Large Language Models (in 2023)

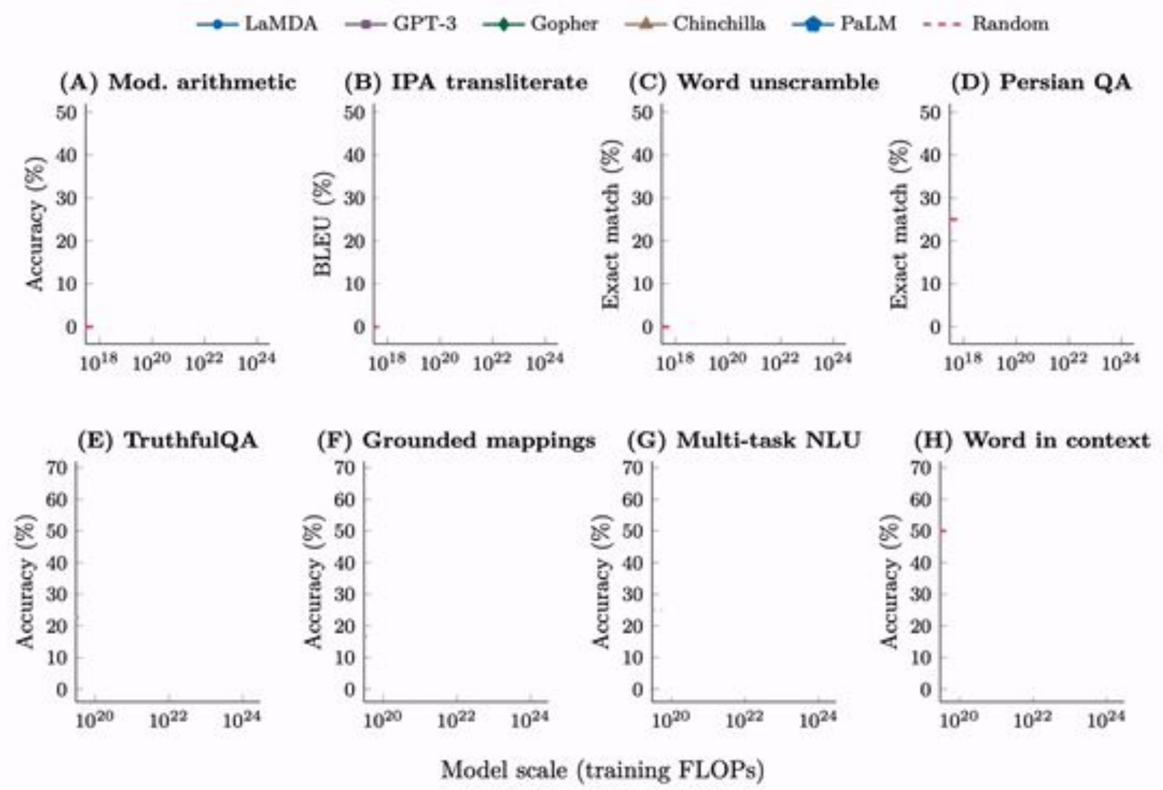
Hyung Won Chung

OpenAI

Twitter: @hwchung27

Some abilities emerge with scale

Having the right perspective is crucial



[Emergent Abilities of Large Language Models]
Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph et al. (2022)

Perspective of “yet”

Perspective of “yet”

This idea doesn’t work

Perspective of “yet”

This idea doesn’t work → This idea doesn’t work *yet*

Why is the perspective of “yet” not so obvious?

We are used to operating in an environment where underlying axioms don’t change

You run an experiment for your new scientific idea. It doesn’t work now. You know that it will not work if you run 3 years later

For language models, the most capable model serves as an “axiom” for many research experiments run on top

Need for constant unlearning

Many ideas get outdated and invalidated at larger scale

We need to constantly unlearn intuitions built on such invalidated ideas

With less to unlearn, newcomers can have advantages over more experienced ones. This is an interesting neutralizing force

Going ahead of the scaling curve

Document experiments that failed because of insufficient “intelligence”

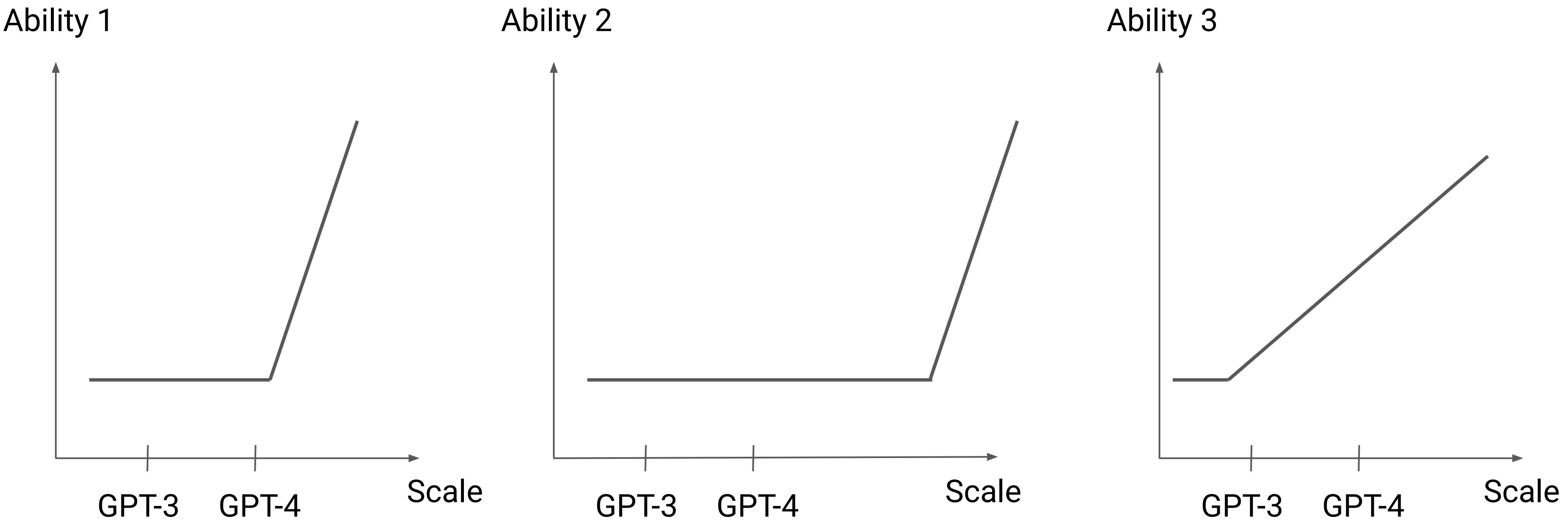
Do not declare failure yet and make it easy to rerun in the future

As soon as the new model comes out, rerun them

Learn what works and what doesn’t

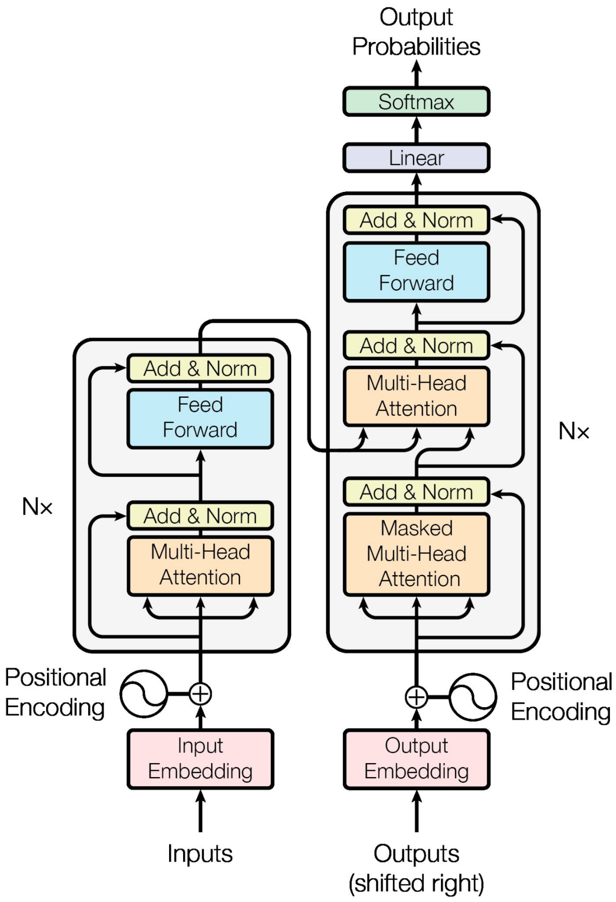
Update your intuition on emergent abilities and scale

Highly simplified view of emergent abilities

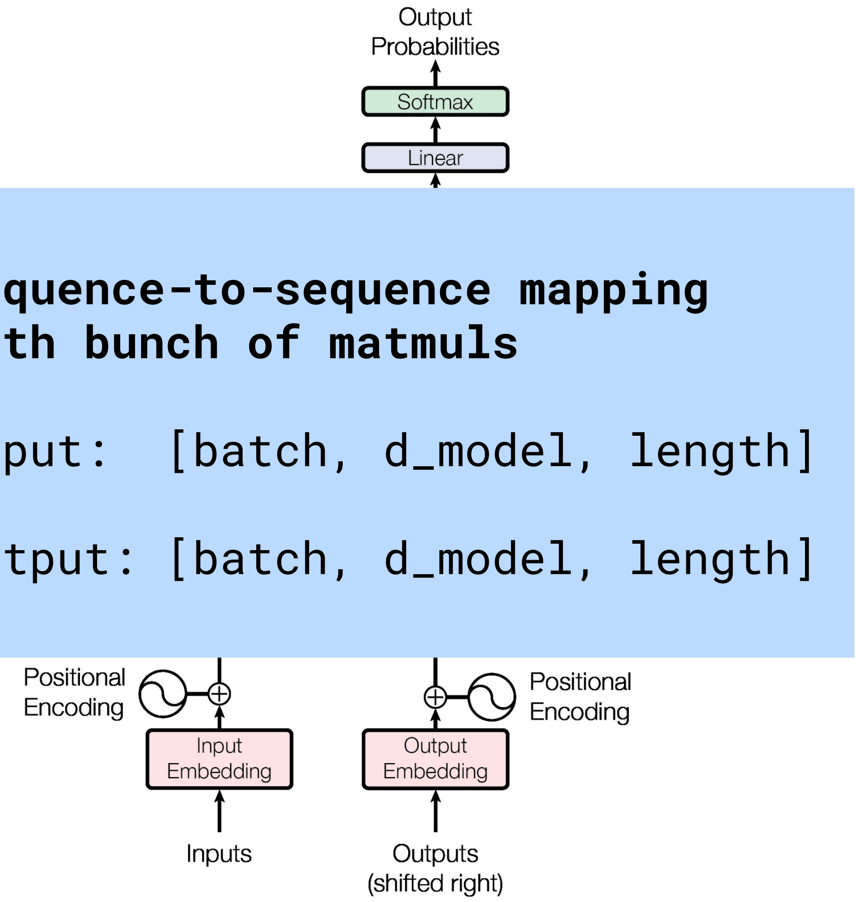


How is the scaling actually done?

All LLMs so far use Transformer architecture



Let’s take a “functional” viewpoint on the Transformer



Sequence-to-sequence mapping with bunch of matmuls

```
Input:  [batch, d_model, length]

Output: [batch, d_model, length]
```

Process	**Shape**
“Many words don't map to one token: indivisible.”	[ ]

Process

“Many words don't map to one token: indivisible.”

⬇ Tokenization

`Unicode characters like emojis may be split.`

`[7085, 2456, 836, 470, 3975, 284, 530, 11241, 25, 773, 452, 12843, 13]`

Shape

[ ]

⬇

`[length]`

Process

“Many words don't map to one token: indivisible.”

