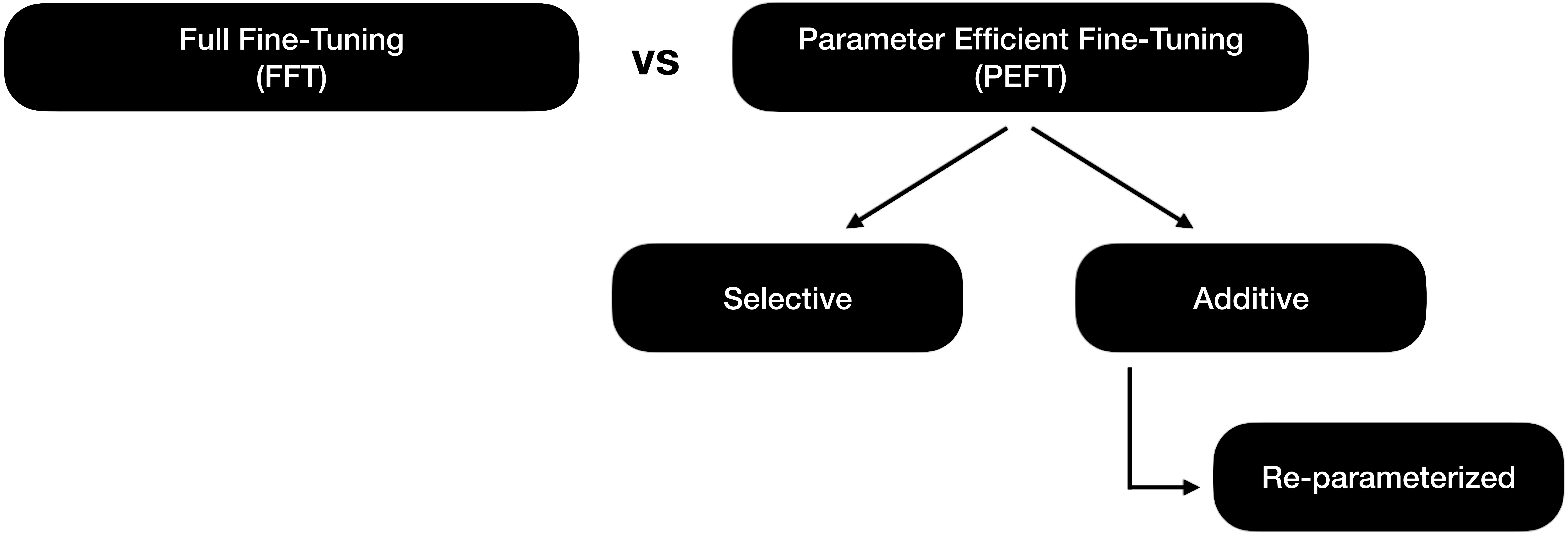
# Weight Update

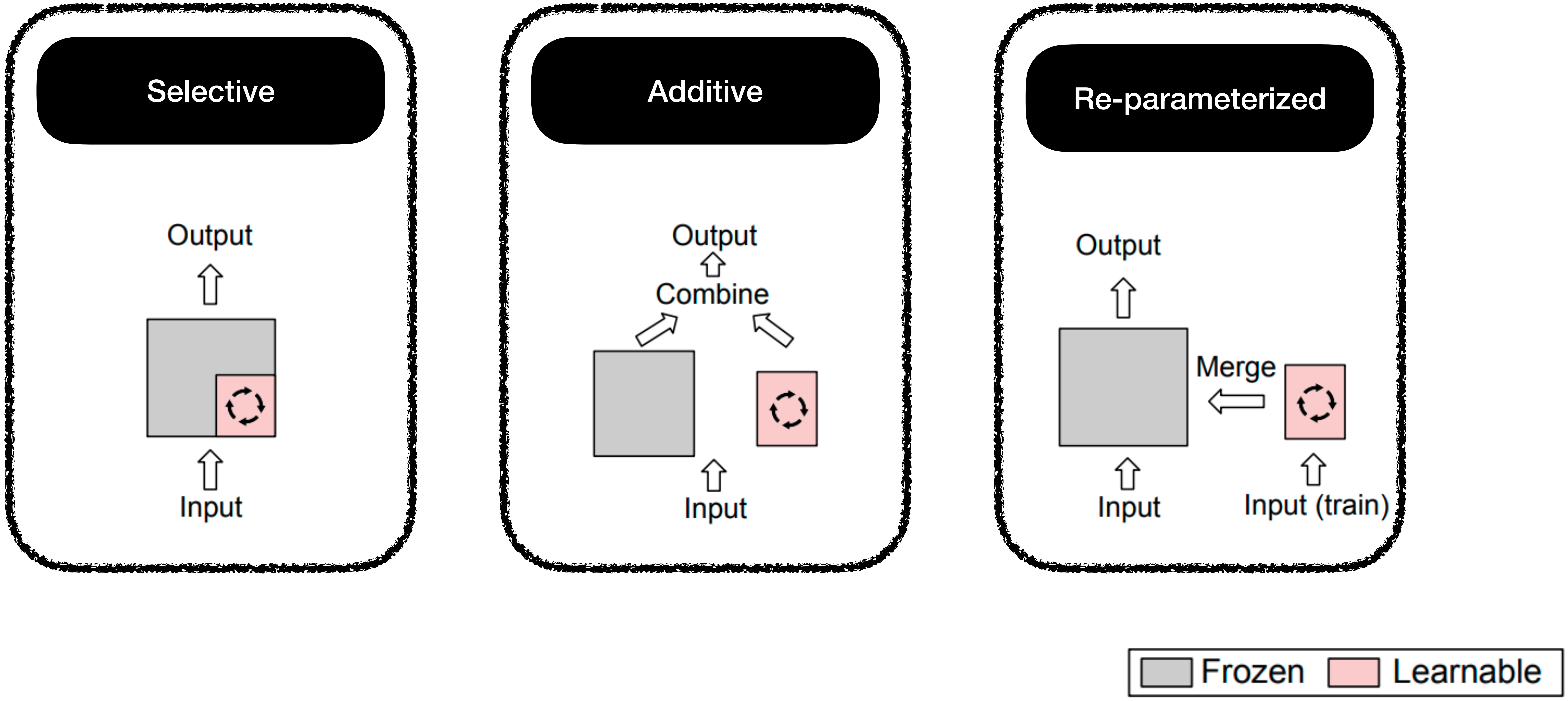
Full Fine-Tuning
(FFT)

vs

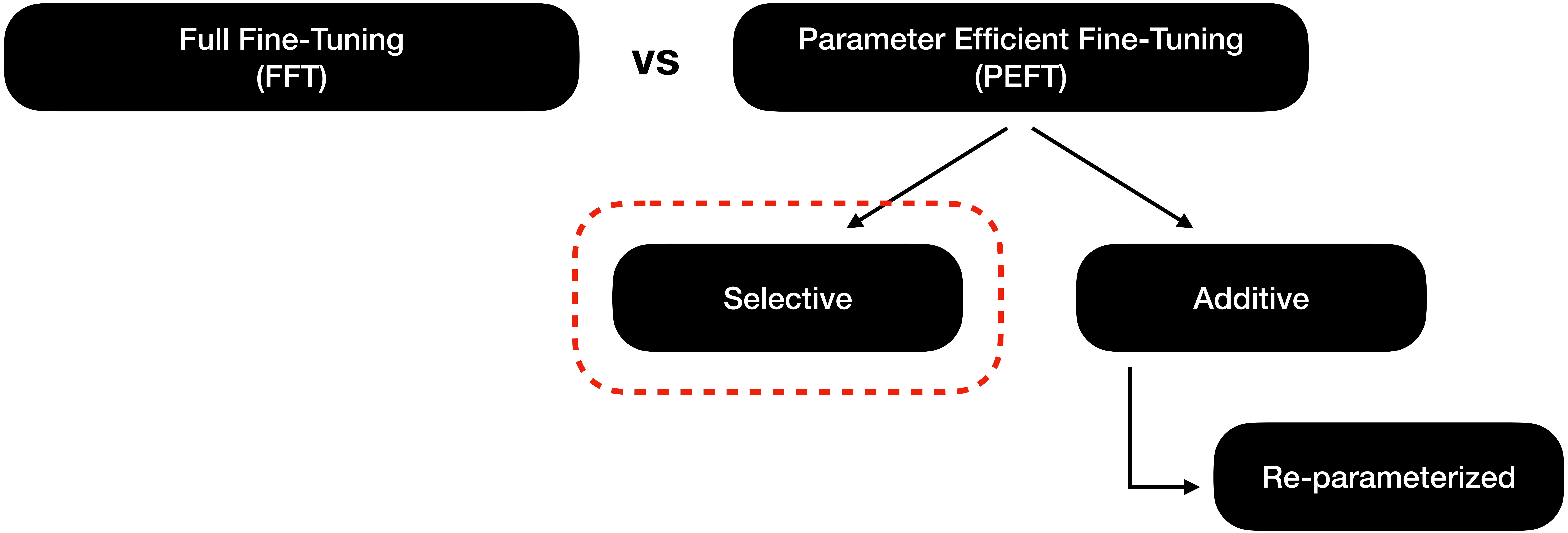
Parameter Efficient Fine-Tuning
(PEFT)

# PEFT Taxonomy

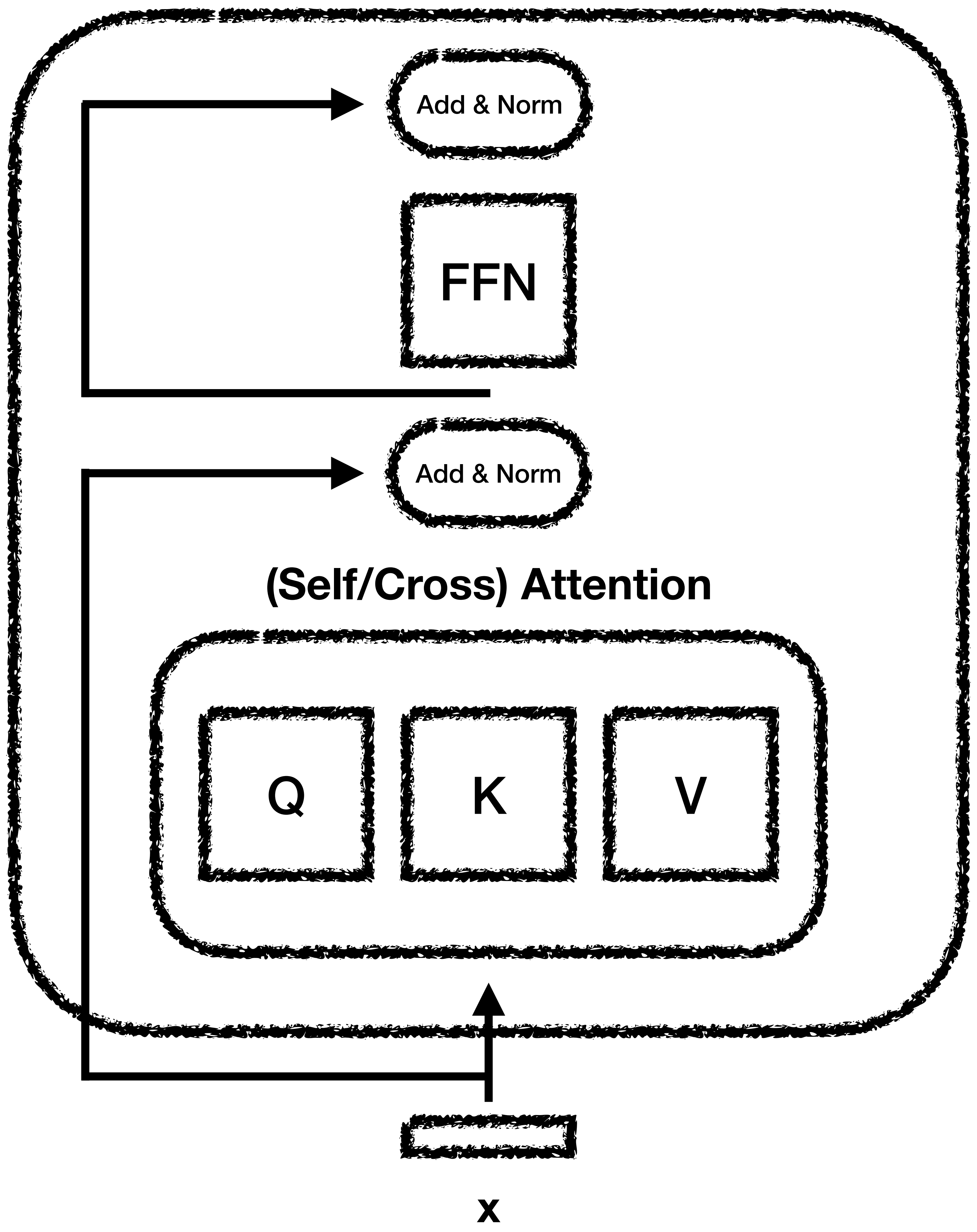



Selective
Output
Input
Additive
Output
Combine
Input
Re-parameterized
Output
Merge
Input
Input (train)
Frozen
Learnable