[Tokenization](#)

`Unicode characters like emojis may be split.`

`[7085, 2456, 836, 470, 3975, 284, 530, 11241, 25, 773, 452, 12843, 13]`

Embedding

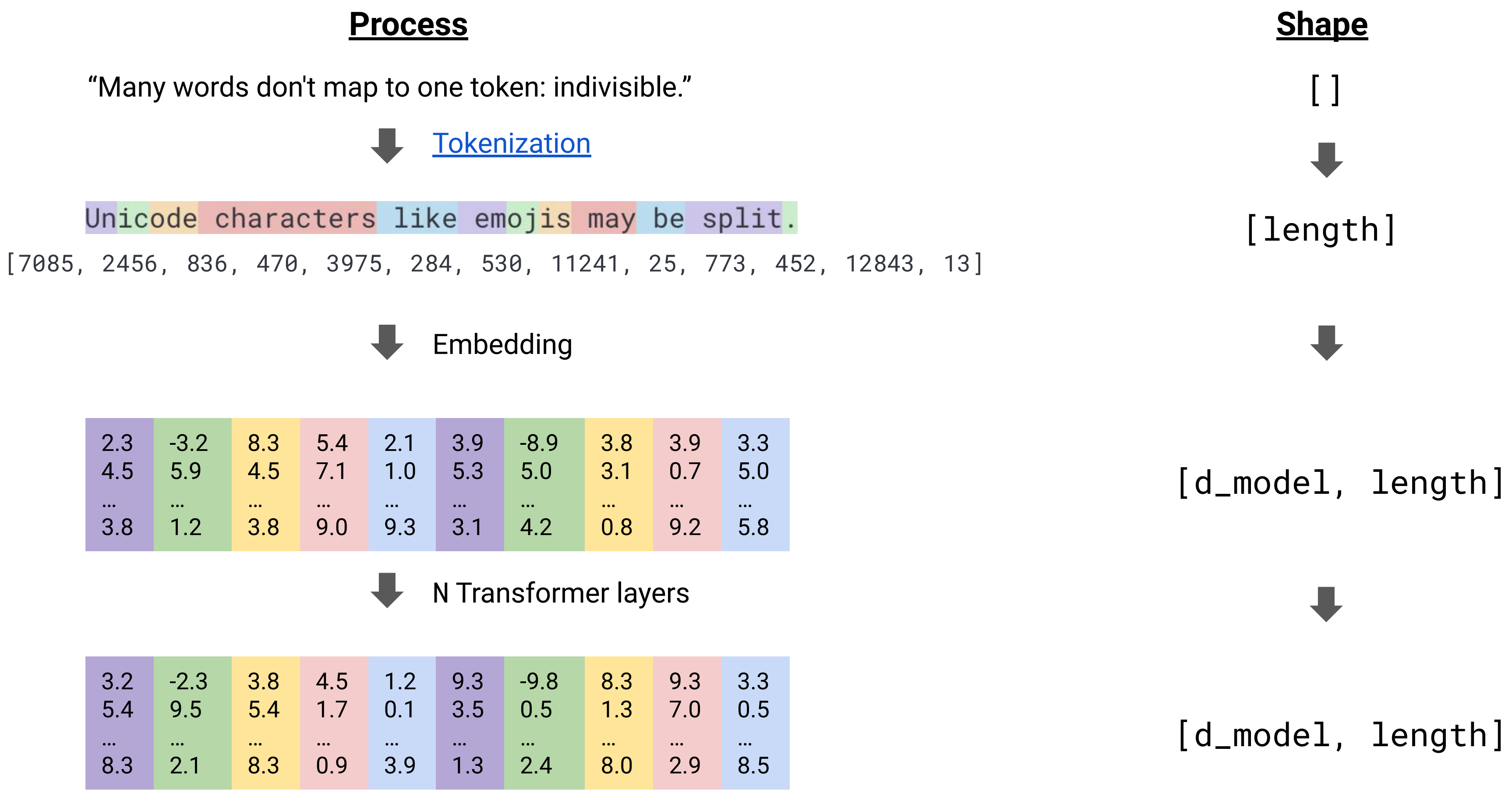
2.3	-3.2	8.3	5.4	2.1	3.9	-8.9	3.8	3.9	3.3
4.5	5.9	4.5	7.1	1.0	5.3	5.0	3.1	0.7	5.0
…	…	…	…	…	…	…	…	…	…
3.8	1.2	3.8	9.0	9.3	3.1	4.2	0.8	9.2	5.8

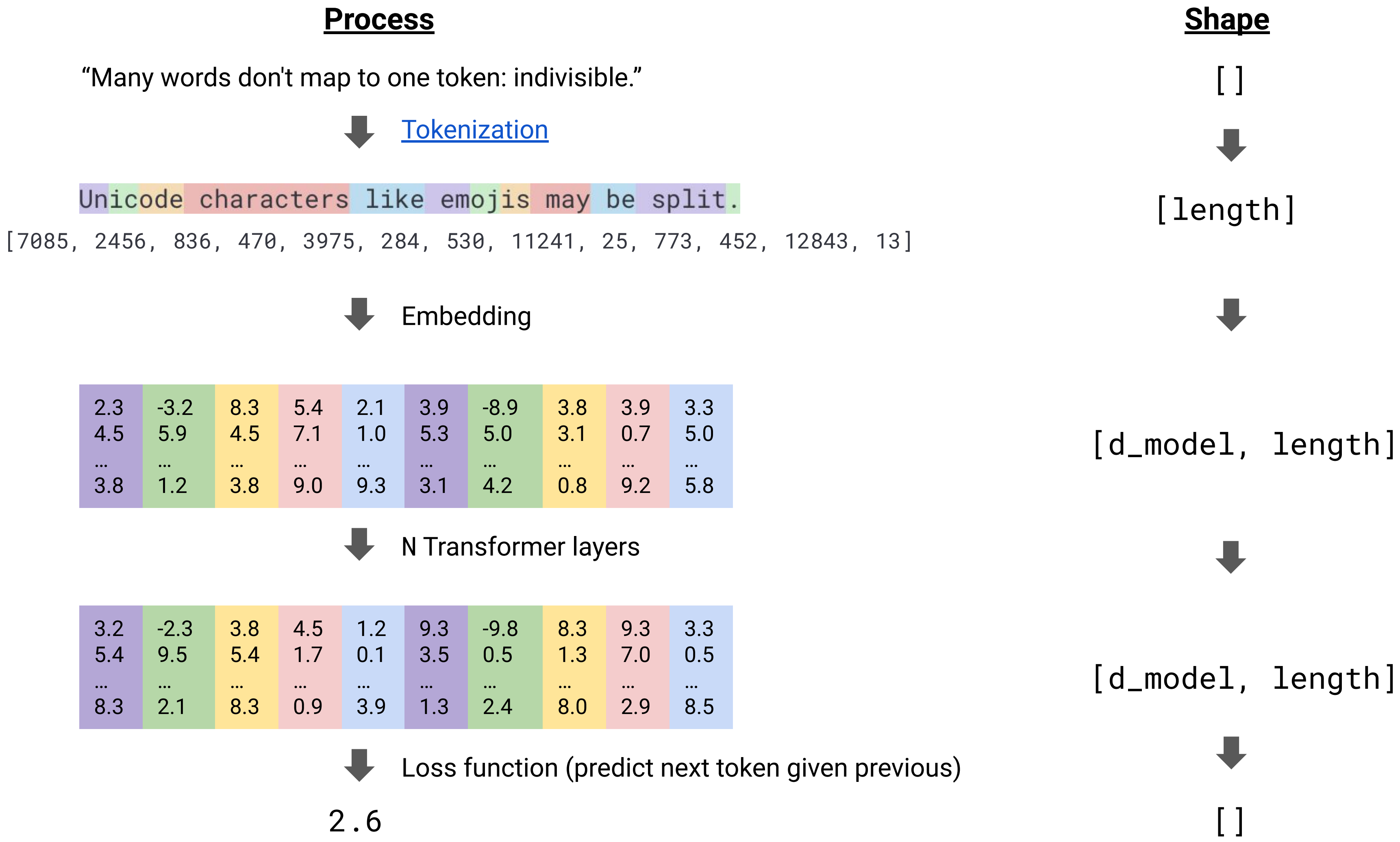
Shape

`[ ]`

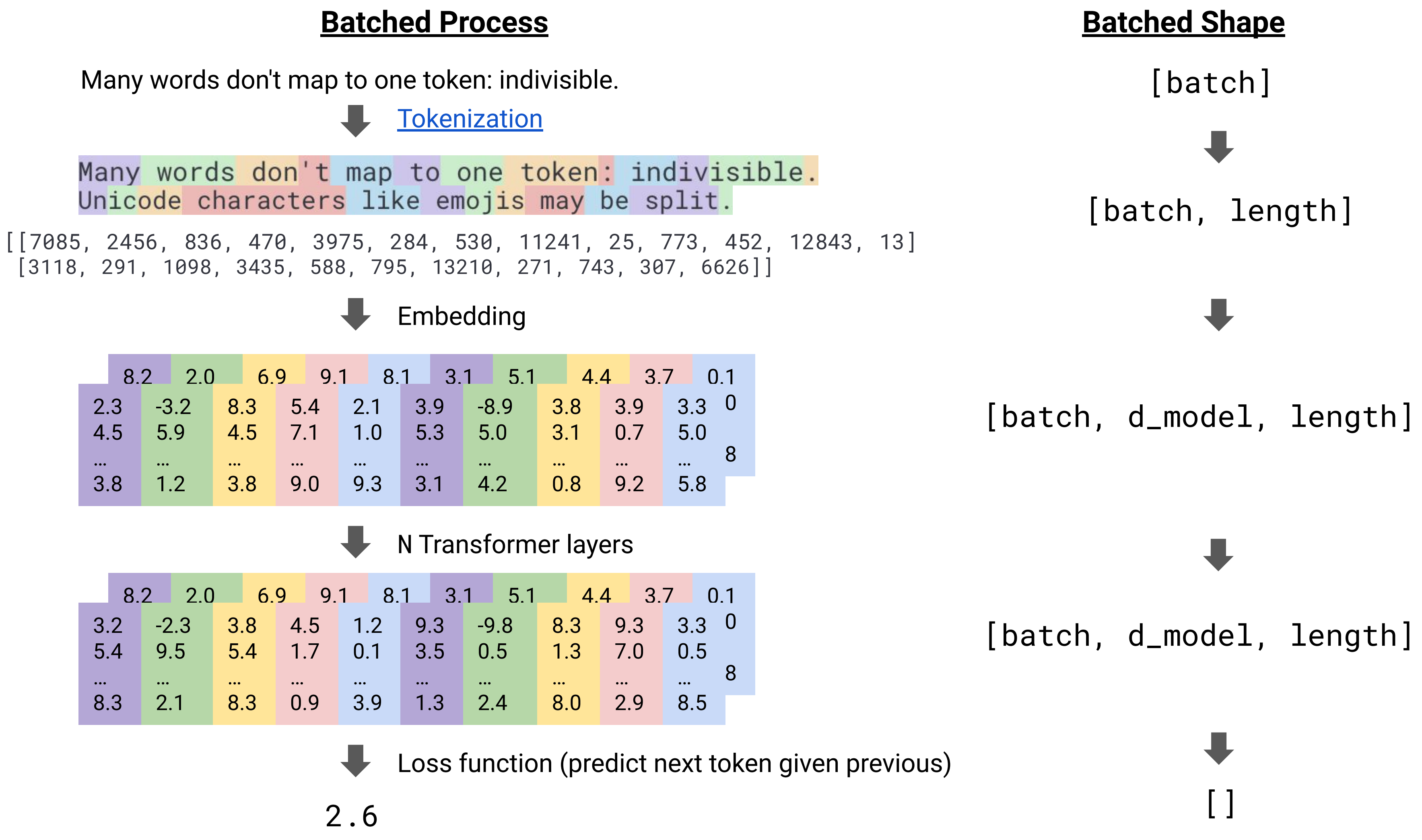
`[length]`

`[d_model, length]`

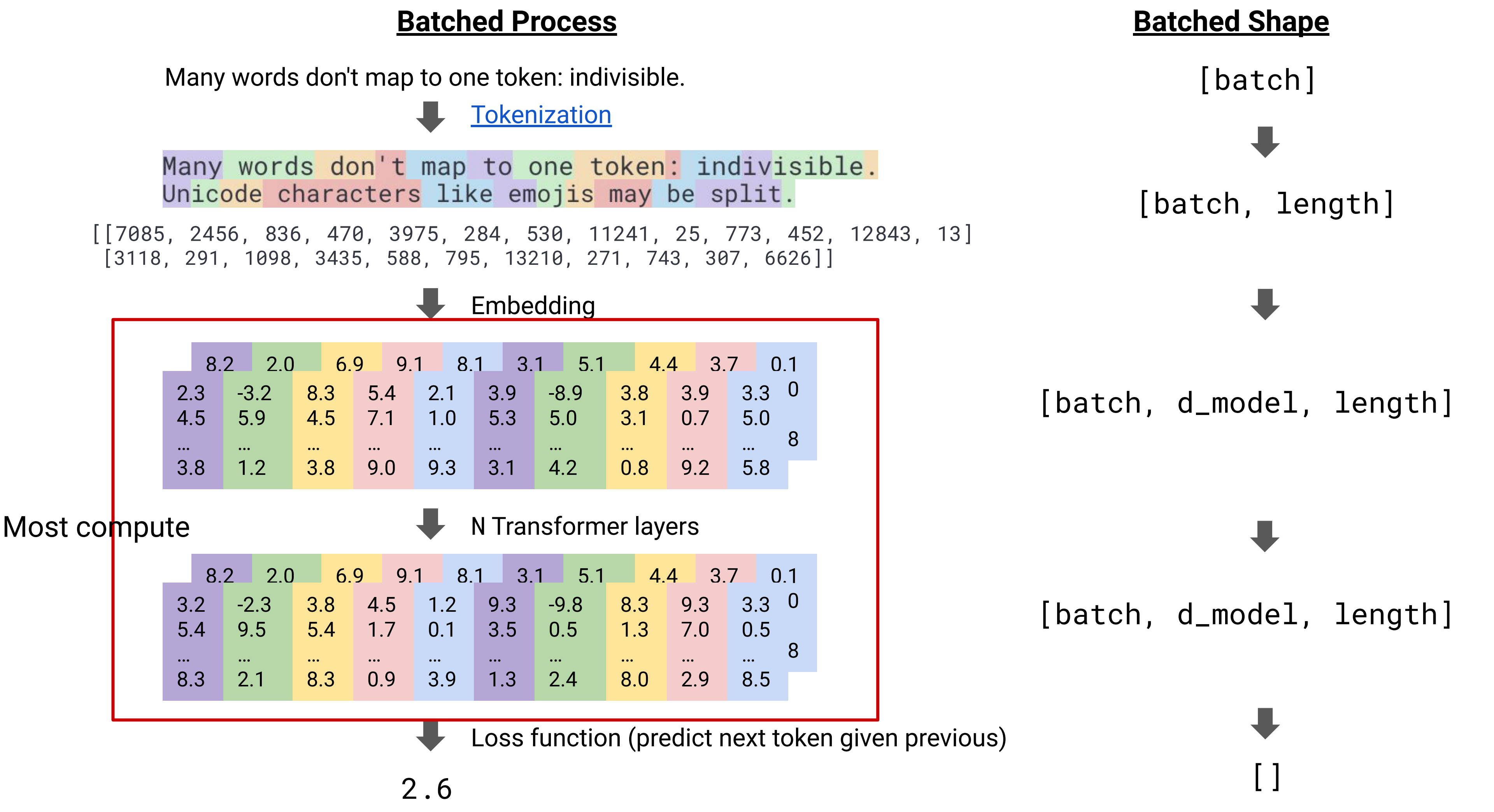

Process
Shape
“Many words don't map to one token: indivisible.”
[ ]
Tokenization
Unicode characters like emojis may be split.
[length]
[7085, 2456, 836, 470, 3975, 284, 530, 11241, 25, 773, 452, 12843, 13]
Embedding
2.3 4.5 … 3.8
-3.2 5.9 … 1.2
8.3 4.5 … 3.8
5.4 7.1 … 9.0
2.1 1.0 … 9.3
3.9 5.3 … 3.1
-8.9 5.0 … 4.2
3.8 3.1 … 0.8
3.9 0.7 … 9.2
3.3 5.0 … 5.8
[d_model, length]
N Transformer layers
3.2 5.4 … 8.3
-2.3 9.5 … 2.1
3.8 5.4 … 8.3
4.5 1.7 … 0.9
1.2 0.1 … 3.9
9.3 3.5 … 1.3
-9.8 0.5 … 2.4
8.3 1.3 … 8.0
9.3 7.0 … 2.9
3.3 0.5 … 8.5
[d_model, length]



Process
Shape
“Many words don't map to one token: indivisible.”
[ ]
Tokenization
Unicode characters like emojis may be split.
[7085, 2456, 836, 470, 3975, 284, 530, 11241, 25, 773, 452, 12843, 13]
[length]
Embedding
2.3 4.5 … 3.8
-3.2 5.9 … 1.2
8.3 4.5 … 3.8
5.4 7.1 … 9.0
2.1 1.0 … 9.3
3.9 5.3 … 3.1
-8.9 5.0 … 4.2
3.8 3.1 … 0.8
3.9 0.7 … 9.2
3.3 5.0 … 5.8
[d_model, length]
N Transformer layers
3.2 5.4 … 8.3
-2.3 9.5 … 2.1
3.8 5.4 … 8.3
4.5 1.7 … 0.9
1.2 0.1 … 3.9
9.3 3.5 … 1.3
-9.8 0.5 … 2.4
8.3 1.3 … 8.0
9.3 7.0 … 2.9
3.3 0.5 … 8.5
[d_model, length]
Loss function (predict next token given previous)
2.6
[ ]



Batched Process
Batched Shape
Many words don't map to one token: indivisible.
[batch]
Tokenization
Many words don't map to one token: indivisible.
Unicode characters like emojis may be split.
[batch, length]
[[7085, 2456, 836, 470, 3975, 284, 530, 11241, 25, 773, 452, 12843, 13]
[3118, 291, 1098, 3435, 588, 795, 13210, 271, 743, 307, 6626]]
Embedding
8.2 2.0 6.9 9.1 8.1 3.1 5.1 4.4 3.7 0.1
2.3 -3.2 8.3 5.4 2.1 3.9 -8.9 3.8 3.9 3.3 0
4.5 5.9 4.5 7.1 1.0 5.3 5.0 3.1 0.7 5.0
… … … … … … … … … … 8
3.8 1.2 3.8 9.0 9.3 3.1 4.2 0.8 9.2 5.8
[batch, d_model, length]
N Transformer layers
8.2 2.0 6.9 9.1 8.1 3.1 5.1 4.4 3.7 0.1
3.2 -2.3 3.8 4.5 1.2 9.3 -9.8 8.3 9.3 3.3 0
5.4 9.5 5.4 1.7 0.1 3.5 0.5 1.3 7.0 0.5
… … … … … … … … … … 8
8.3 2.1 8.3 0.9 3.9 1.3 2.4 8.0 2.9 8.5
[batch, d_model, length]
Loss function (predict next token given previous)
2.6
[]


Batched Process
Many words don't map to one token: indivisible.
Tokenization
Many words don't map to one token: indivisible.
Unicode characters like emojis may be split.
[[7085, 2456, 836, 470, 3975, 284, 530, 11241, 25, 773, 452, 12843, 13]
[3118, 291, 1098, 3435, 588, 795, 13210, 271, 743, 307, 6626]]
Embedding
Most compute
N Transformer layers
Loss function (predict next token given previous)
2.6
Batched Shape
[batch]
[batch, length]
[batch, d_model, length]
[batch, d_model, length]
[]

From first-principles view

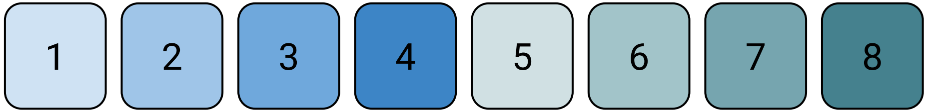
Scaling Transformer means efficiently doing matmuls with many machines

This involves distributing all the matrices (or arrays) involved in the Transformer layer to various machines

Do so while minimizing the communication between machines

Matrix multiplication with multiple machines

We have 8 machines (e.g. 8 GPUs)

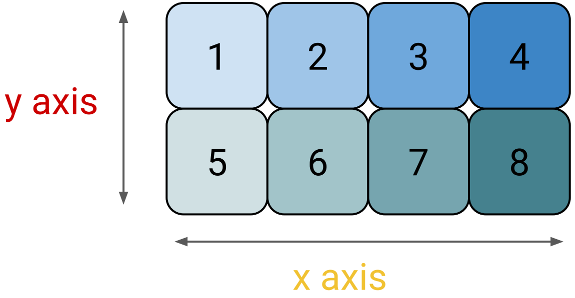


A: [16,16]   B: [16,16]   C: [16,16]

Example adapted from Anselm Levskaya's

Matrix multiplication with multiple machines

8 machines arranged in 2x4 *mesh*

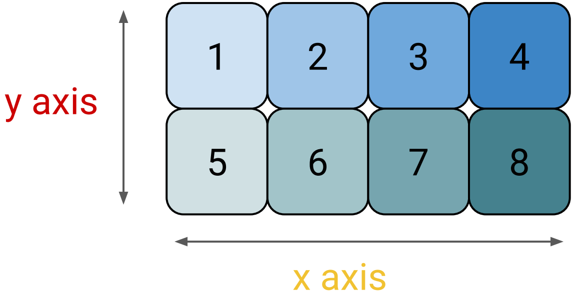


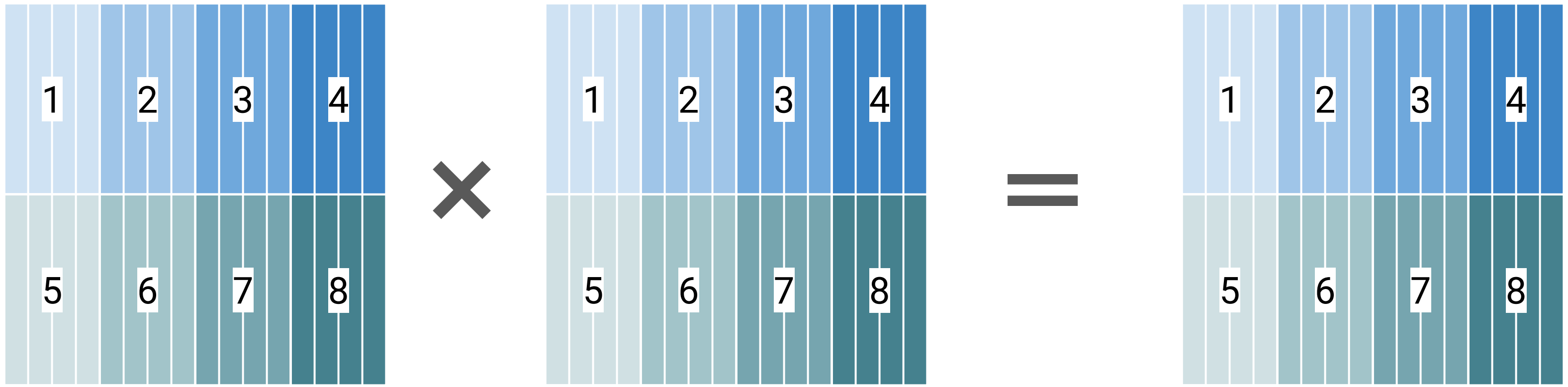
A: [16,16]   B: [16,16]   C: [16,16]

Map each array axis to hardware axis

Matrix multiplication with multiple machines

8 machines arranged in 2x4 *mesh*



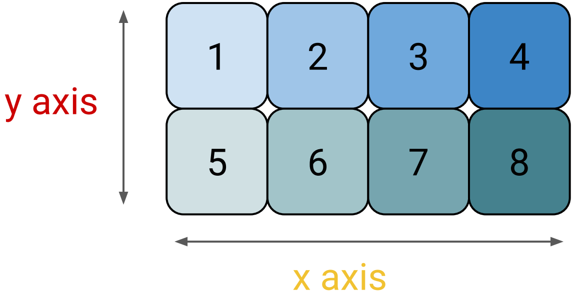


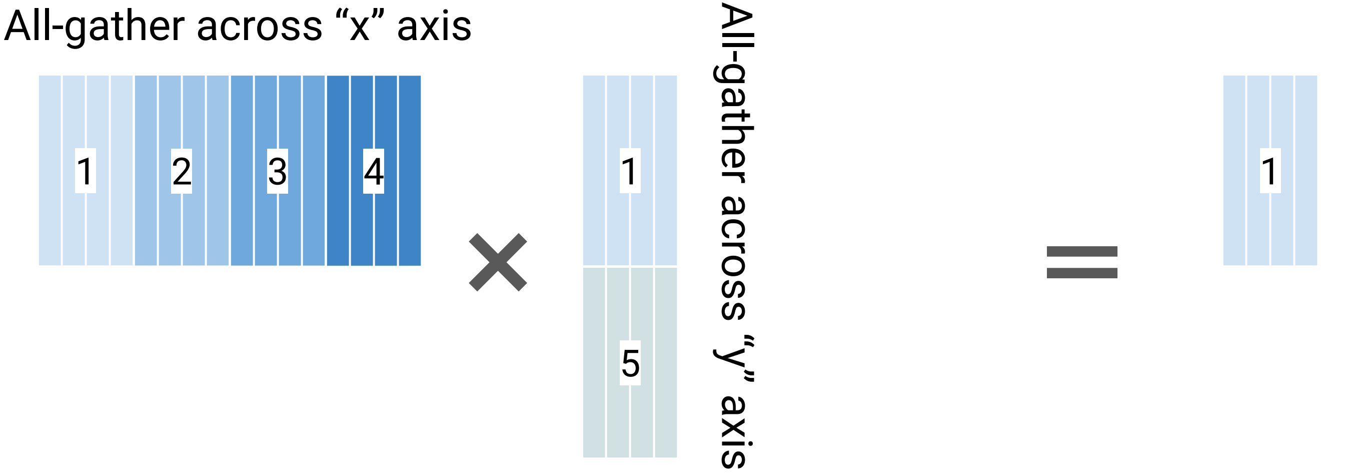
A: $[16_y, 16_x]$ B: $[16_y, 16_x]$ C: $[16_y, 16_x]$