# PEFT Taxonomy

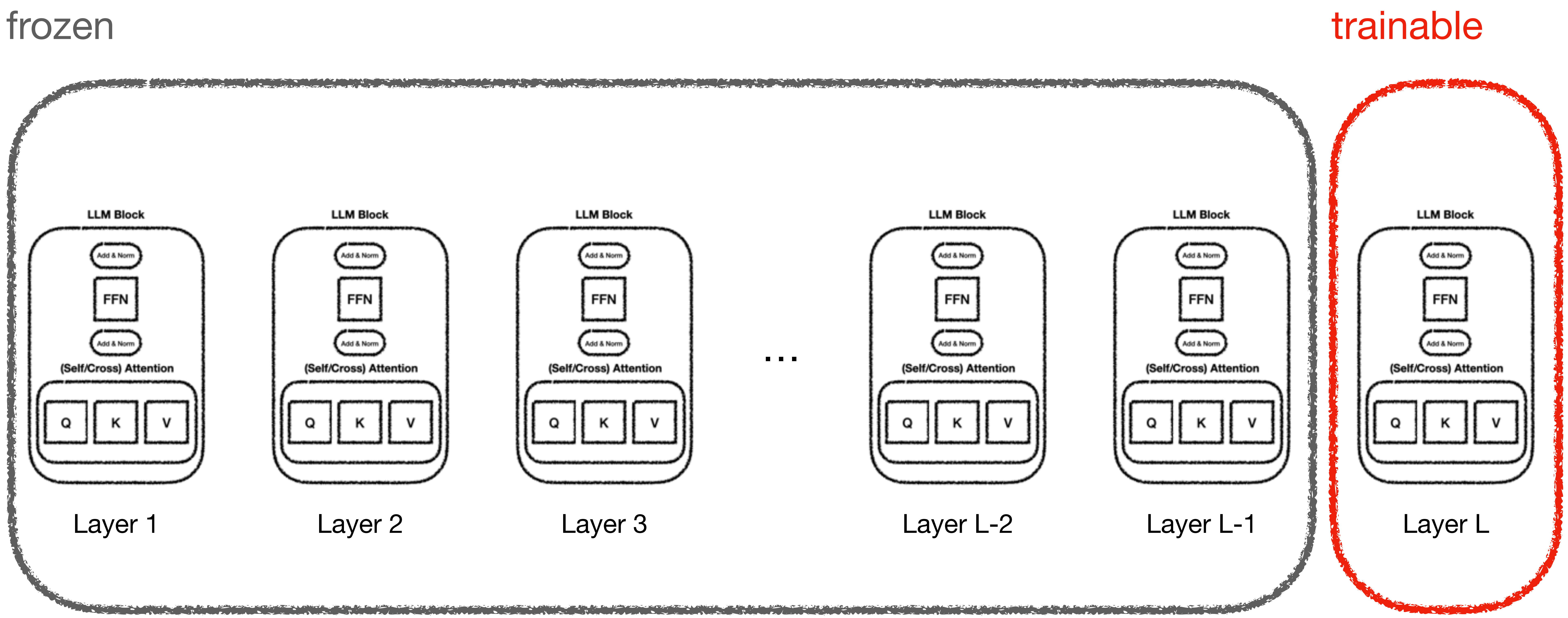


# LLM Block

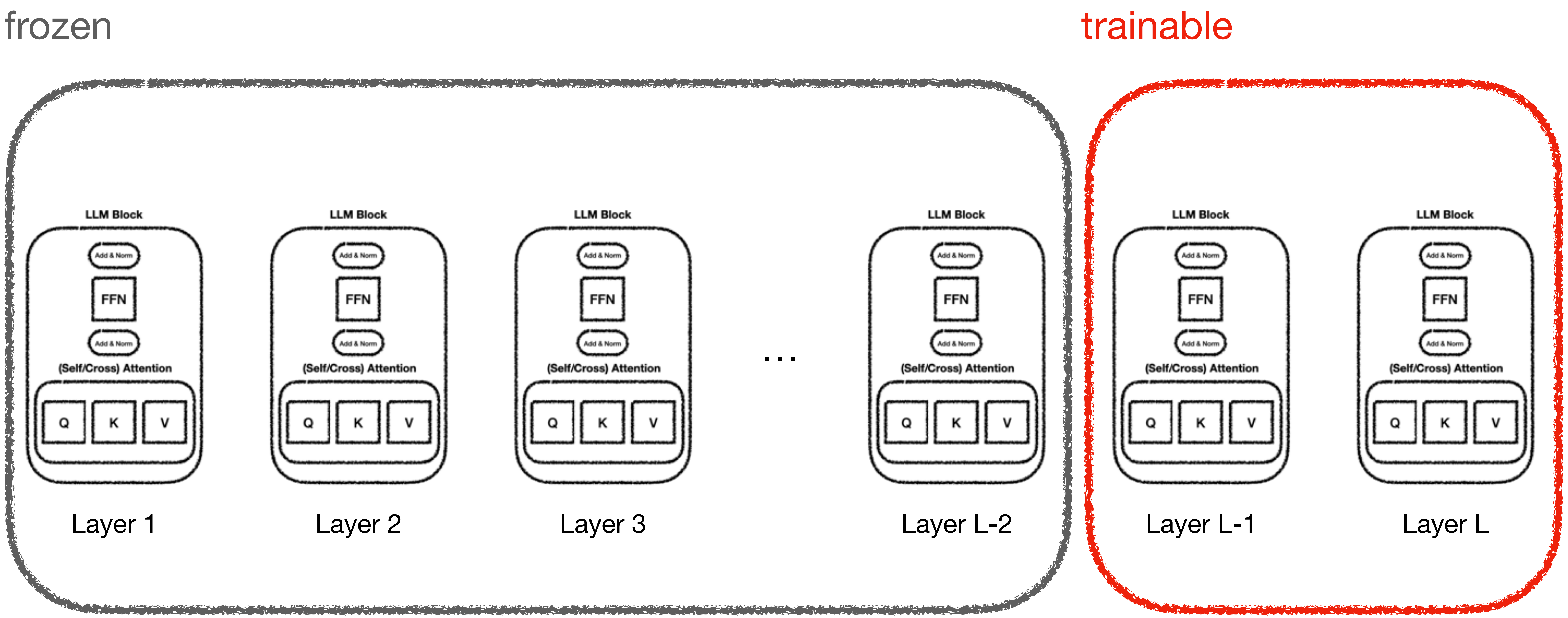



DASlab @ Harvard SEAS

# Selective PEFT



# Selective PEFT



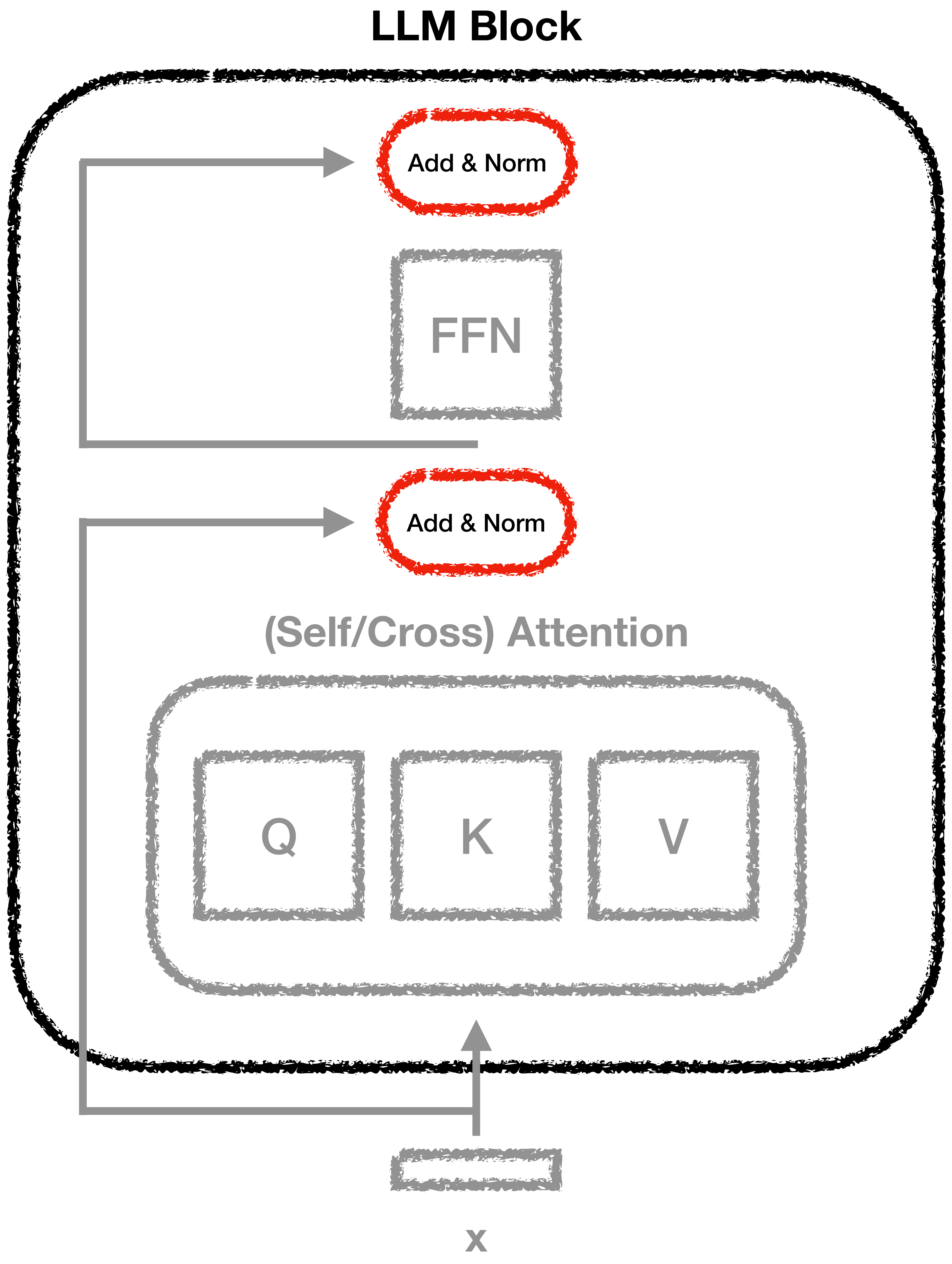
# Selective PEFT

## Two more ideas

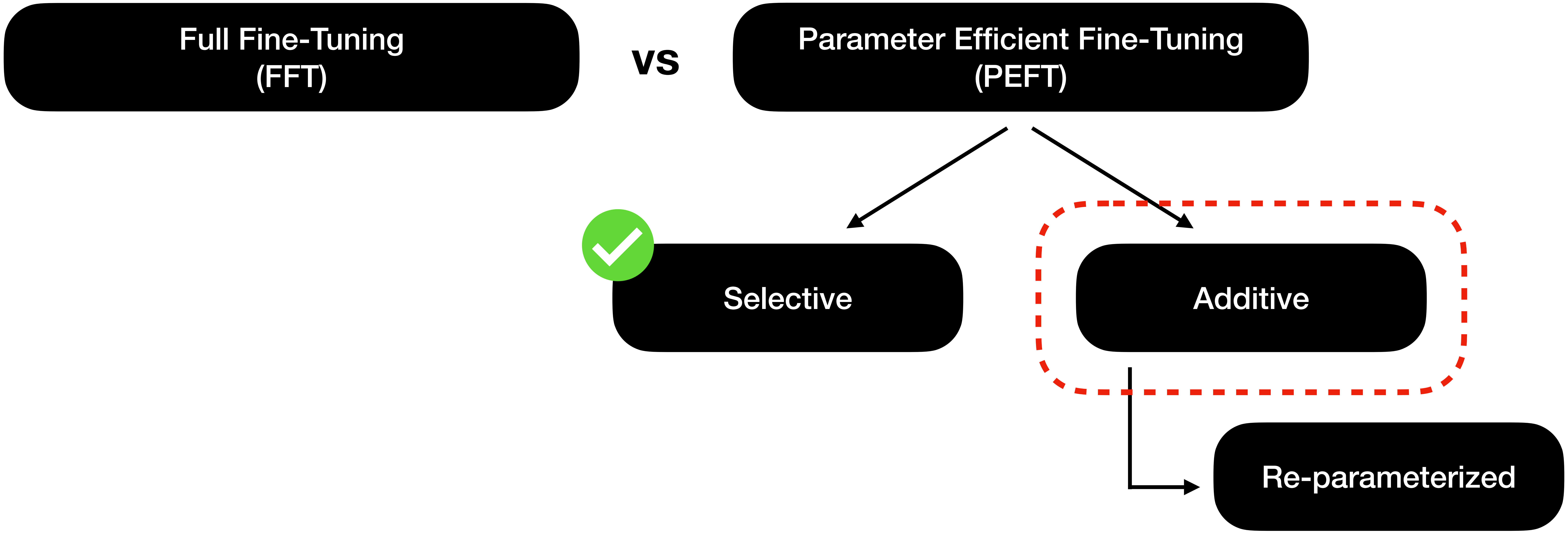
**BitFit**
**(Ben Zaken et al., 2021)**

$$\mathbf{Q}^{m,\ell}(\mathbf{x}) = \mathbf{W}_q^{m,\ell}\mathbf{x} + \mathbf{b}_q^{m,\ell}$$
$$\mathbf{K}^{m,\ell}(\mathbf{x}) = \mathbf{W}_k^{m,\ell}\mathbf{x} + \mathbf{b}_k^{m,\ell}$$
$$\mathbf{V}^{m,\ell}(\mathbf{x}) = \mathbf{W}_v^{m,\ell}\mathbf{x} + \mathbf{b}_v^{m,\ell}$$
$$\mathbf{h}_2^\ell = \mathrm{Dropout}\big(\mathbf{W}_{m_1}^\ell \cdot \mathbf{h}_1^\ell \quad + \quad \mathbf{b}_{m_1}^\ell\big)$$
$$\mathbf{h}_3^\ell = \mathbf{g}_{LN_1}^\ell \odot \frac{(\mathbf{h}_2^\ell + \mathbf{x}) - \mu}{\sigma} + \mathbf{b}_{LN_1}^\ell$$
$$\mathbf{h}_4^\ell = \quad \mathrm{GELU}\big(\mathbf{W}_{m_2}^\ell \cdot \mathbf{h}_3^\ell \quad + \quad \mathbf{b}_{m_2}^\ell\big)$$
$$\mathbf{h}_5^\ell = \mathrm{Dropout}\big(\mathbf{W}_{m_3}^\ell \cdot \mathbf{h}_4^\ell \quad + \quad \mathbf{b}_{m_3}^\ell\big)$$
$$\mathrm{out}^\ell = \mathbf{g}_{LN_2}^\ell \odot \frac{(\mathbf{h}_5^\ell + \mathbf{h}_3^\ell) - \mu}{\sigma} + \mathbf{b}_{LN_2}^\ell$$

**LayerNorm Tuning**
**(Bingchen Zhao et al., 2024, ICLR)**



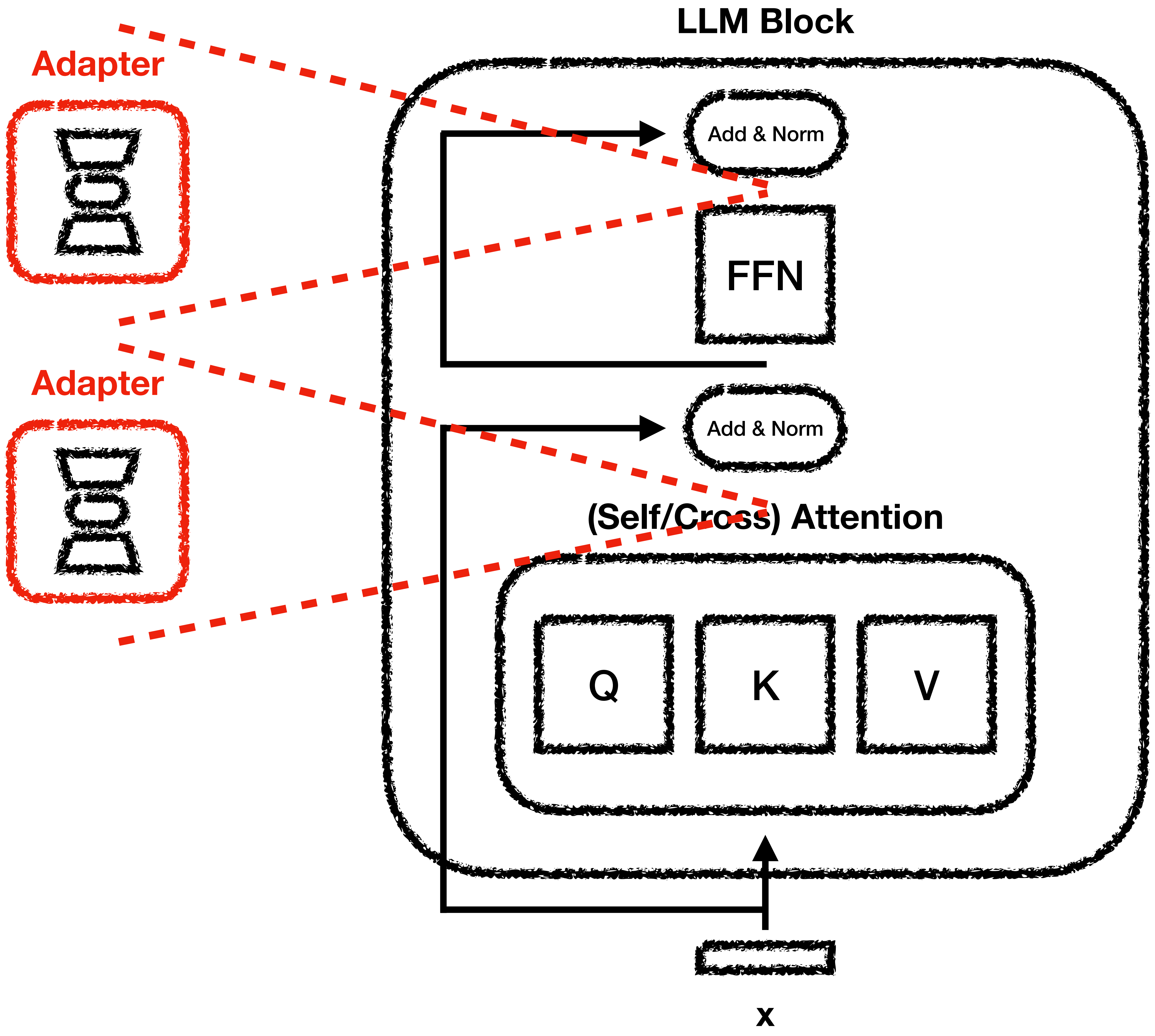
# PEFT Taxonomy




DASlab
@ Harvard SEAS

# Additive PEFT

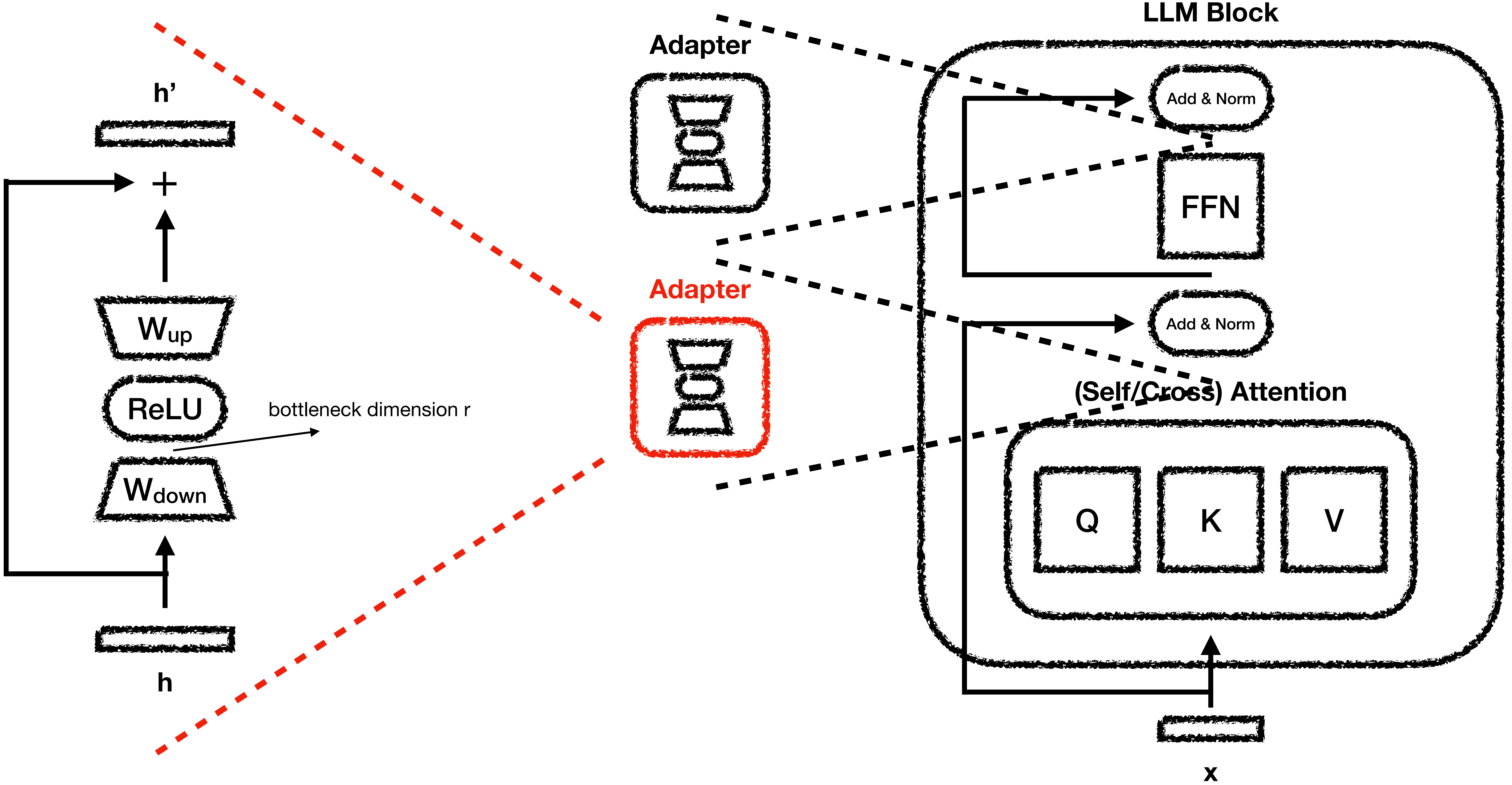
## The Adapter Architecture




DASlab @ Harvard SEAS

# Additive PEFT

## The Adapter Architecture




DASlab @ Harvard SEAS

# Additive PEFT

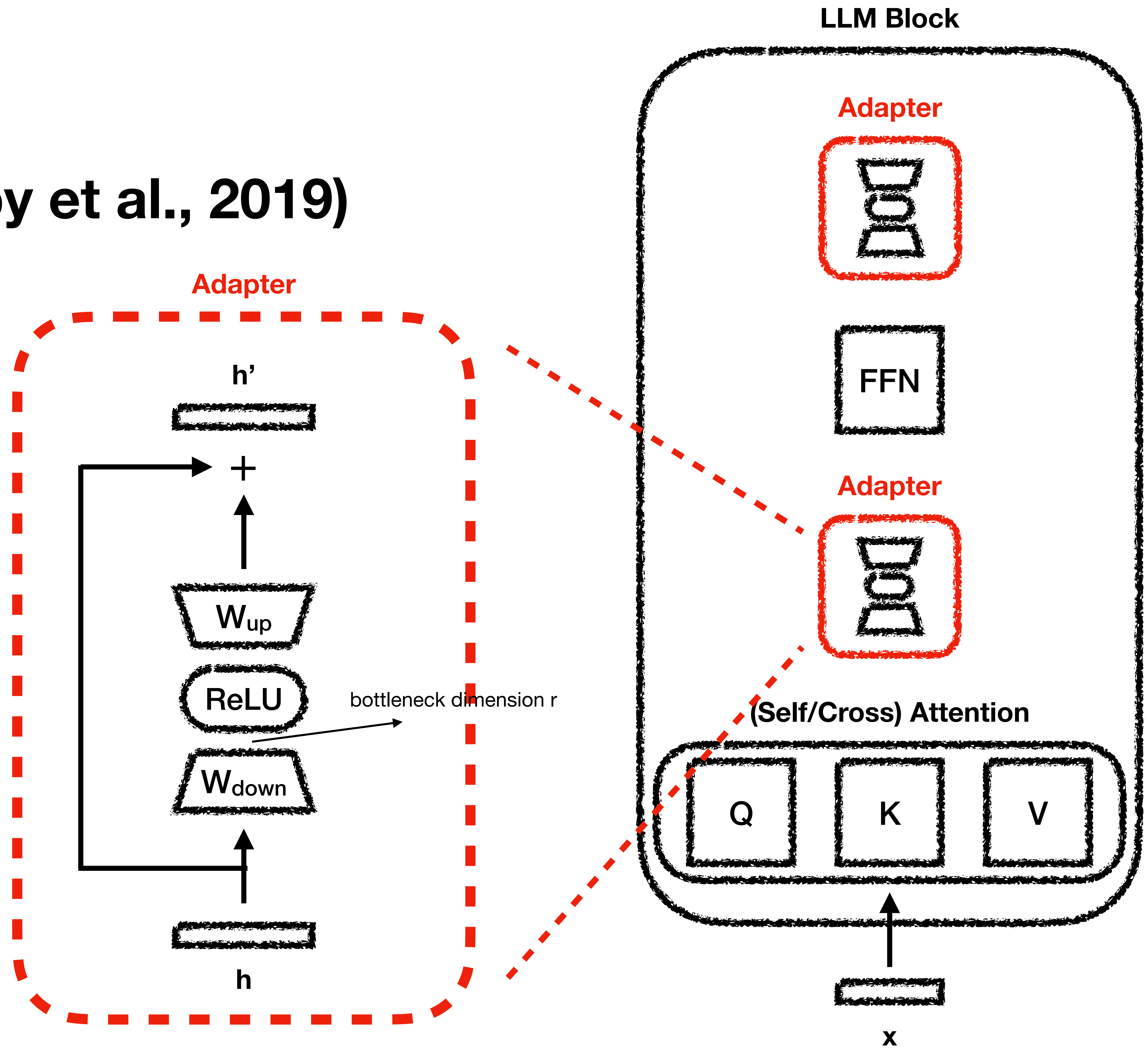
## [Serial] Adapter (Houlsby et al., 2019)