Map each array axis to hardware axis

Let's focus on what machine 1 does

8 machines arranged in 2x4 *mesh*

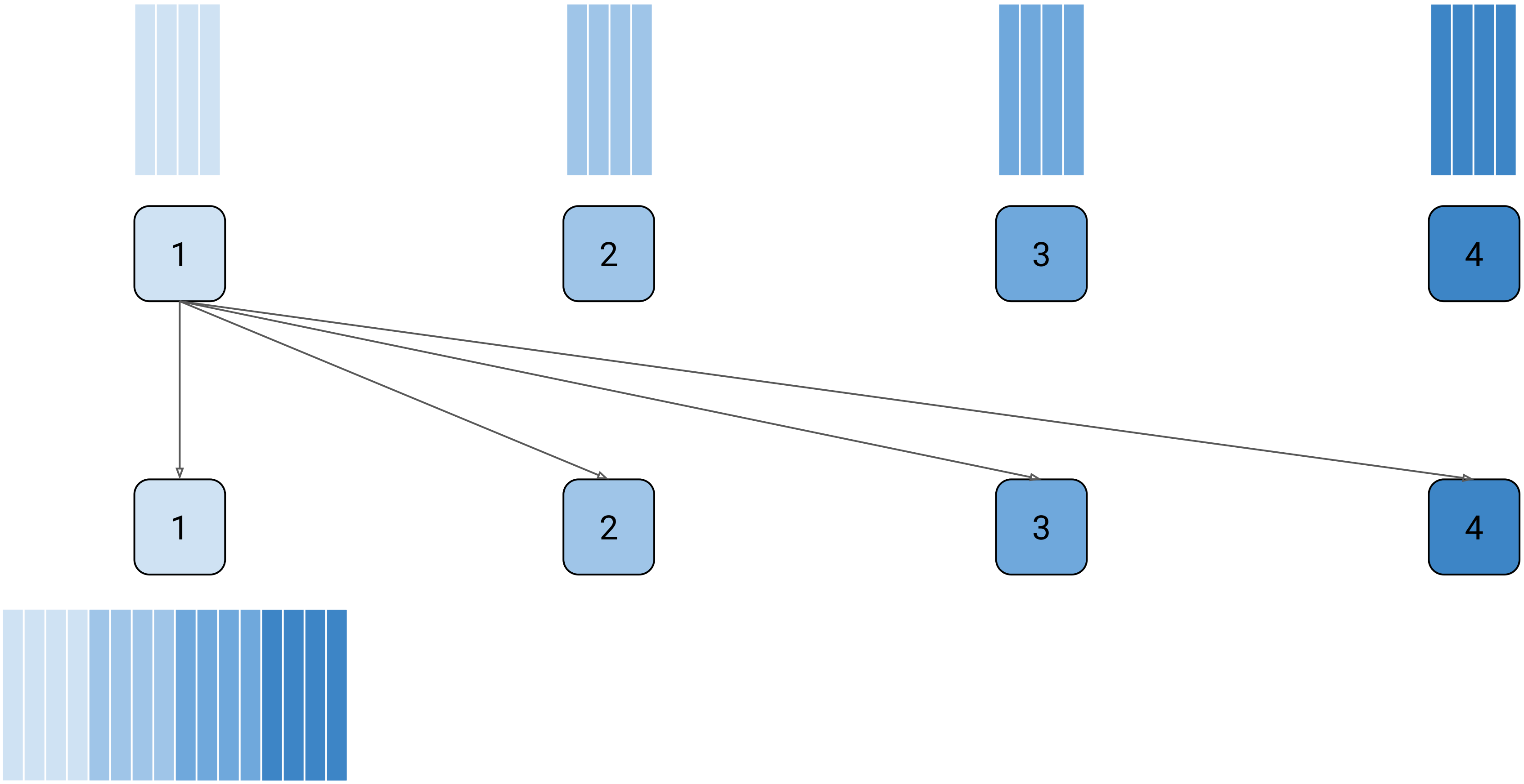




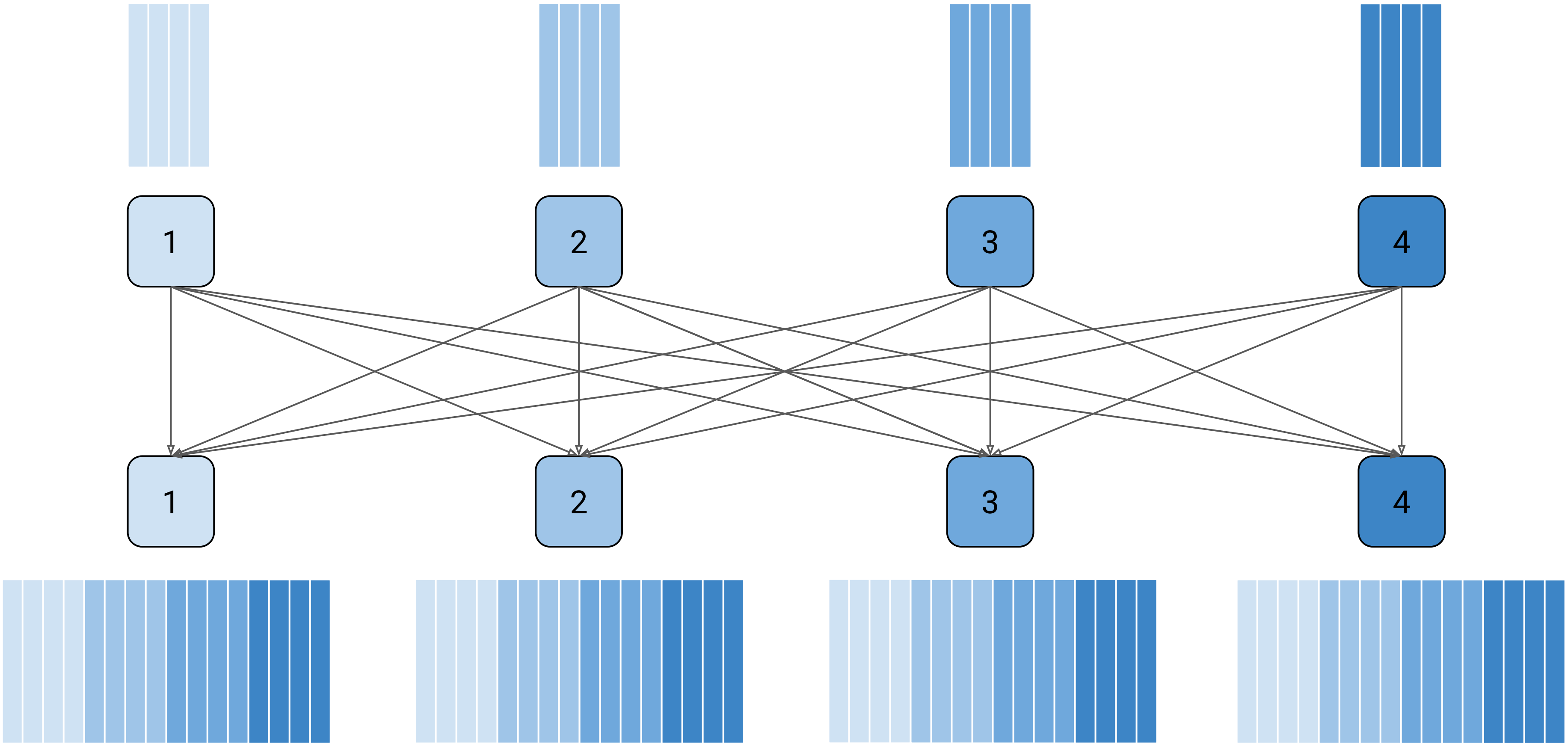
$A: [16_y, 16_x]$ $B: [16_y, 16_x]$ $C: [16_y, 16_x]$

All-gather

All-gather

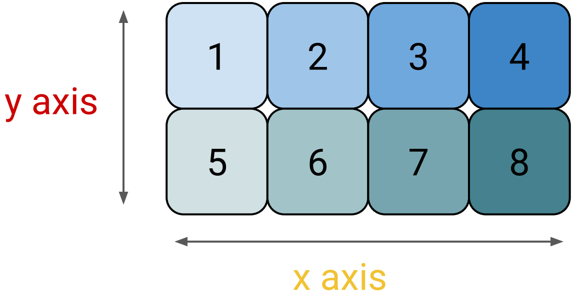


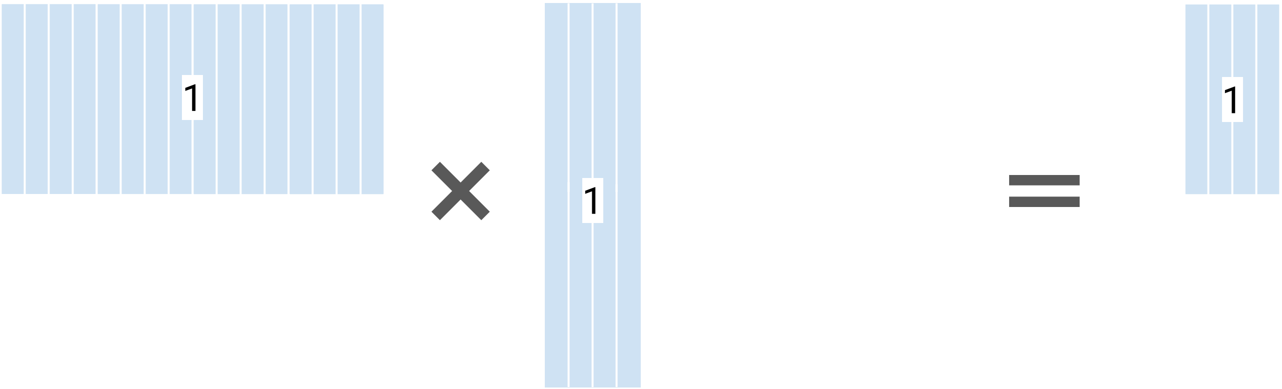
All-gather



Local matmul after all-gather

8 machines arranged in 2x4 *mesh*





A: $[8_y, 16_x]$ B: $[16_y, 4_x]$ C: $[8_y, 4_x]$

Einsum: generalization of matmul

```
np.einsum("i,i->i", a, b) == a * b
np.einsum("i,i->", a, b) == (a * b).sum()
np.einsum("ij,j->i", c, b) == c.dot(b)
```

If a letter appears in both input, multiply component-wise

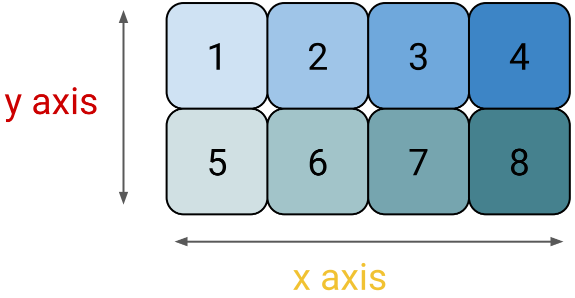
If a letter doesn’t exist on the output, sum over the dimension

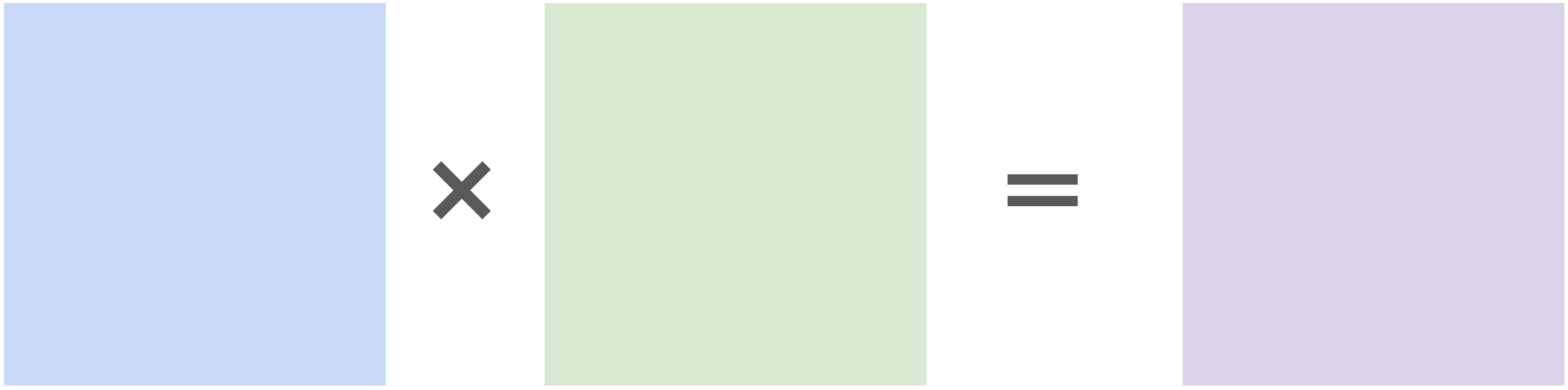
Matmul with Einsum

```python
def matmul(A, B):
    C = einsum("mn,np->mp", A, B)
    return C
```