$$h' \leftarrow h + f(hW_{down})W_{up}$$

$$W_{down} \in \mathbb{R}^{d \times r}$$

$$W_{up} \in \mathbb{R}^{r \times d}$$

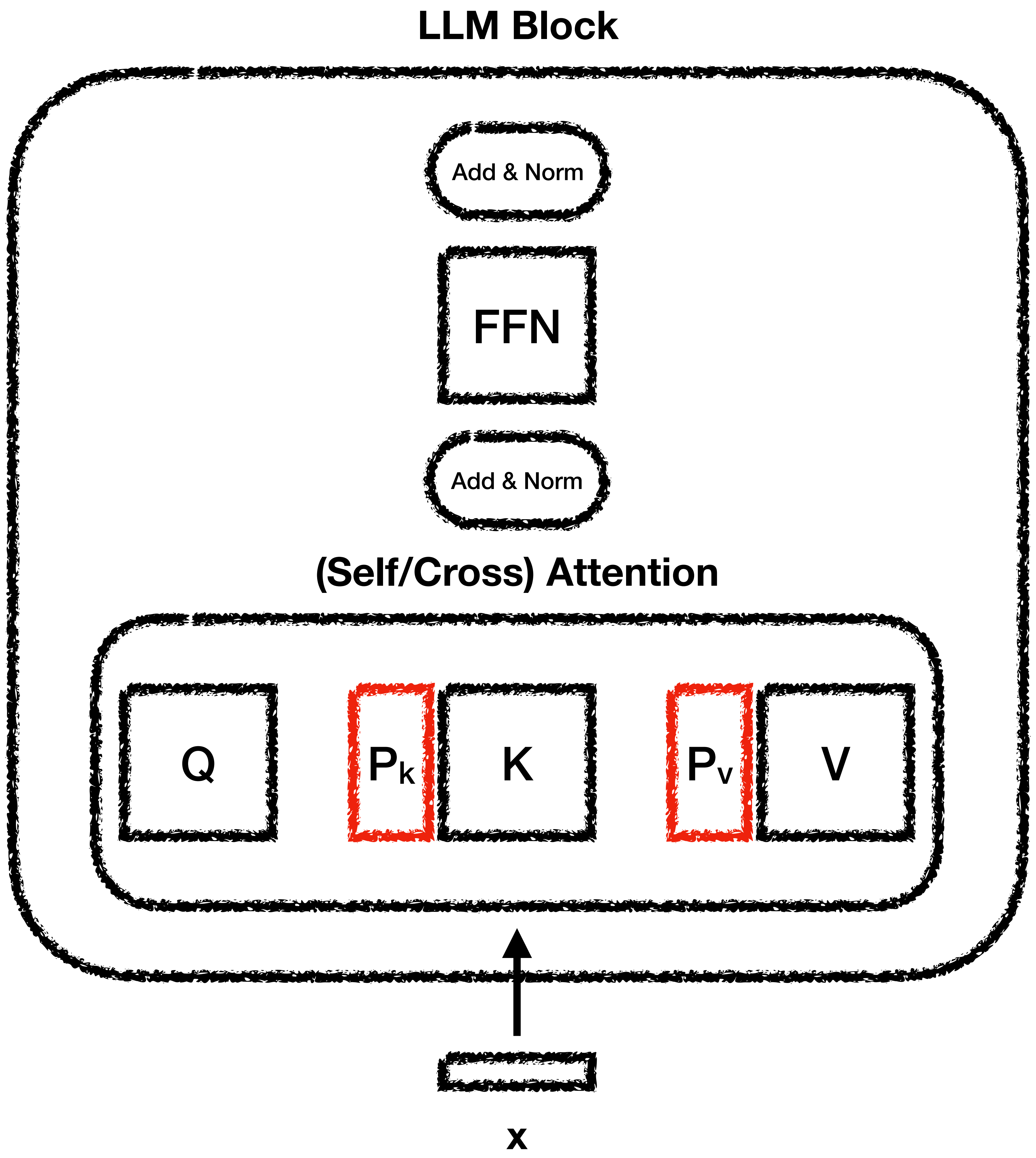
Adapters: small trainable feed-forward networks inserted between the layers in the frozen pre-trained model.




DASlab @ Harvard SEAS

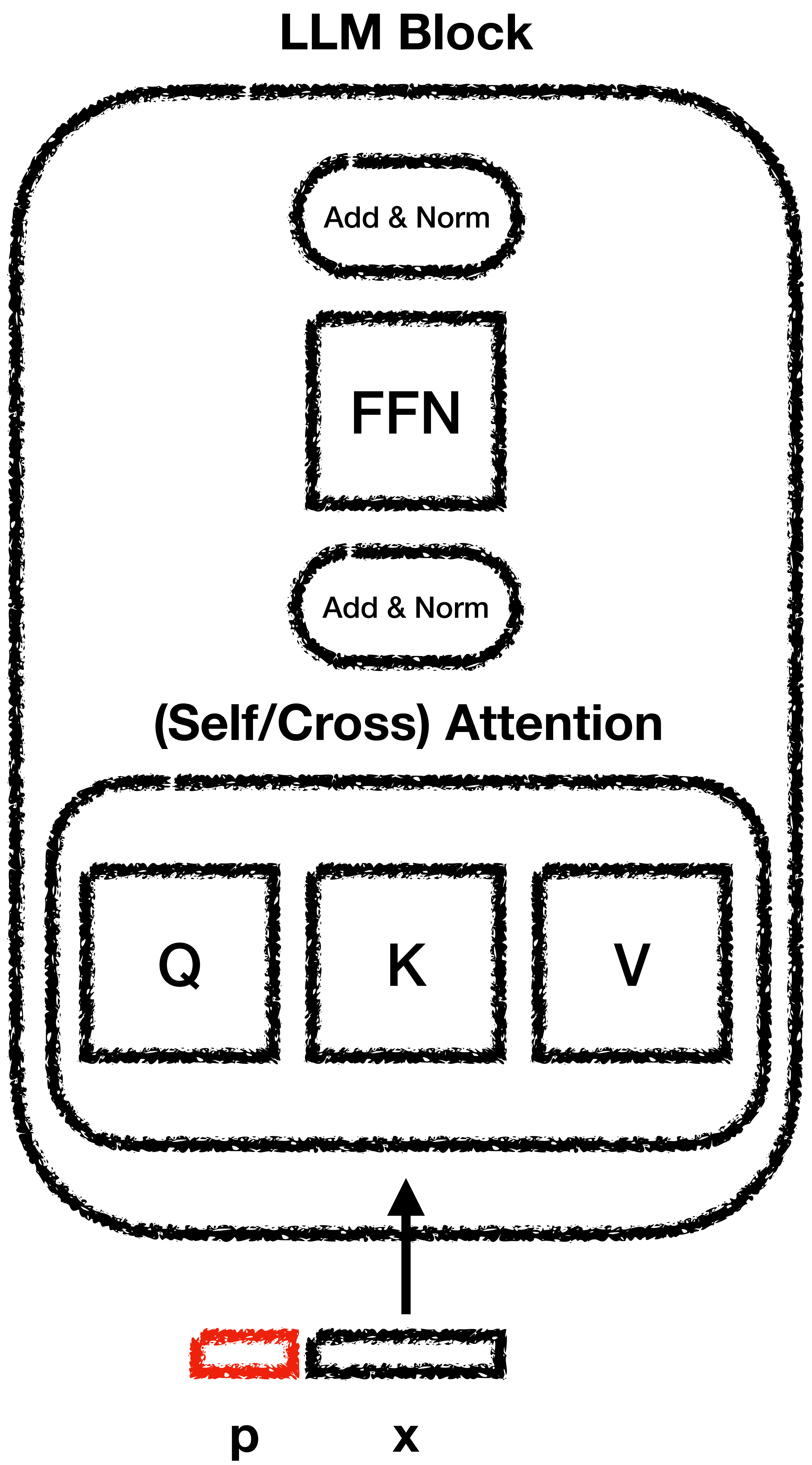
# Additive PEFT

## Prefix Tuning (Li & Liang, 2021)



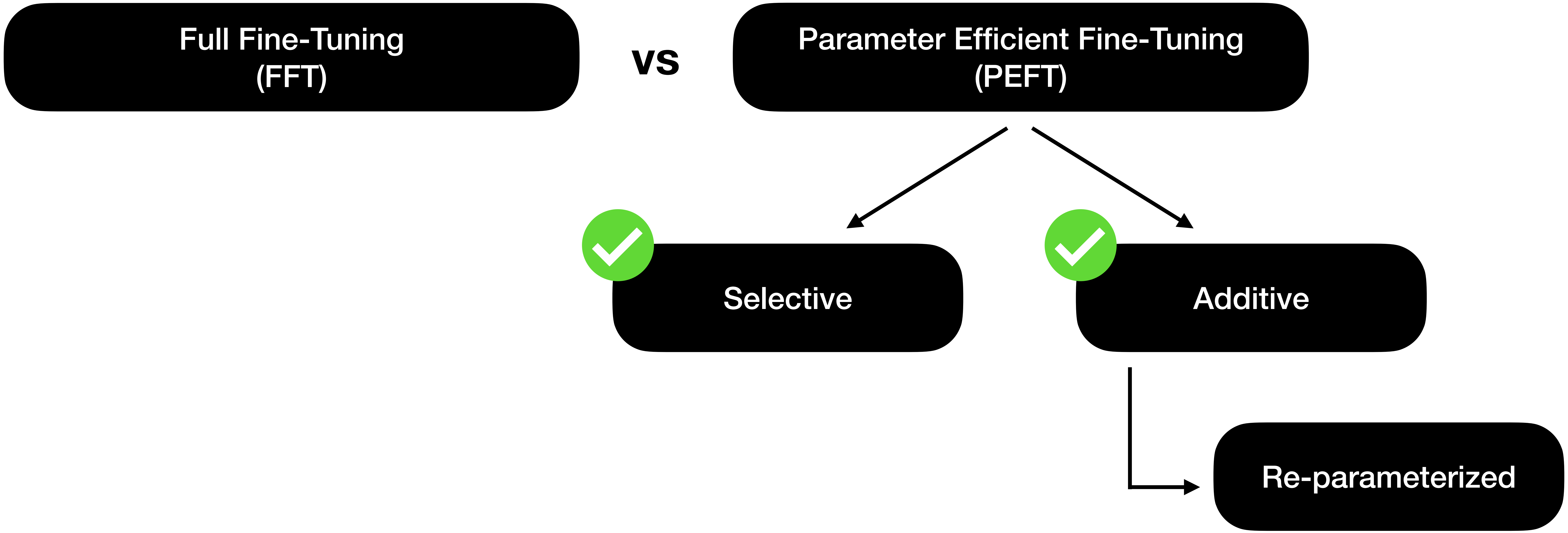
# Additive PEFT

## Prompt Tuning (Brian Lester et al., 2021)

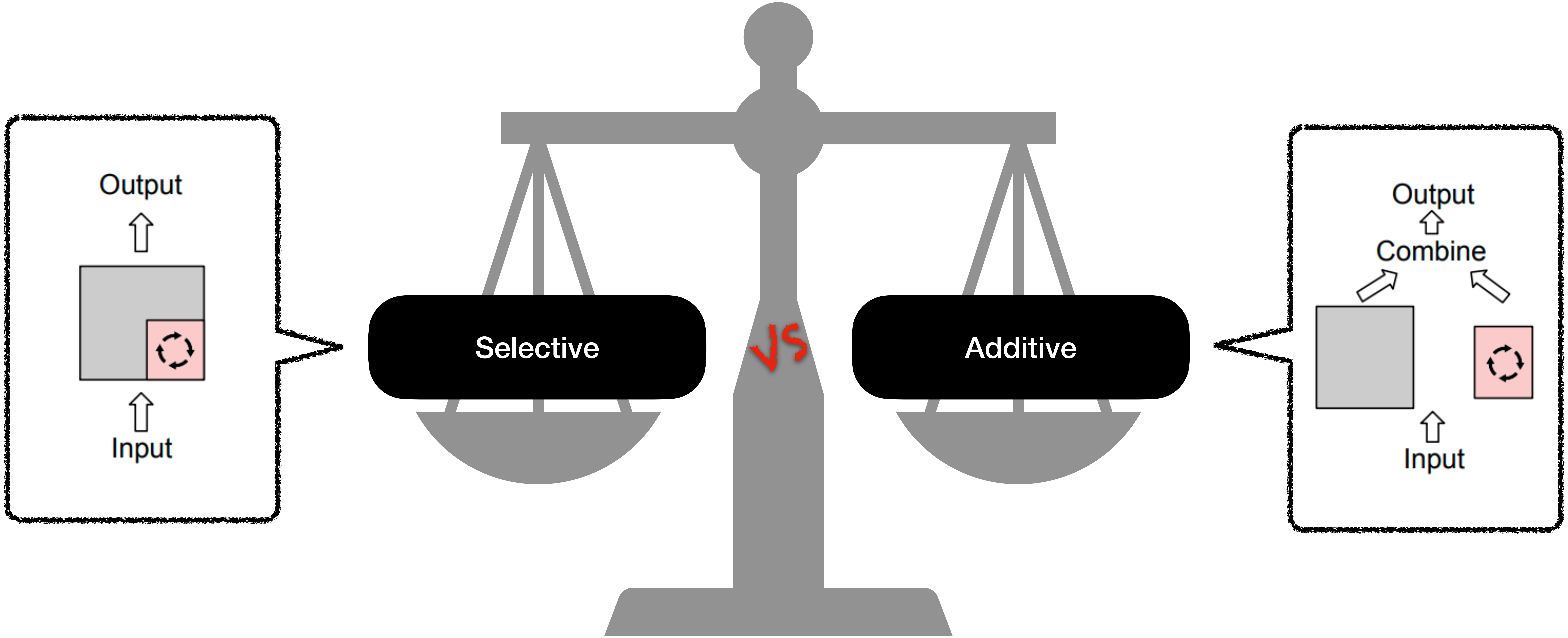



DASlab @ Harvard SEAS

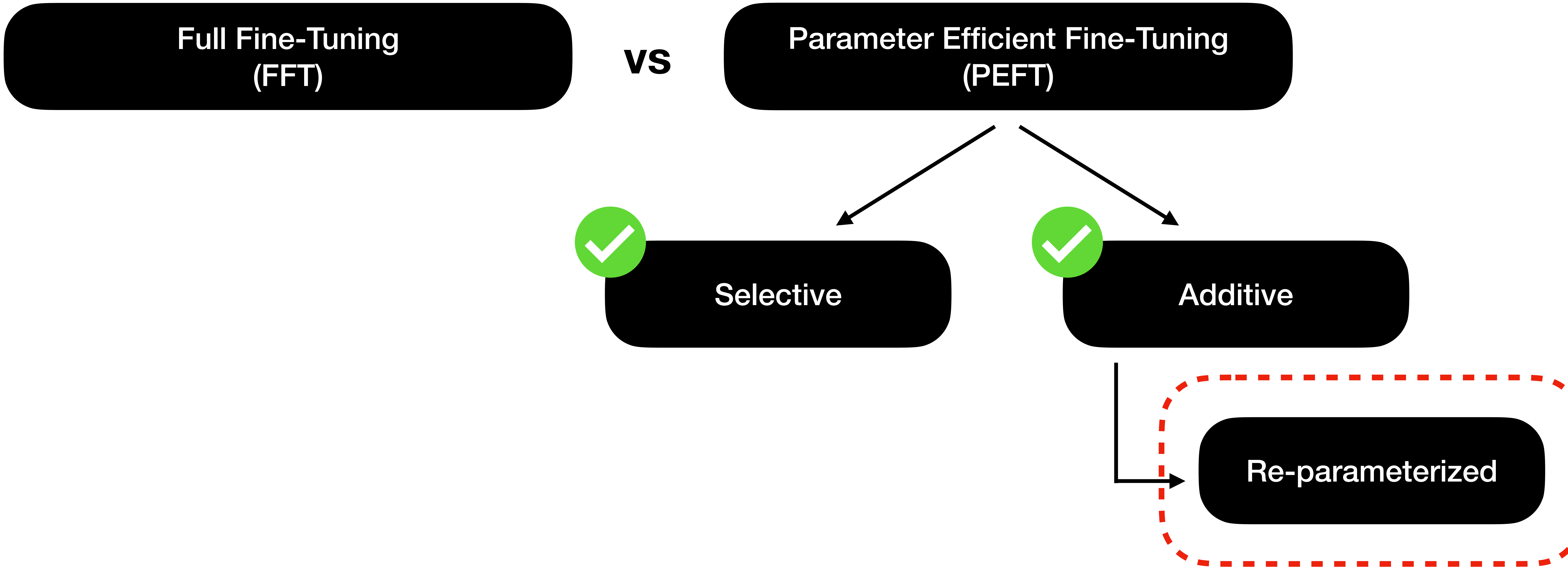
# PEFT Taxonomy



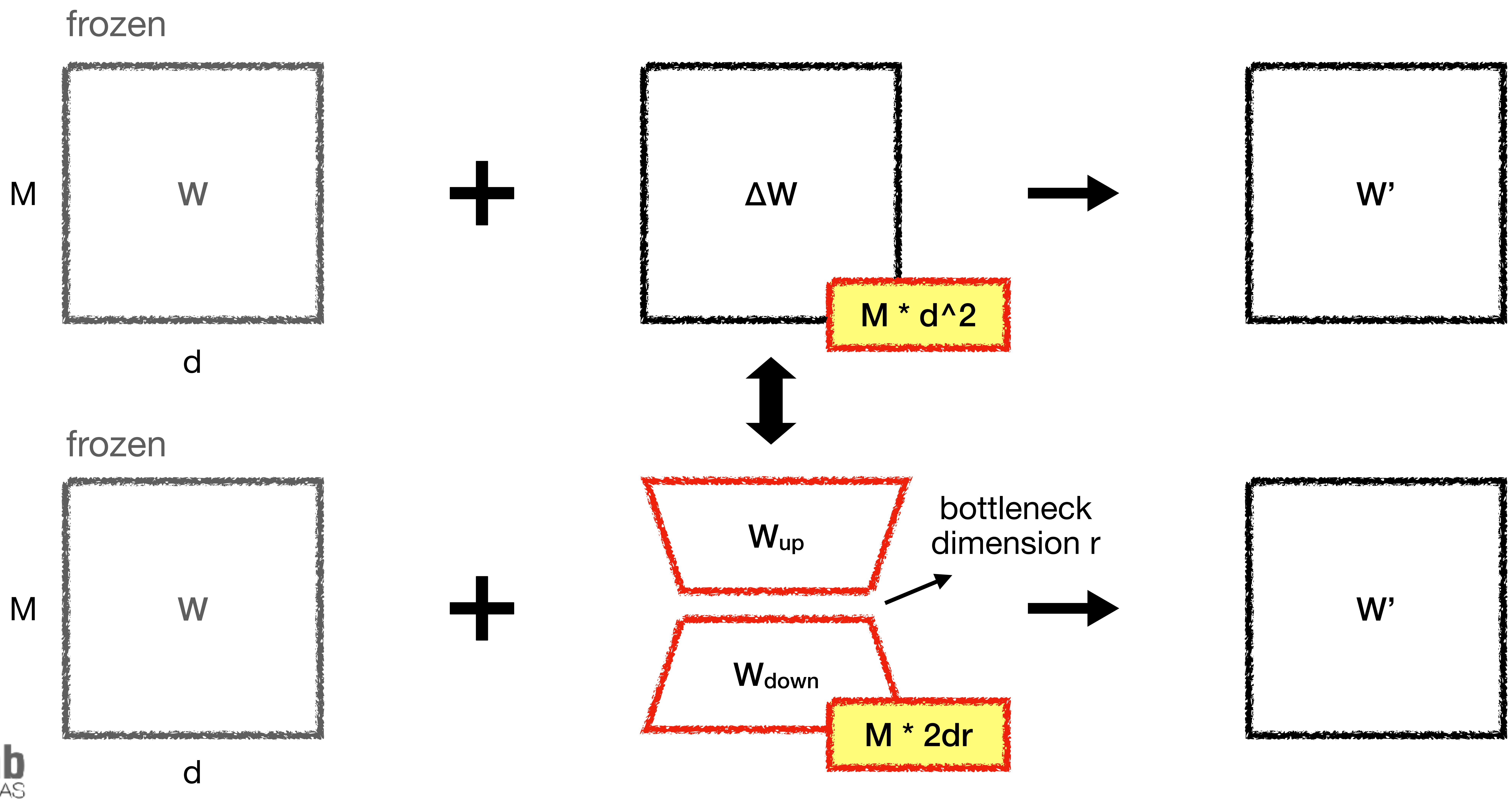
# Which one is better from a systems perspective?