Matrix multiplication: einsum view

8 machines arranged in 2x4 *mesh*





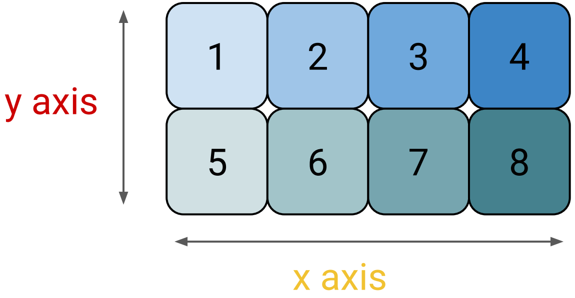
A: [16,16]   B: [16,16]   C: [16,16]

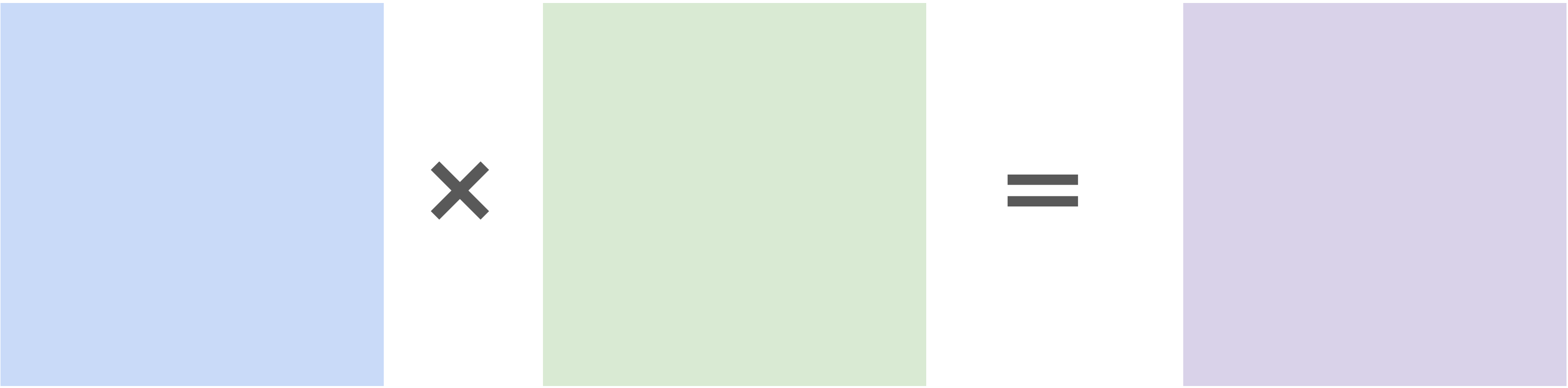
Map each array axis to hardware axis

Matrix multiplication: einsum view

```
def matmul(A, B):
    C = einsum("mn,np->mp", A, B)
    return C
```

8 machines arranged in 2x4 *mesh*





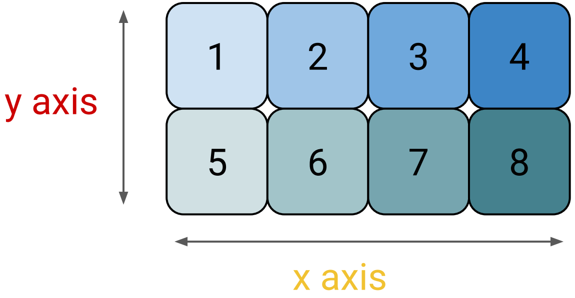
A: [16,16]　　B: [16,16]　　C: [16,16]

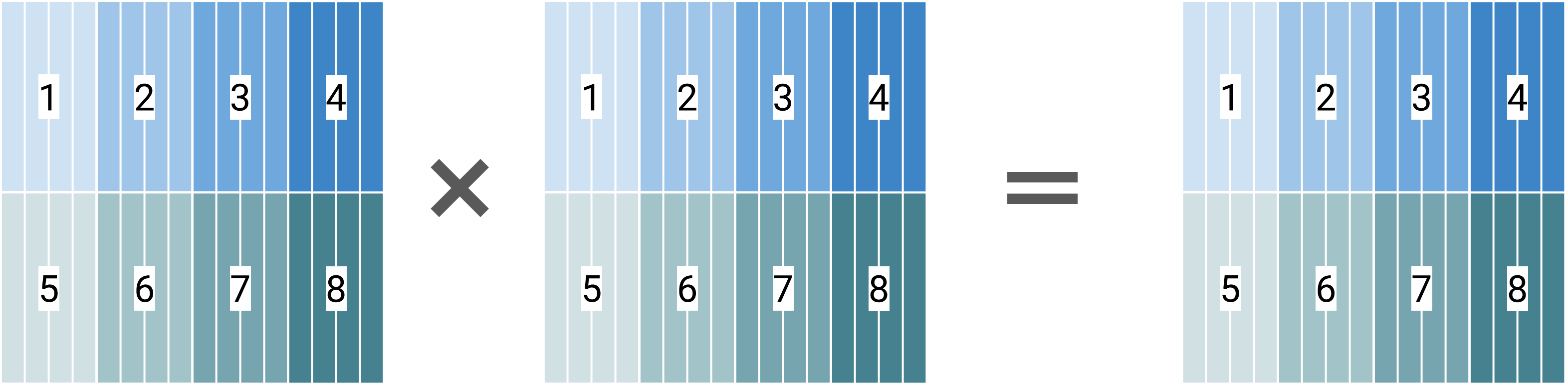
Map each array axis to hardware axis

Hardware-to-array axis mapping defines parallelism

```
# m (array) -> y (hardware)
# n (array) -> x (hardware)
@parallelize({"m": “y”, "n": “x”})
def matmul(A, B):
    C = einsum("mn,np->mp", A, B)
    return C
```

8 machines arranged in 2x4 *mesh*





A: $[16_y, 16_x]$ B: $[16_y, 16_x]$ C: $[16_y, 16_x]$

Map each array axis to hardware axis

Hardware-to-array axis mapping defines parallelism

```
# m (array) -> y (hardware)
# n (array) -> x (hardware)
@parallelize({"m": “y”, "n": “x”})
def matmul(A, B):
    C = einsum("mn,np->mp", A, B)
    return C
```

For now treat `parallelize` as a black box magic that inserts necessary all-gather operations

More details later

Now let’s generalize from matmul to a self-attention layer

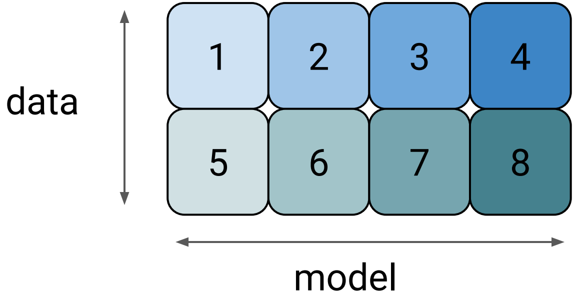
```
# b: batch
# n: sequence length
# d: embedding dimension
# h: number of heads
# k: dimension of each head
def multihead_attention(X, W_q, W_k, W_v, W_o):
    """
    X: [b, n, d] (input array)
    W_q, W_k, W_v, W_o: [h, d, k] (projection parameters)
    Y: [b, n, d] (output array)
    """
    Q = einsum("bnd,hdk->bhnk", X, W_q)
    K = einsum("bnd,hdk->bhnk", X, W_k)
    V = einsum("bnd,hdk->bhnk", X, W_v)
    scores = einsum("bhnk,bhmk->bhnm", Q, K)
    weights = softmax(scores)
    O = einsum("bhnm,bhmk->bhnk", weights, V)
    X = einsum("bhnk,hdk->bnd", O, W_o)
    return Y
```

Adapted from https://arxiv.org/abs/1911.02150

Now let's generalize from matmul to a self-attention layer

```
# b: batch
# n: sequence length
# d: embedding dimension
# h: number of heads
# k: dimension of each head
def multihead_attention(X, W_q, W_k, W_v, W_o):
    """
    X: [b, n, d] (input array)
    W_q, W_k, W_v, W_o: [h, d, k] (projection parameters)
    Y: [b, n, d] (output array)
    """
    Q = einsum("bnd,hdk->bhnk", X, W_q)
    K = einsum("bnd,hdk->bhnk", X, W_k)
    V = einsum("bnd,hdk->bhnk", X, W_v)
    scores = einsum("bhnk,bhmk->bhnm", Q, K)
    weights = softmax(scores)
    O = einsum("bhnm,bhmk->bhnk", weights, V)
    X = einsum("bhnk,hdk->bnd", O, W_o)
    return Y
```

8 machines arranged in 2x4 mesh



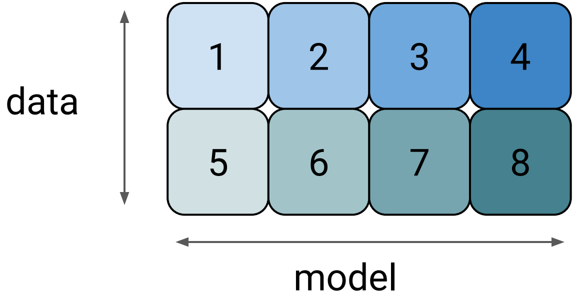
In the past (e.g. in Mesh TensorFlow) "data" and "model" represented "data parallelism" and "model parallelism"

Now this is generalized and mostly by convention

Now let's generalize from matmul to a self-attention layer

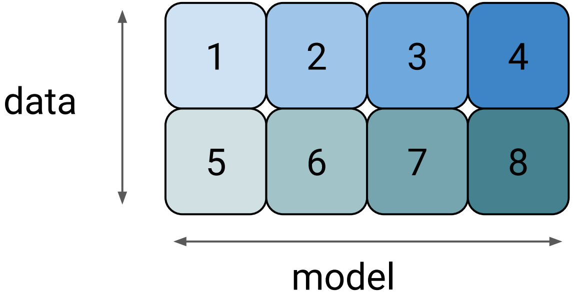
```
@parallelize({
    "b": "data",
    "n": None,
    "d": None,
    "h": "model",
    "k": None
})
def multihead_attention(X, W_q, W_k, W_v, W_o):
    """
    X: [b, n, d] (input array)
    W_q, W_k, W_v, W_o: [h, d, k] (projection parameters)
    Y: [b, n, d] (output array)
    """
    Q = einsum("bnd,hdk->bhnk", X, W_q)
    K = einsum("bnd,hdk->bhnk", X, W_k)
    V = einsum("bnd,hdk->bhnk", X, W_v)
    scores = einsum("bhnk,bhmk->bhnm", Q, K)
    weights = softmax(scores)
    O = einsum("bhnm,bhmk->bhnk", weights, V)
    X = einsum("bhnk,hdk->bnd", O, W_o)
    return Y
```

8 machines arranged in 2x4 mesh

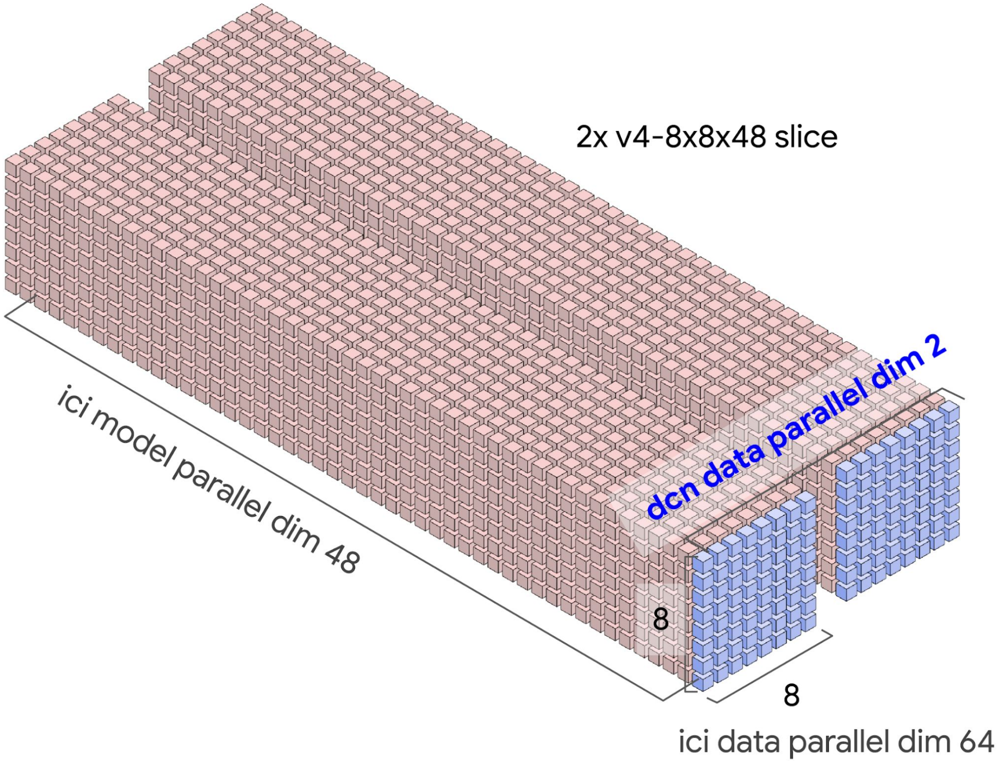


Toy example to full-scale: same underlying principle

8 machines arranged in 2x4 mesh



6144 TPU chips



https://cloud.google.com/blog/products/compute/using-cloud-tpu-multislice-to-scale-ai-workloads

So far we have assumed that `parallelize` decorator just works

One approach: GSPMD

- Write neural net as if you have a machine with infinite memory (no need to parallelize)
- Represent the core part (e.g. `train_step`) as a computational graph
- Map the input and output of that graph to hardware axes
- Give the graph to XLA. It inserts necessary communication operations and returns the parallelized version

Other approaches (e.g. manual annotation) exist but at the end, all these approaches involve mapping the array axes are mapped to hardwares

Concrete example: JAX's `pjit`, a front-end to the XLA GSPMD backend

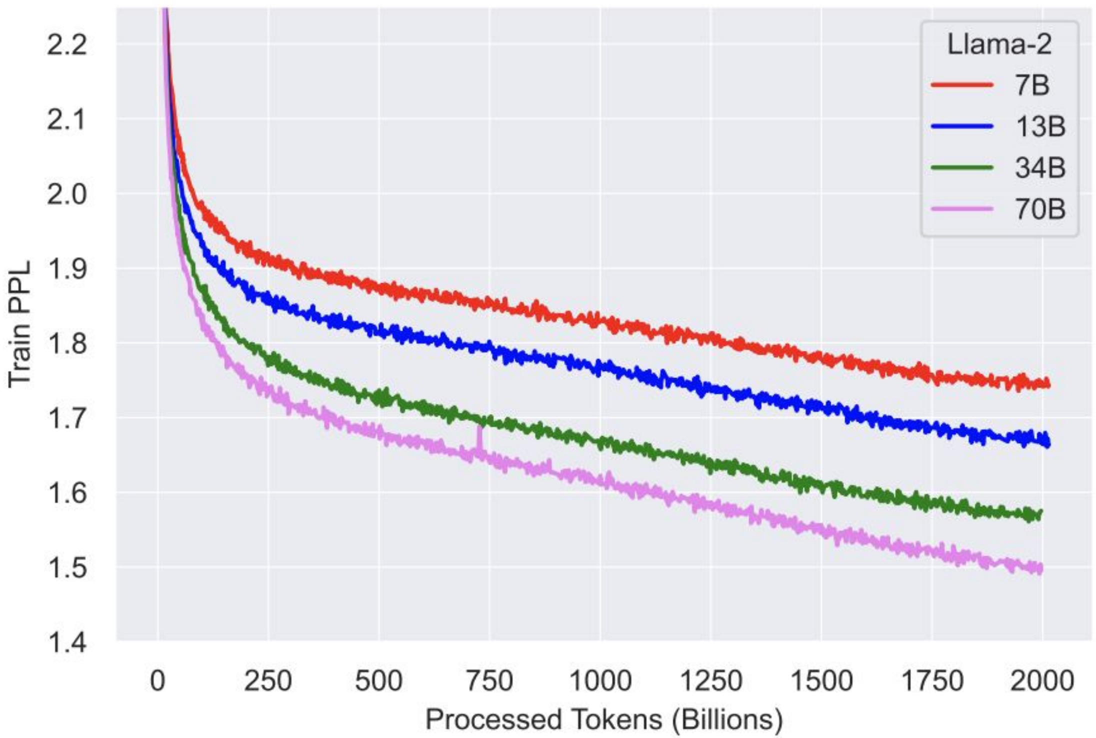
Define a function [`train_step`] that runs both forward and backward passes

"Partition" by [wrapping with `jax.pjit`] to get `partitioned_train_step`

These code paths in T5X[1] were used to train PaLM (540B dense language model)

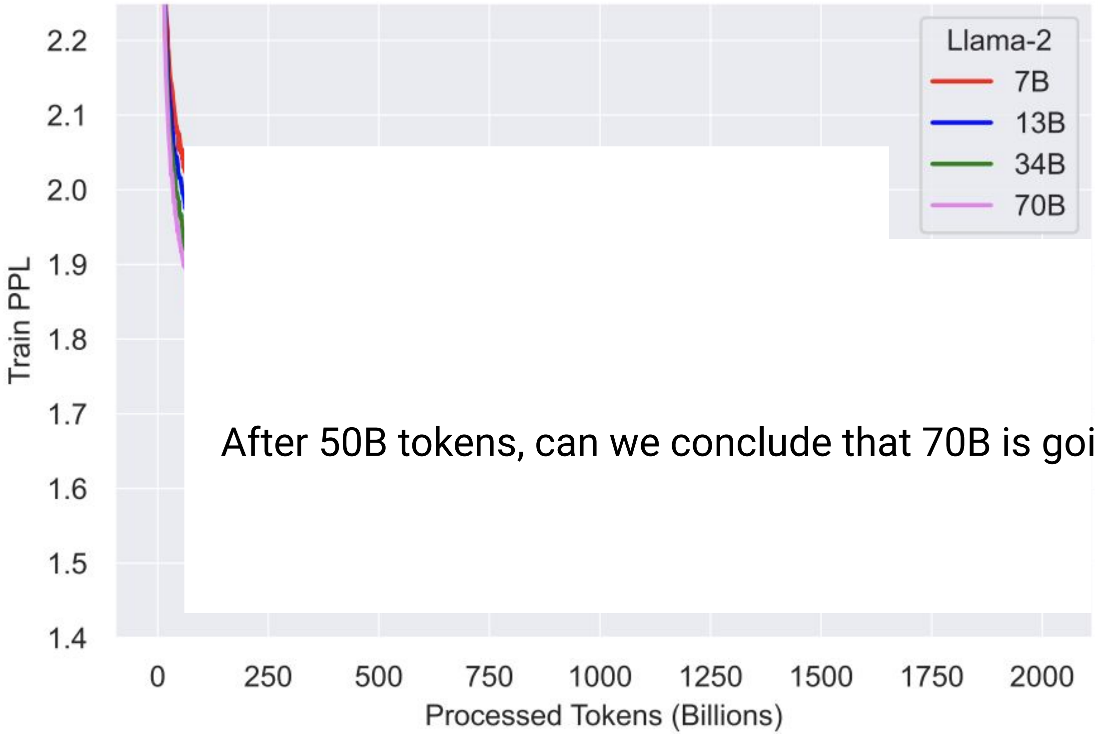
1. *Scaling Up Models and Data with t5x and seqio*
Adam Roberts, Hyung Won Chung*, Anselm Levskaya*, Gaurav Mishra*, James Bradbury*, et al. (2022)*

Iteration on pre-training is very expensive



[Llama 2: Open Foundation and Fine-Tuned Chat Models](), *Hugo Touvron, Louis Martin, Kevin Stone et al. (2023)*

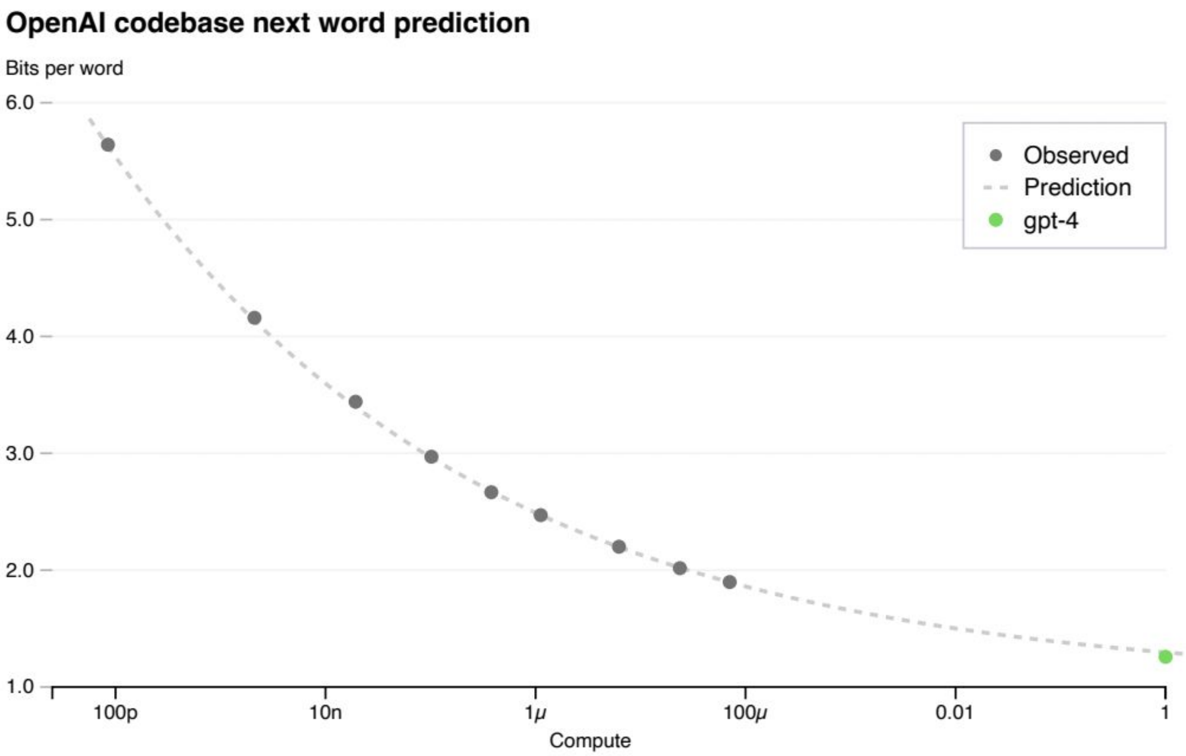
Iteration on pre-training is very expensive



After 50B tokens, can we conclude that 70B is going smoothly? 🤔

[Llama 2: Open Foundation and Fine-Tuned Chat Models](), *Hugo Touvron, Louis Martin, Kevin Stone et al. (2023)*

Scaling laws



[GPT-4 Technical Report](), *OpenAI (2023)*

Scaling to the largest scale ever is very, very hard

Scaling is not going from

```
python train.py --model_size=small
```

to

```
python train.py --model_size=very_large
```

Example: loss spikes

During PaLM training, there were about 20 loss spikes that unnerved many people

We trained 3 models (8B, 62B, 540B) on exact same data. Only happened at 540B

This is not caused by bad data

Every hour not making the decision to handle this is 6144 chips sitting idle

It is becoming easier to train a given size, BUT

Scale is increasing at a faster rate than the rate at which things become easier

At the frontier, it is always challenging for many reasons

Scaling doesn’t solve all problems

We also need post-training

We can’t talk to the pretrained model directly

Model input

Make up a word that means "when two AI researchers go on a date".

PaLM 540B output

Make up a word that means "when two AI researchers go on a date".

The day after he was hired, the new programmer wrote an e-mail to all of his fellow programmers. It said, "I will be on vacation next week."

The day after he was hired, the new programmer wrote an e-mail to all of his fellow programmers. It said, "I will be on vacation next week."

The day after [...]