# PEFT Taxonomy



# Re-parameterized PEFT




DASlab @ Harvard SEAS

# Re-parameterized PEFT

## Low Rank Adaptation (Hu et al., 2021)

$$h \leftarrow xW + \frac{a}{r} \cdot xW_{down}W_{up}$$

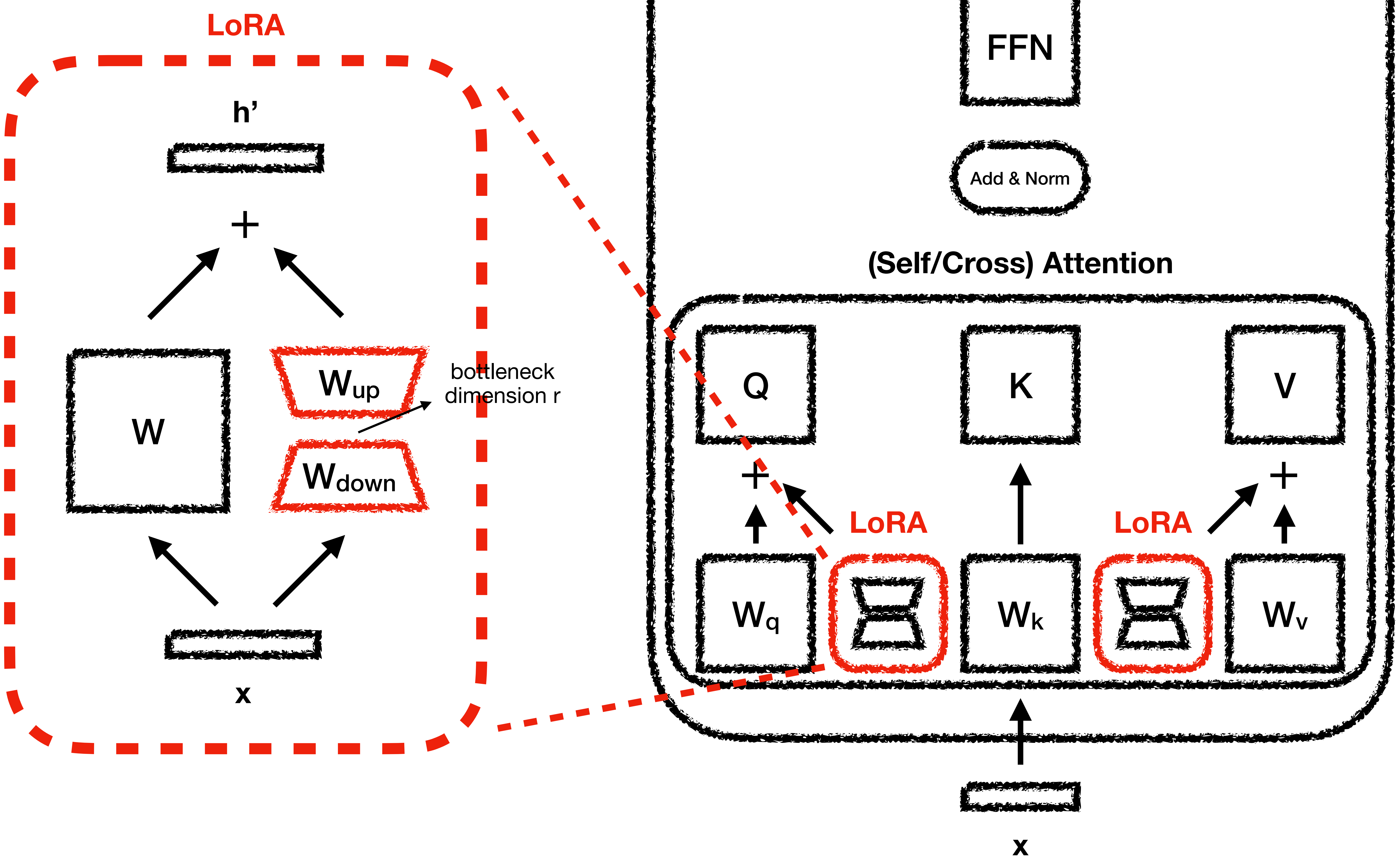
$x \in \mathbb{R}^{1\times d}$

$h, h' \in \mathbb{R}^{1\times k}$

$W \in \mathbb{R}^{d\times k}$

$W_{down} \in \mathbb{R}^{d\times r}$

$W_{up} \in \mathbb{R}^{r\times k}$




DASlab @ Harvard SEAS

# Re-parameterized PEFT

## Low Rank Adaptation (Hu et al., 2021)

$$h \leftarrow xW + \frac{a}{r} \cdot xW_{down}W_{up}$$

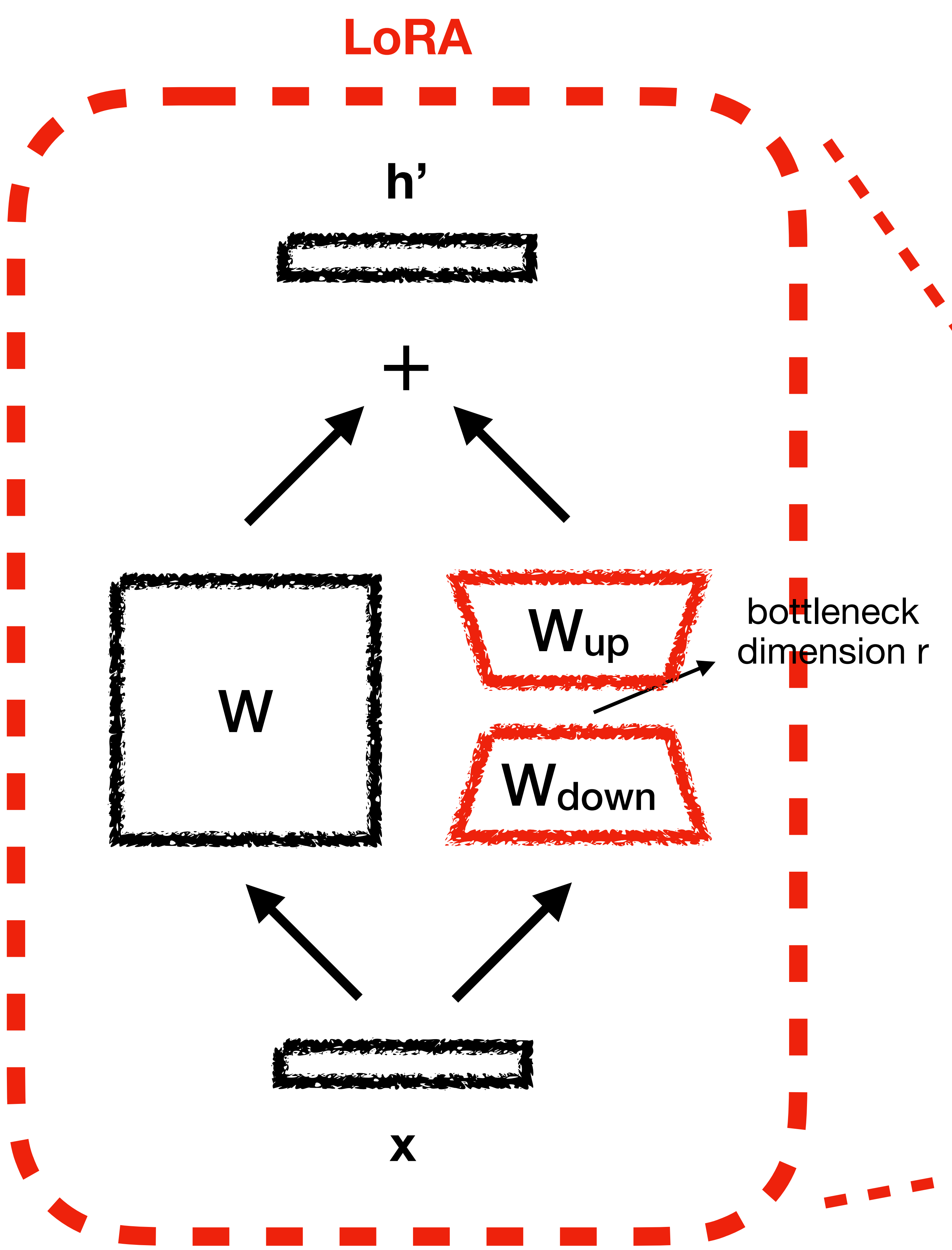
$x \in \mathbb{R}^{1\times d}$

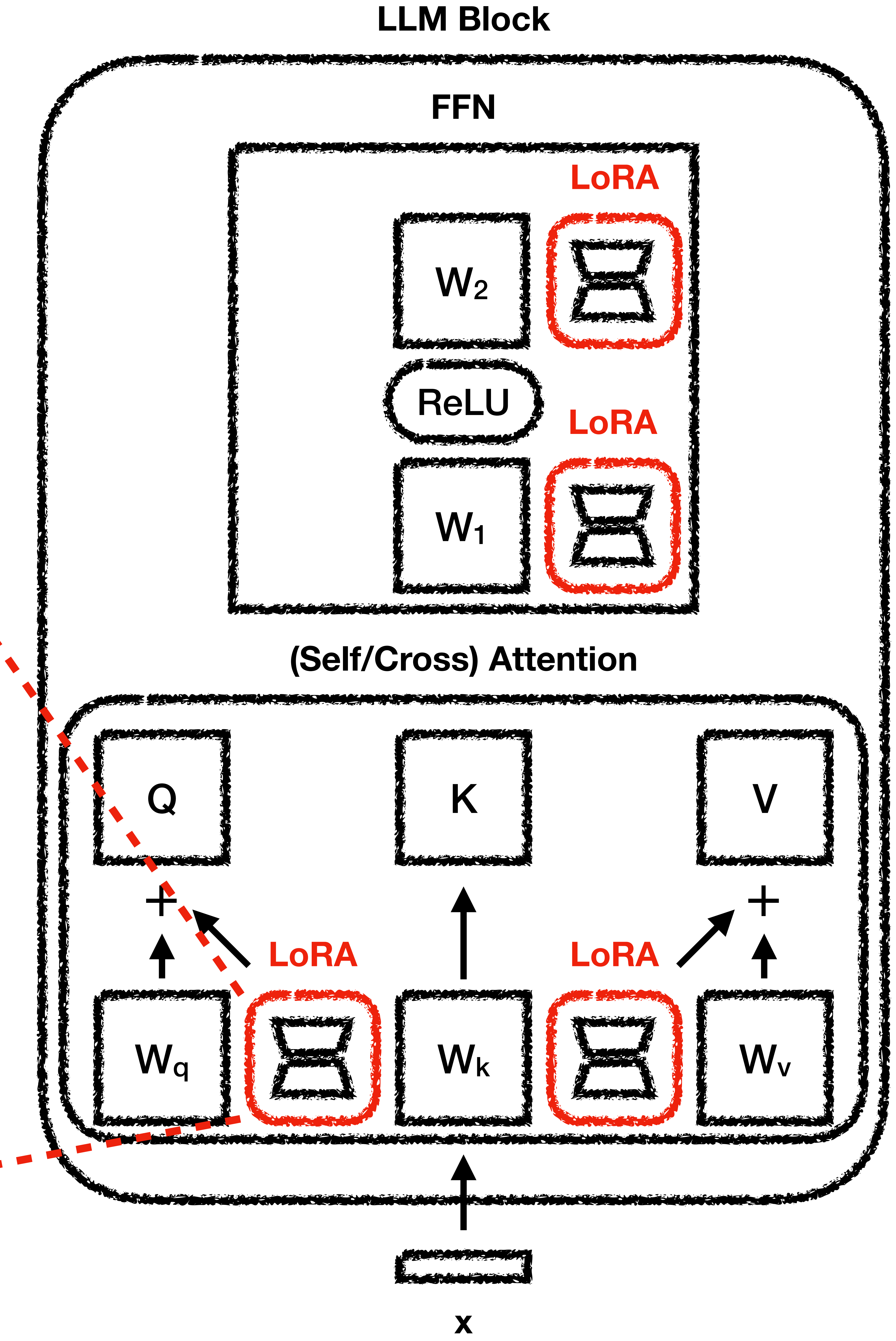
$h, h' \in \mathbb{R}^{1\times k}$

$W \in \mathbb{R}^{d\times k}$

$W_{down} \in \mathbb{R}^{d\times r}$

$W_{up} \in \mathbb{R}^{r\times k}$



# Re-parameterized PEFT

## Low Rank Adaptation (Hu et al., 2021)

$$h \leftarrow xW + \frac{a}{r} \cdot xW_{down}W_{up}$$

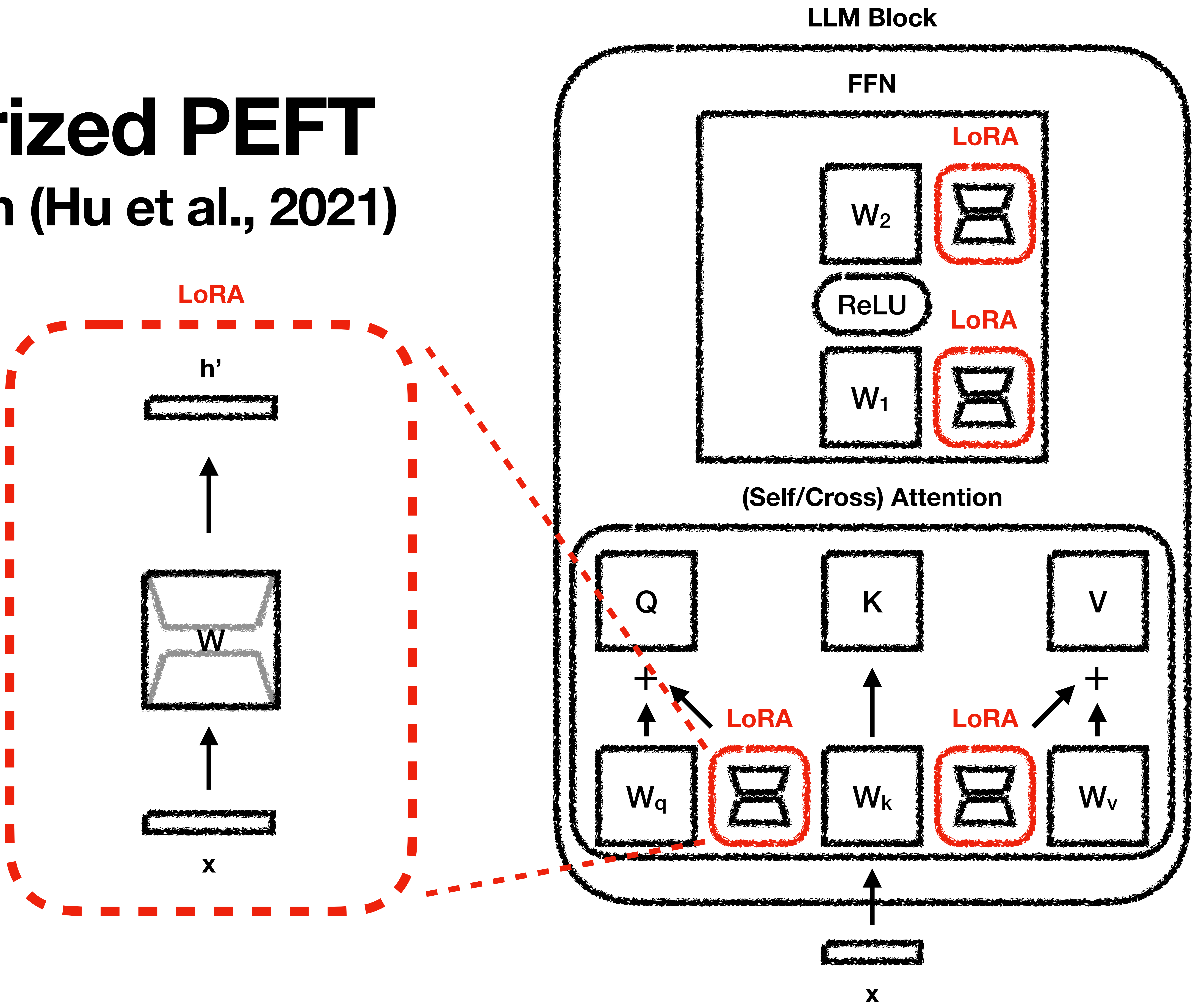
$x \in \mathbb{R}^{1\times d}$

$h, h' \in \mathbb{R}^{1\times k}$

$W \in \mathbb{R}^{d\times k}$

$W_{down} \in \mathbb{R}^{d\times r}$

$W_{up} \in \mathbb{R}^{r\times k}$




DASlab @ Harvard SEAS

# Re-parameterized PEFT

## Low Rank Adaptation (Hu et al., 2021)

$$h \leftarrow xW + \frac{a}{r} \cdot xW_{down}W_{up}$$

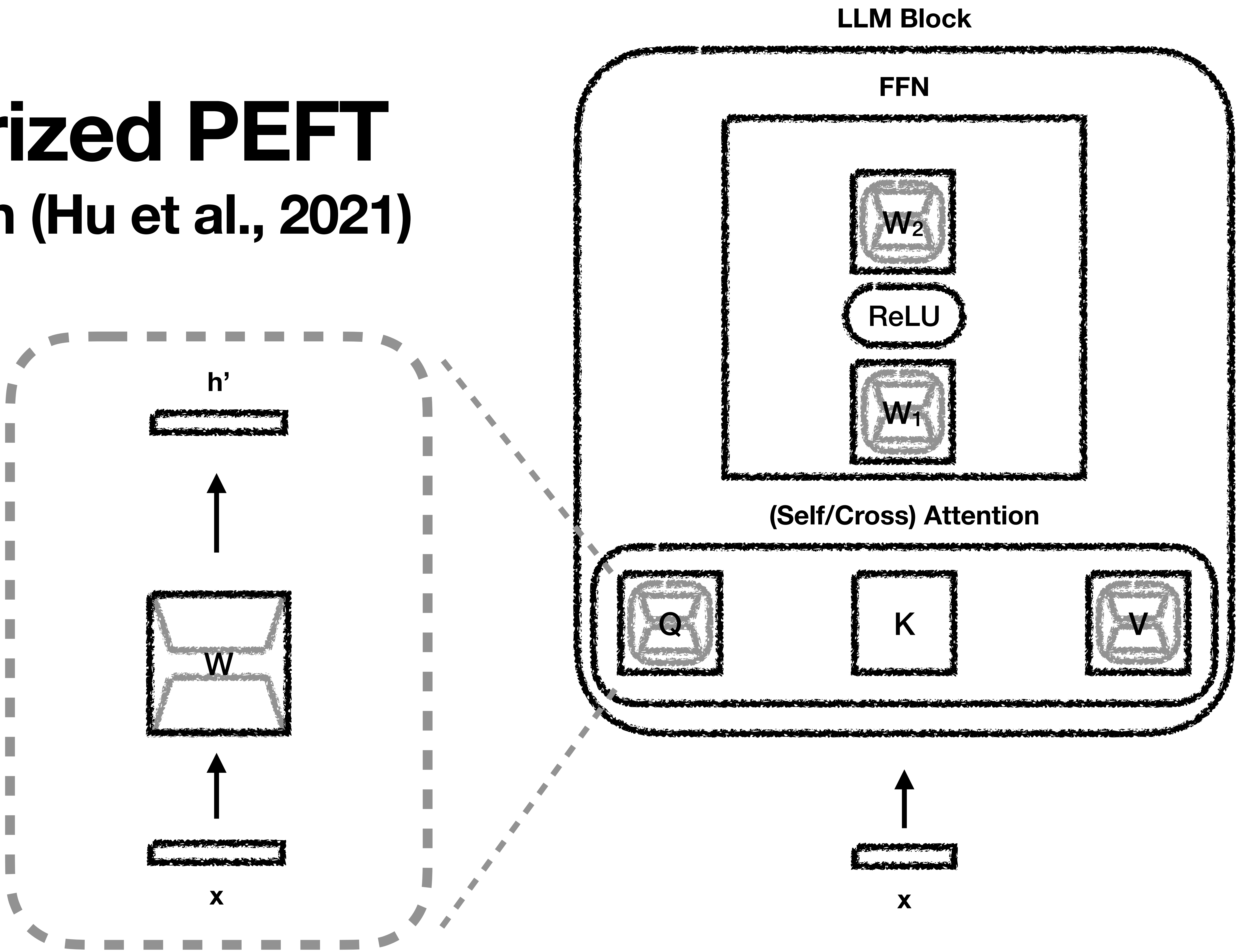
$x \in \mathbb{R}^{1 \times d}$

$h, h' \in \mathbb{R}^{1 \times k}$

$W \in \mathbb{R}^{d \times k}$

$W_{down} \in \mathbb{R}^{d \times r}$

$W_{up} \in \mathbb{R}^{r \times k}$




DASlab @ Harvard SEAS

# PEFT Taxonomy

Full Fine-Tuning
(FFT)