✖ (repeats input and keep repeating generations)

Flan-PaLM 540B output

date-mining ✔

Scaling Instruction-Finetuned Language Models
Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus et al. (2022)

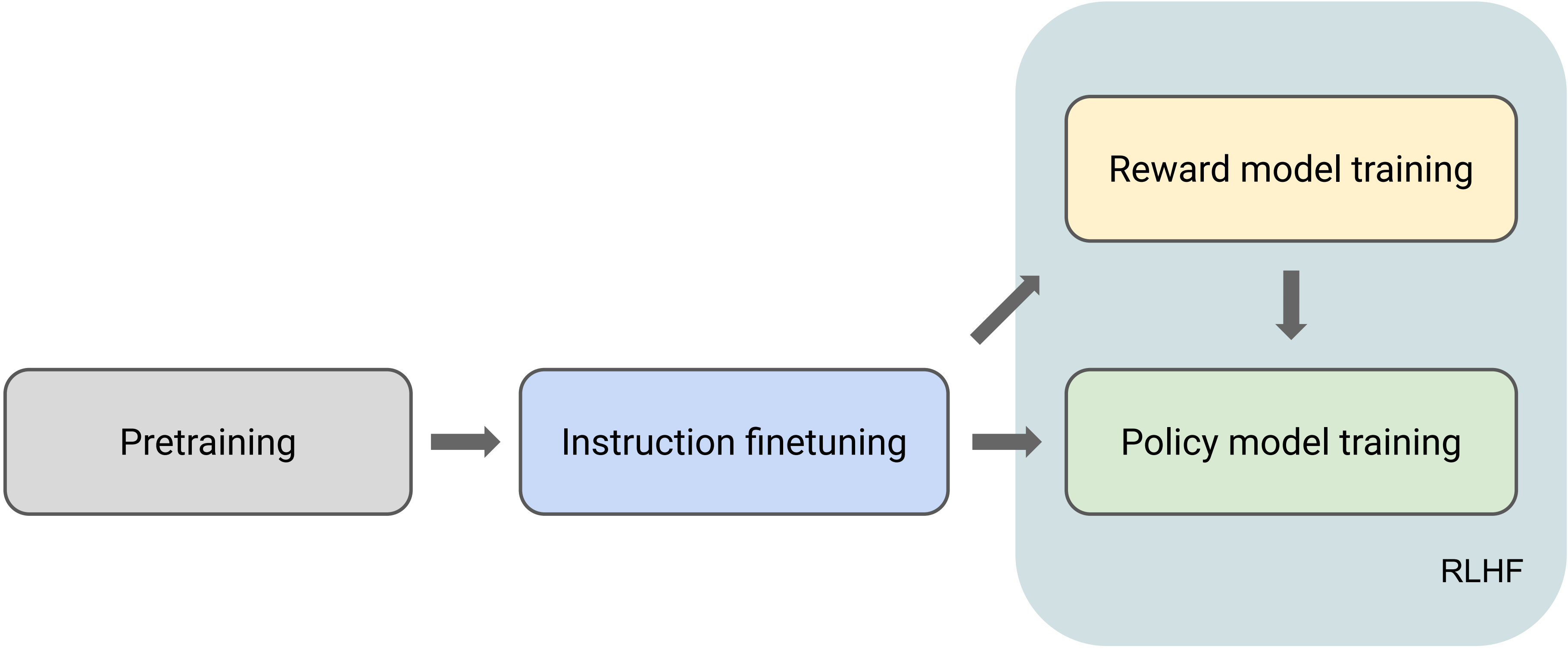
Hack: we can frame the question so that the answer is the next token

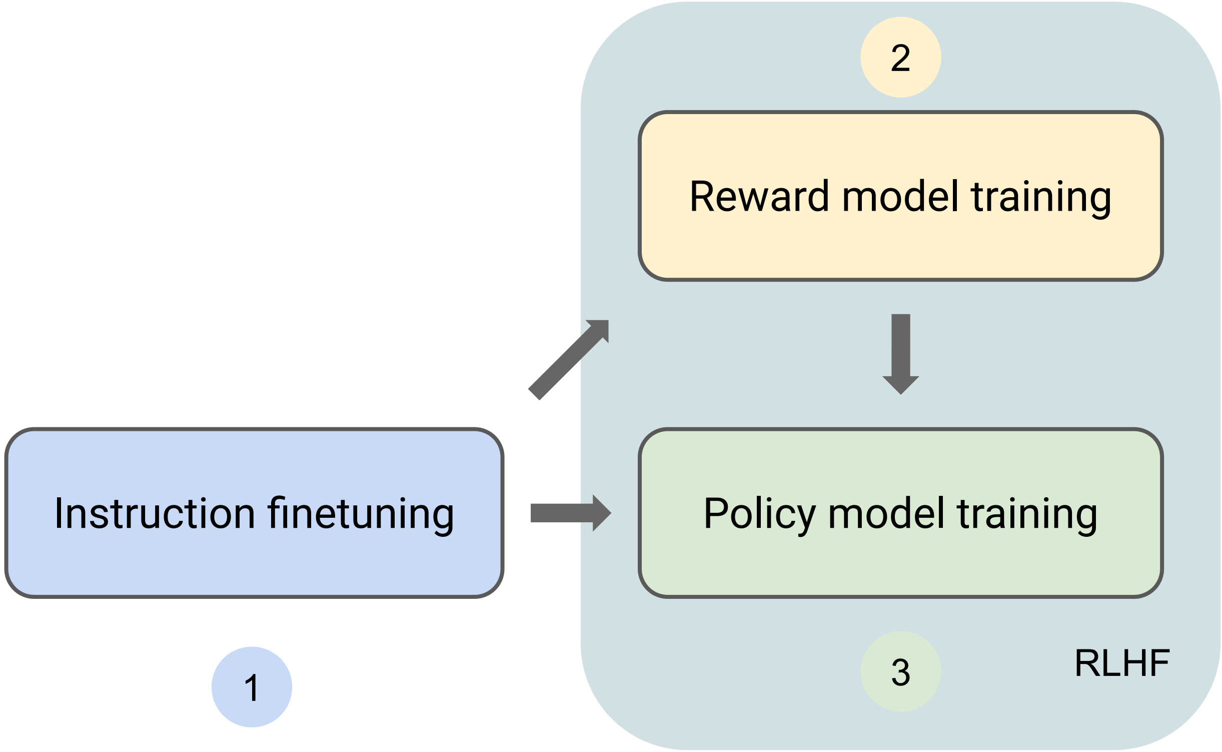
Q: The square root of x is the cube root of y. What is y to the power of 2, if x = 4?

A:

Pretrained model just predicts the next token, which happens to be the answer

Pre-trained models always generate something that is a natural continuation of the prompts *even if the prompts are malicious*

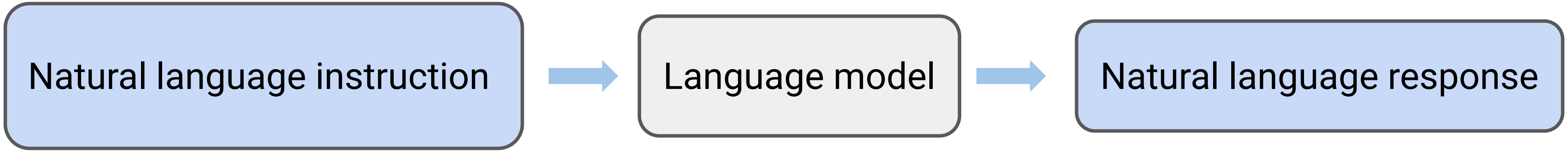

Pretraining
Instruction finetuning
Reward model training
Policy model training
RLHF


2
Reward model training
Instruction finetuning
Policy model training
1
3
RLHF

Instruction finetuning

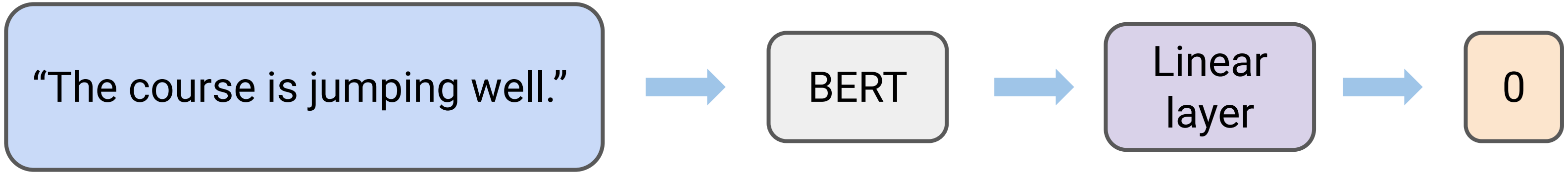
*Frame **all** tasks in the form of*

natural language instruction to natural language response mapping



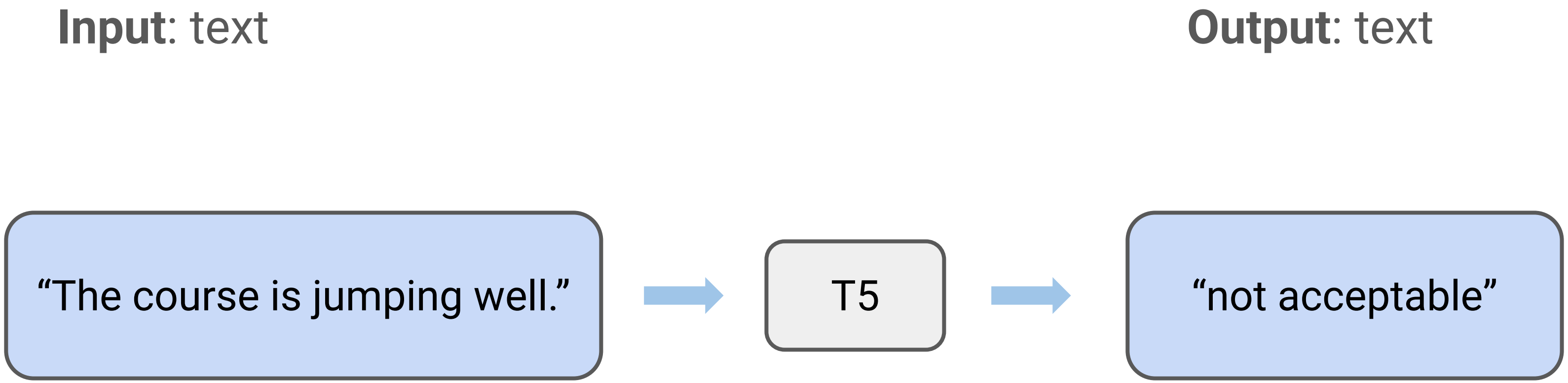
Input: text

Output: label



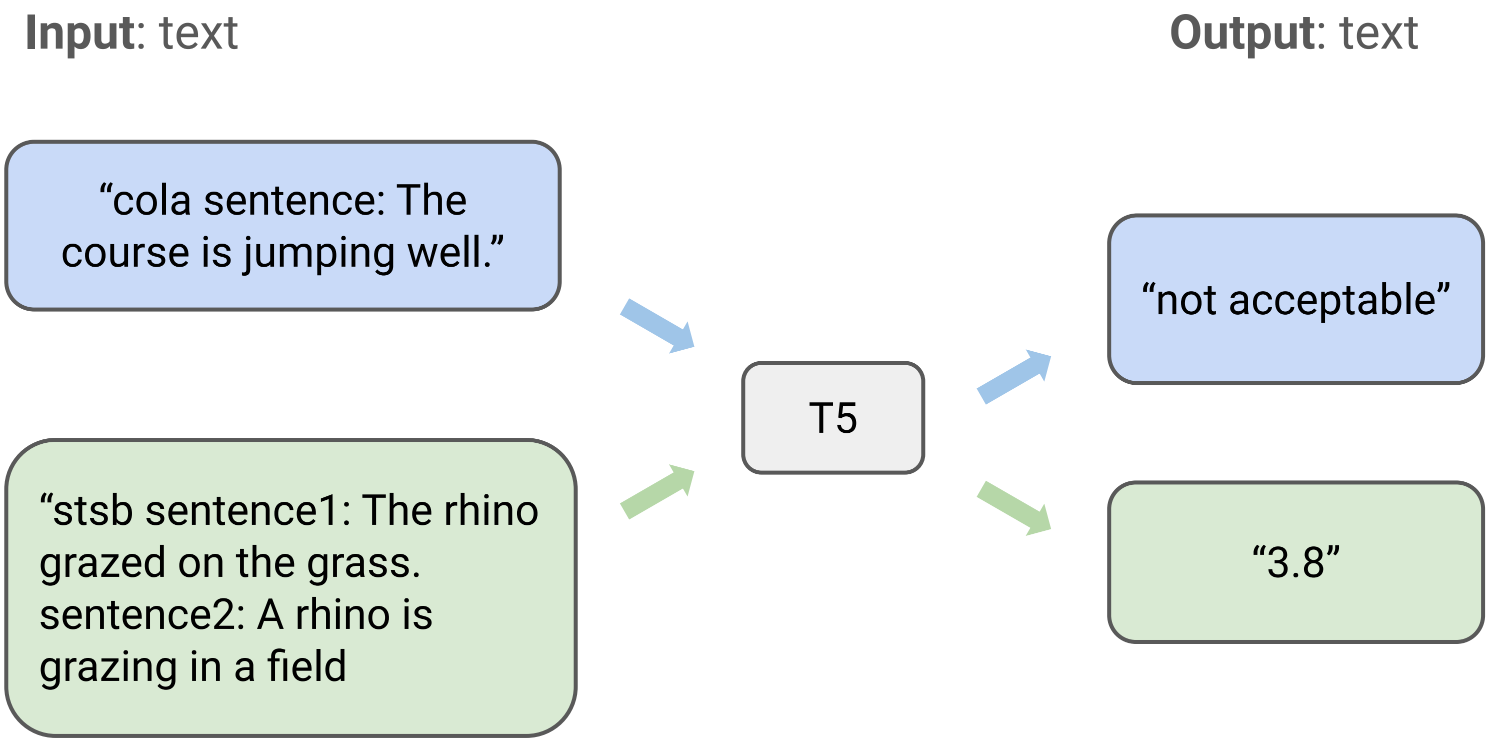
<u>Task specific linear layer is necessary</u>