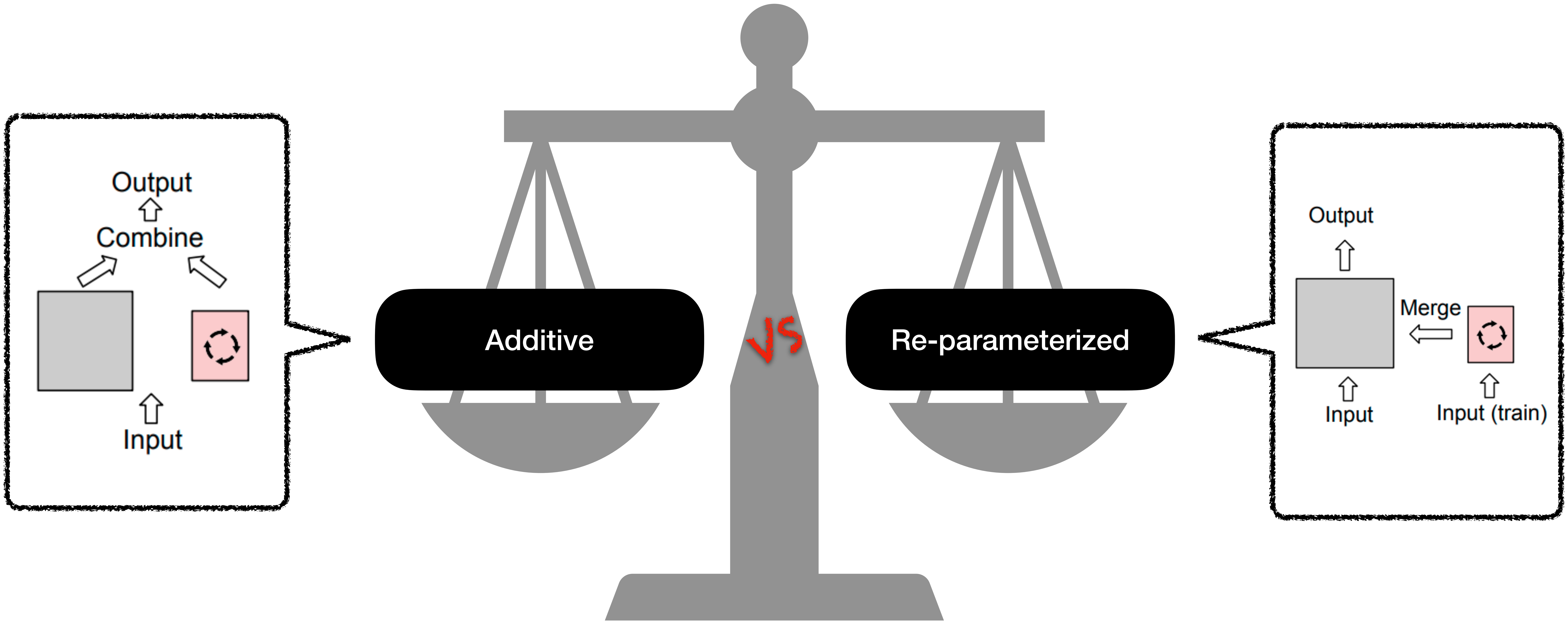
vs

Parameter Efficient Fine-Tuning
(PEFT)

- Selective
- Additive
  - Re-parameterized

DASlab
@ Harvard SEAS

# Which one is better from a systems perspective ?



# PEFT Taxonomy

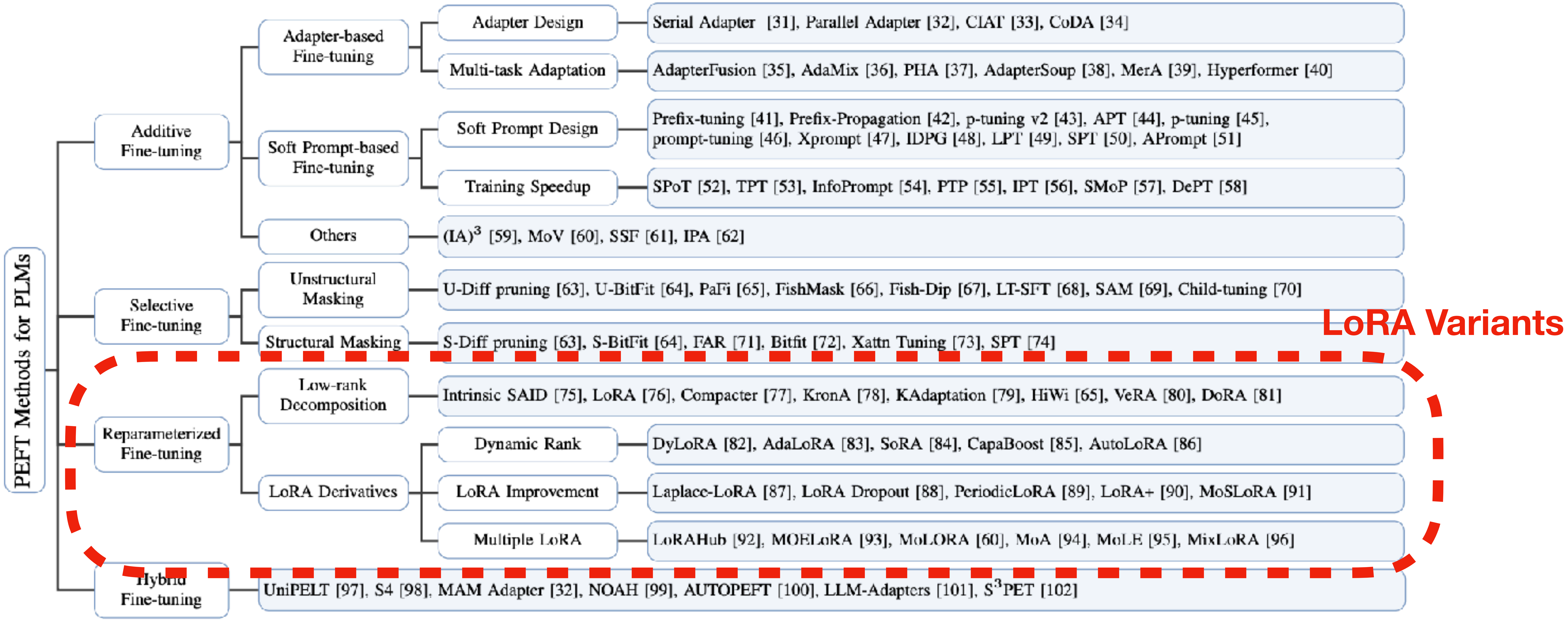



Fig. 3: Taxonomy of Parameter-Efficient Fine-Tuning Methods for Large Models.



(2024, arXiv) Parameter-Efficient Fine-Tuning (PEFT) for Large Models: A Comprehensive Survey

# LoRA Variants

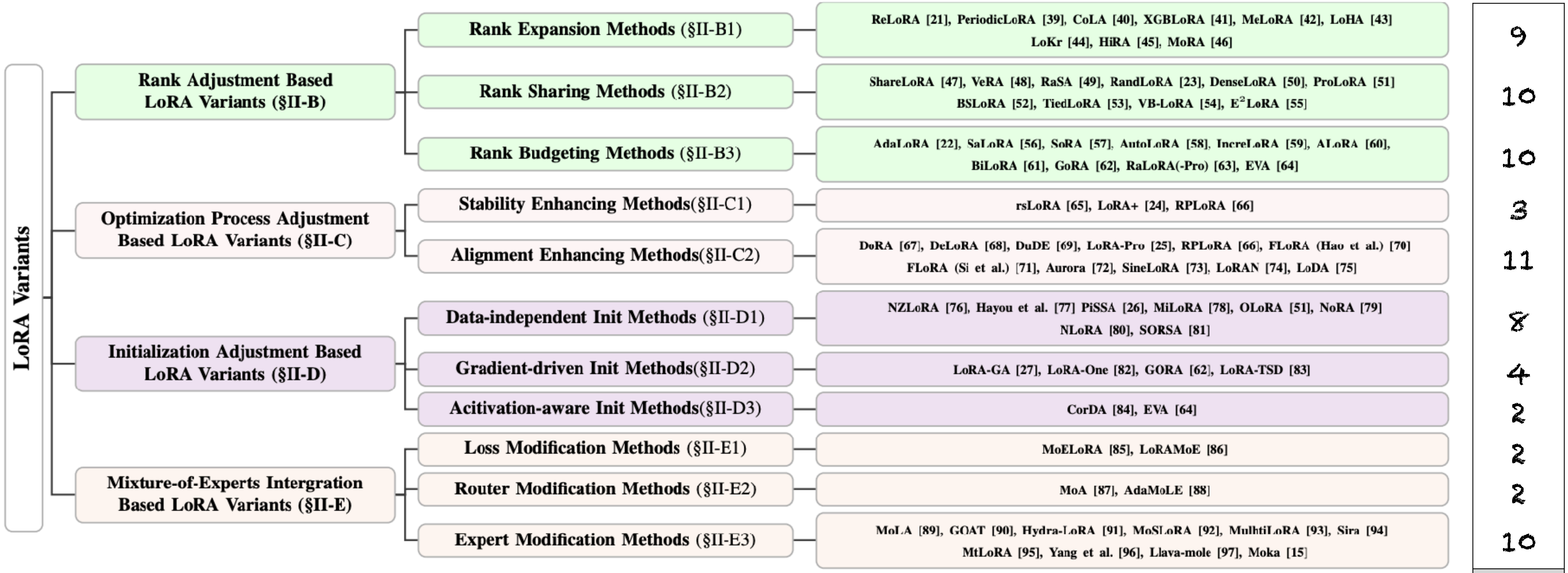



Fig. 1. Hierarchical taxonomy of LoRA variants based on four core principle operational axes.

71



(2025, arXiv) PEFT A2Z: Parameter-Efficient Fine-Tuning Survey for Large Language and Vision Models

## 2. Parameter Budget Allocation in Parameter Efficient Fine-Tuning

**PROBLEM**

Given a fixed parameter budget, how should LoRA adapters be allocated across the layers and modules of an LLM to maximize task performance?

*The design space is combinatorially large — methods distribute them uniformly or dynamically*

# 2. Parameter Budget Allocation in Parameter Efficient Fine-Tuning

Published as a conference paper at ICLR 2021

## More or Less: When and How to Build Convolutional Neural Network Ensembles

**Abdul Wasay**
Harvard University
awasay@seas.harvard.edu

**Stratos Idreos**
Harvard University
stratos@seas.harvard.edu

### Abstract

Convolutional neural networks are utilized to solve increasingly more complex problems and with more data. As a result, researchers and practitioners seek to scale the representational power of such models by adding more parameters. However, increasing parameters requires additional critical resources in terms of memory and compute, leading to increased training and inference cost. Thus a consistent challenge is to obtain as high as possible accuracy within a parameter budget. As neural network designers navigate this complex landscape, they are guided by conventional wisdom that is informed from past empirical studies. We identify a critical part of this design space that is not well-understood: How to decide between the alternatives of expanding a single convolutional network model or increasing the number of networks in the form of an ensemble. We study this question in detail across various network architectures and data sets. We build an extensive experimental framework that captures numerous angles of the possible design space in terms of how a new set of parameters can be used in a model. We consider a holistic set of metrics such as training time, inference time, and memory usage. The framework provides a robust assessment by making sure it controls for the number of parameters. Contrary to conventional wisdom, we show that when we perform a holistic and robust assessment, we uncover a wide design space, where ensembles provide better accuracy, train faster, and deploy at speed comparable to single convolutional networks with the same total number of parameters.

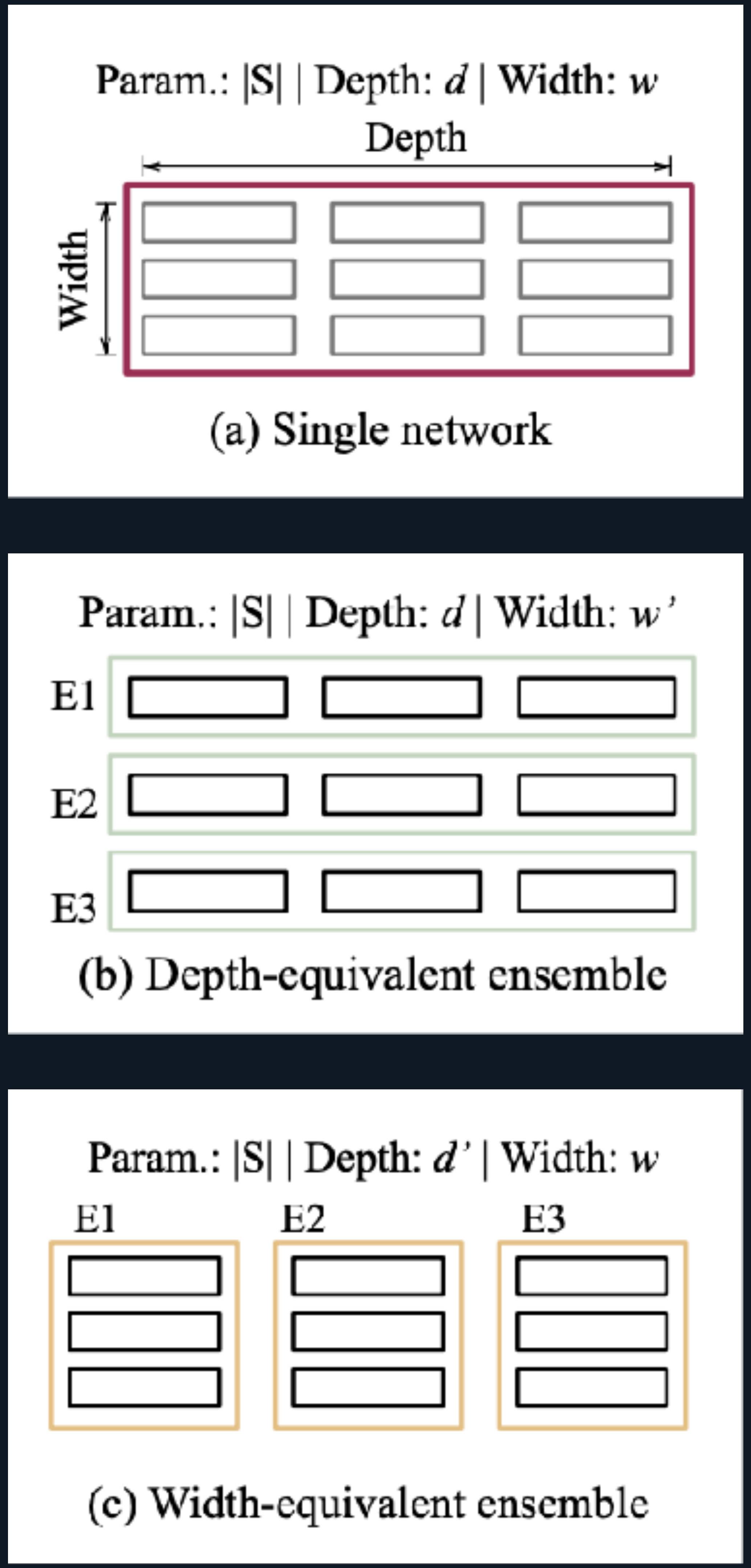
### 1 Introduction

**Scaling capacity of deep learning models.** Convolutional neural network models are becoming as accurate as humans on perceptual tasks. They are now used in numerous and diverse applications such as drug discovery, data compression, and automating gameplay. These models increasingly grow in size with more parameters and layers, driven by two major trends. First, there is a continuous rise in data complexity and sizes in many applications (Shazeer et al., 2017). Second, there is an increasing need for higher accuracy as models are utilized in more critical applications – such as self-driving cars and medical diagnosis (Grzywaczewski, 2017). This effect is especially pronounced in computer vision and natural language processing: Model sizes are three orders of magnitude larger than they were just three years ago (Sanh et al., 2019).

With bigger model sizes, the time, computation, and memory needed to train and deploy such models also increase. Thus, it is a consistent challenge to design models that maximize accuracy while remaining practical with respect to the resources they need (Lee et al., 2015; Huang et al., 2017b). In this paper, we study the following question: Given a number of parameters (neurons), how to design a convolutional neural network to optimize holistically for accuracy, training cost, and inference cost?

**The holistic design space is very complex.** Designers of convolutional neural network models navigate a complex design landscape to address this question: First, they need to decide on network architecture. Then, they have to consider whether to use a single network or build an ensemble model with multiple networks. Additionally, they have to decide how many neural networks to use and their individual designs, i.e., the depth, width, and number of networks in their model. Modern






(a) Single network

(b) Depth-equivalent ensemble

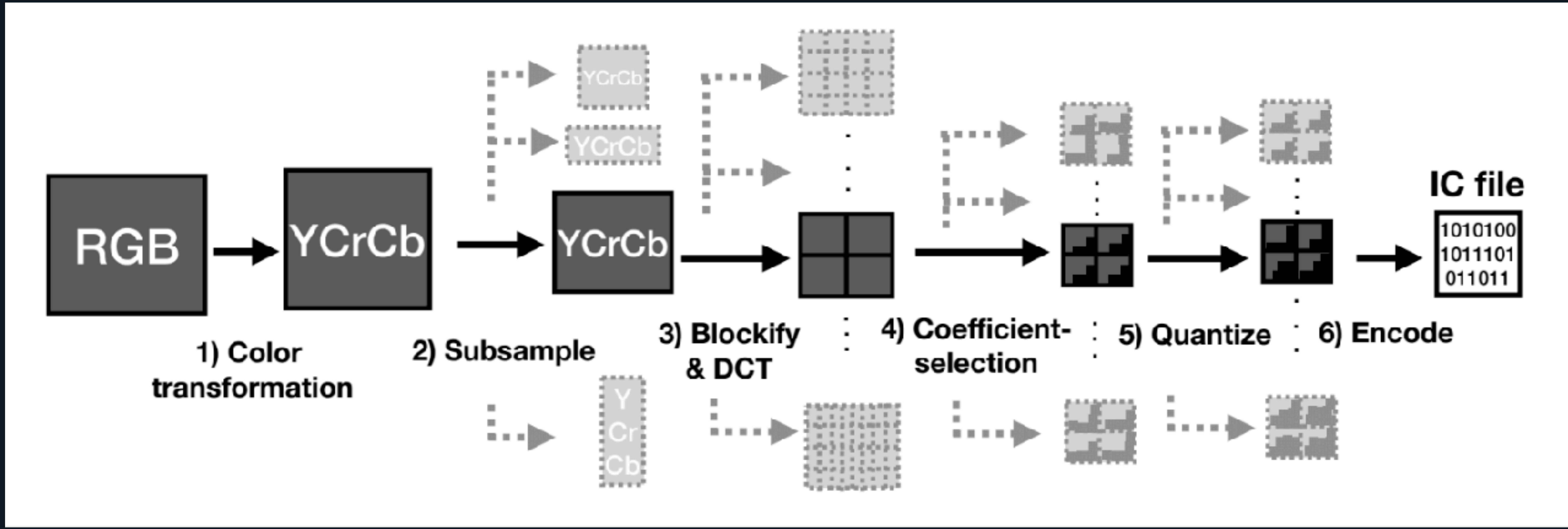
(c) Width-equivalent ensemble

## 2. Parameter Budget Allocation in Parameter Efficient Fine-Tuning

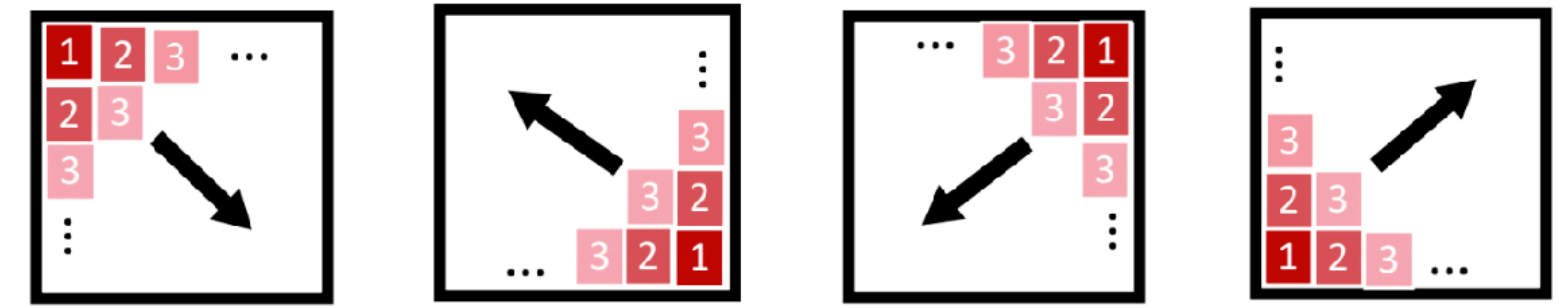
**SOLUTION**

Formally define the primitives of the LoRA placement design space and identify good domains to restrict the otherwise infinite design space

*Generalize across models and domains — find simple allocation heuristics*




# Image Calculator Throwback



(a) Strategy 1: Increasing-sized upper-left triangle (b) Strategy 2: Increasing-sized lower-right triangle (c) Strategy 3: Increasing-sized upper-right triangle (d) Strategy 4: Increasing-sized lower-left triangle

DASlab @ Harvard SEAS

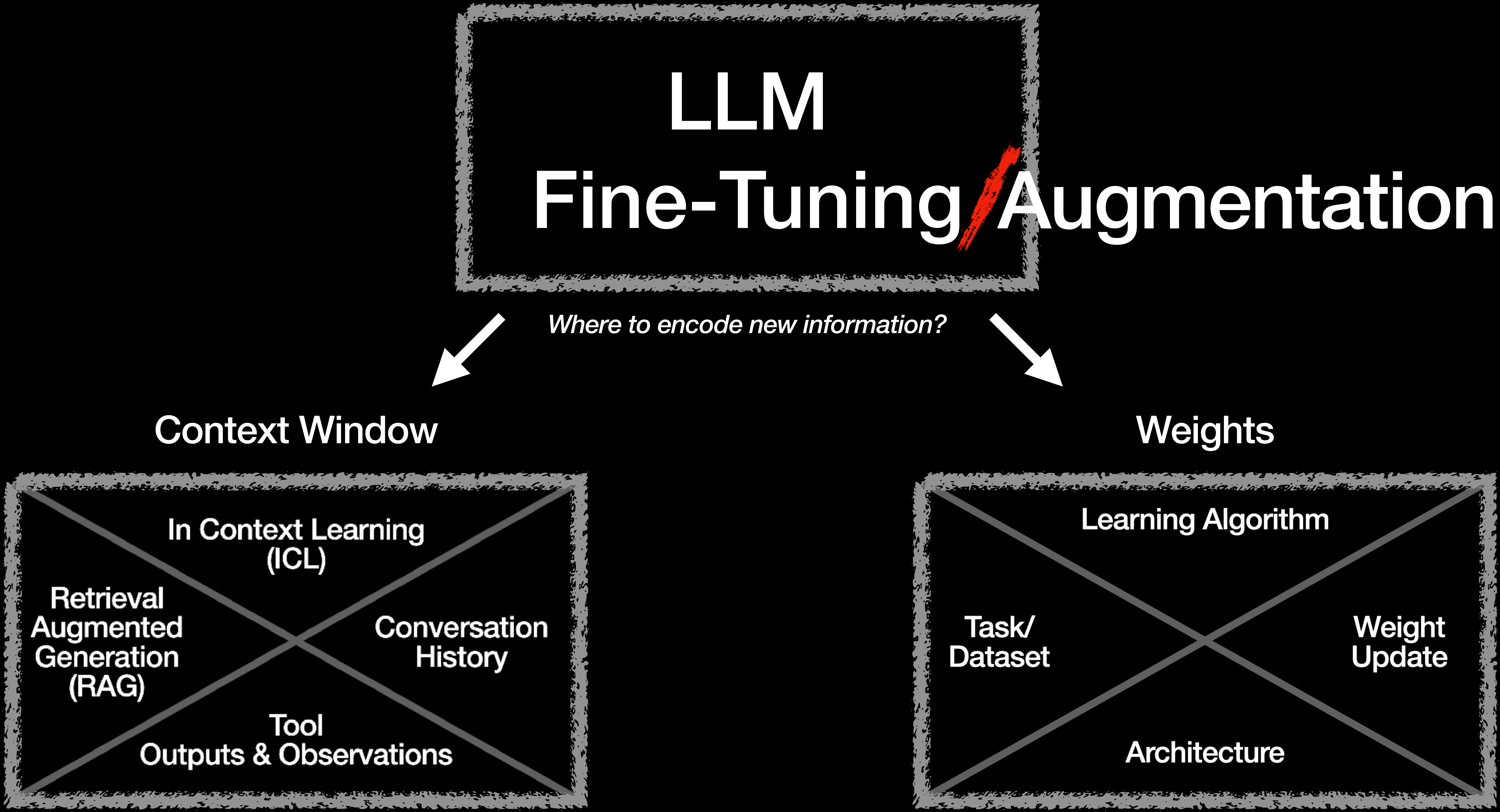
Learning Algorithm

Task/
Dataset

LLM
Fine-Tuning

Weight
Update

Architecture


LLM
Fine-Tuning/Augmentation
Where to encode new information?
Context Window
In Context Learning (ICL)
Retrieval Augmented Generation (RAG)
Conversation History
Tool Outputs & Observations
Weights
Learning Algorithm
Task/ Dataset
Weight Update
Architecture