Devlin et al. (2018)

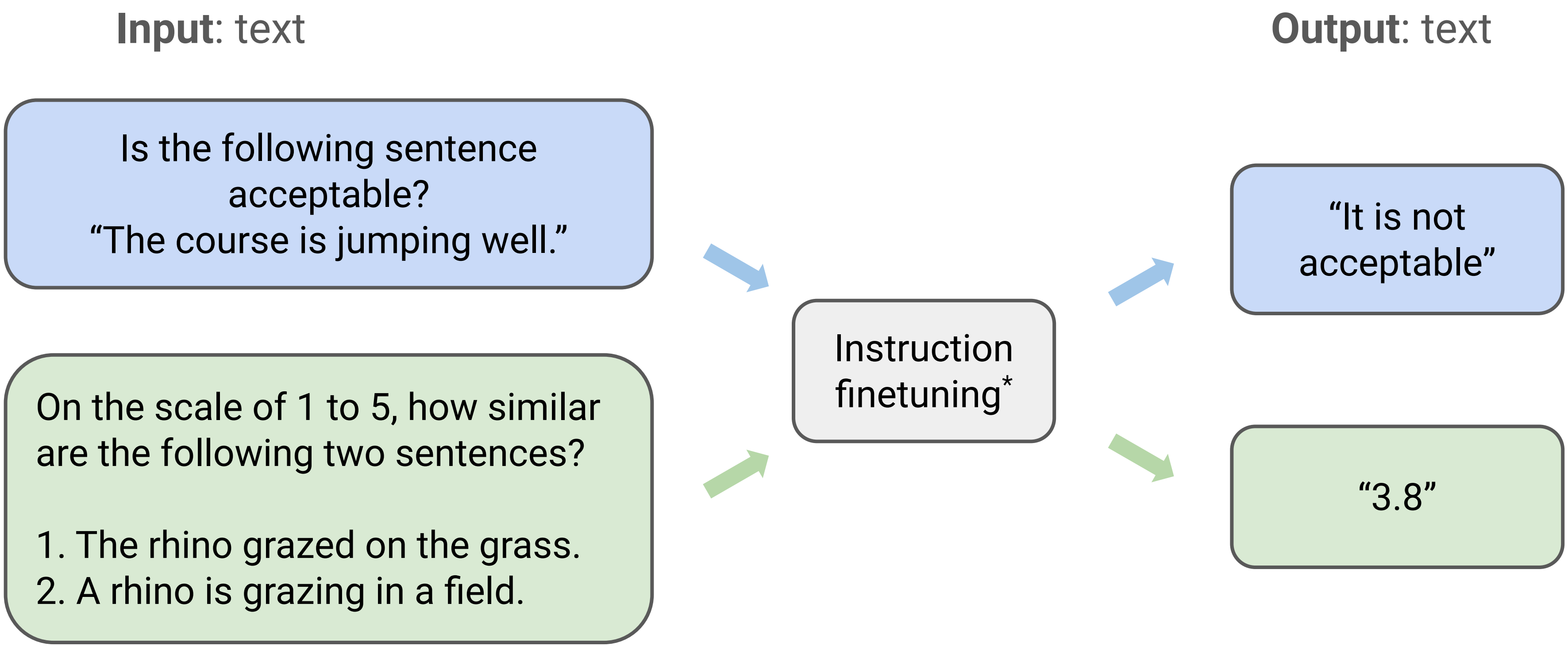


<u>Architecture is unified across tasks with *text-to-text* format</u>

Raffel et al. (2019)



<u>Tasks are not semantically related</u>



<u>Tasks are unified. So for an unseen task, the model just needs to respond to the natural language instruction</u>

*[Wei et al. (2021)](), [Sanh et al. (2021)](), [Ouyang et al. (2022)]()

Finetuning tasks

T0-SF

Commonsense reasoning
Question generation
Closed-book QA
Adversarial QA
Extractive QA
Title/context generation
Topic classification
Struct-to-text
...

55 Datasets, 14 Categories, 193 Tasks

Muffin

Natural language inference
Code instruction gen.
Program synthesis
Dialog context generation
Closed-book QA
Conversational QA
Code repair
...

69 Datasets, 27 Categories, 80 Tasks

CoT (Reasoning)

Arithmetic reasoning
Commonsense Reasoning
Implicit reasoning
Explanation generation
Sentence composition
...

9 Datasets, 1 Category, 9 Tasks

Natural Instructions v2

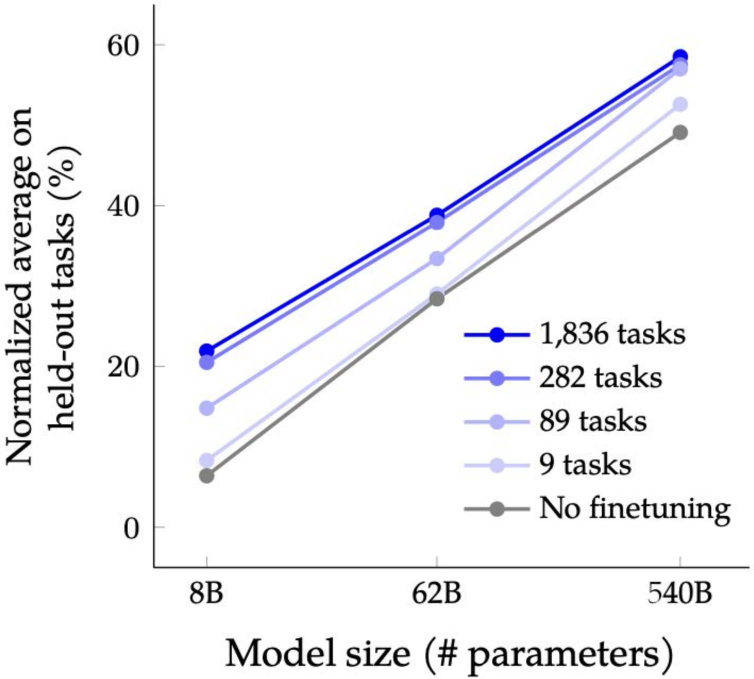
Cause effect classification
Commonsense reasoning
Named entity recognition
Toxic language detection
Question answering
Question generation
Program execution
Text categorization
...

372 Datasets, 108 Categories, 1554 Tasks

- A **Dataset** is an original data source (e.g. SQuAD).
- A **Task Category** is unique task setup (e.g. the SQuAD dataset is configurable for multiple task categories such as extractive question answering, query generation, and context generation).
- A **Task** is a unique <dataset, task category> pair, with any number of templates which preserve the task category (e.g. query generation on the SQuAD dataset.)

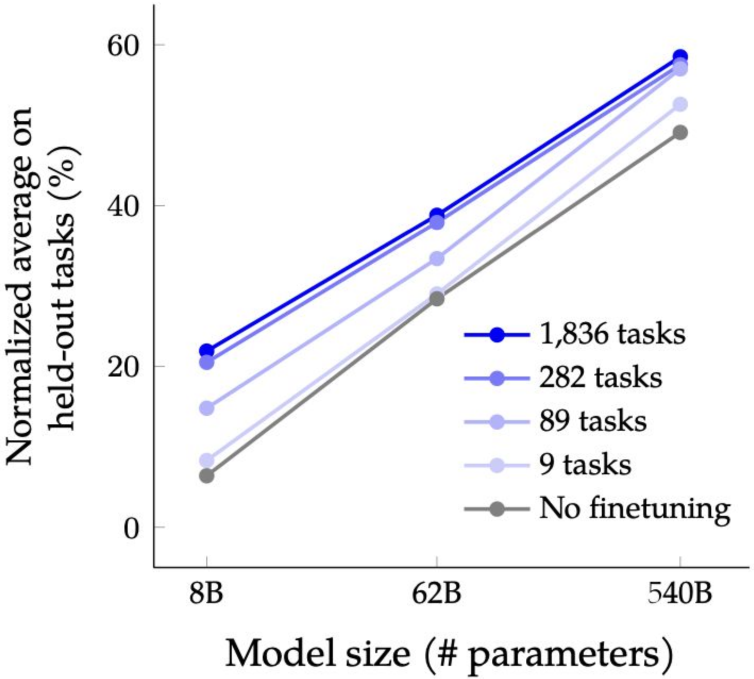
Instruction finetuning on 1836 (!!) academic tasks

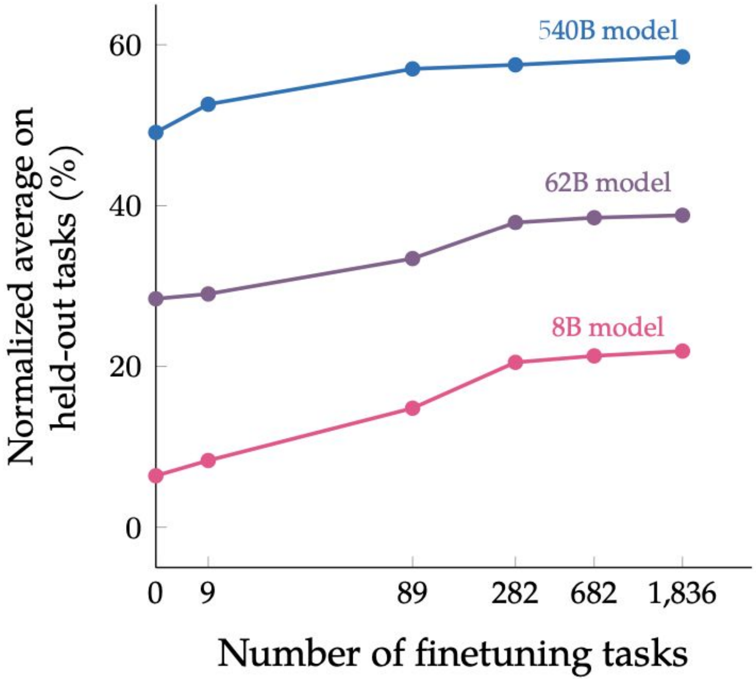
Scaling the number of tasks and model size improves the performance



[Scaling Instruction-Finetuned Language Models]()
Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus et al. (2022)

Scaling the number of tasks and model size improves the performance





[*Scaling Instruction-Finetuned Language Models*]()
Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus et al. (2022)

Instruction fine-tuning is highly effective but it has inherent limitations

What is the learning objective in instruction finetuning?

For a given input, the target is the single correct answer

In RL, this is called "behavior cloning"

What is the learning objective in instruction finetuning?

For a given input, the target is the single correct answer

In RL, this is called "behavior cloning"

Hope is that if we have enough of these, the model can learn to generalize

What is the learning objective in instruction finetuning?

For a given input, the target is the single correct answer

In RL, this is called “behavior cloning”

Hope is that if we have enough of these, the model can learn to generalize

This requires <u>formalizing the correct behavior</u> for a given input

Exercise: think about the single correct answer

Input

2 + 3?

Target

5

Exercise: think about the single correct answer

Input

Translate this to Korean:
"I should have studied instead of watching this movie"

Target

나는 이 영화 보는 대신 공부를 했어야 했다

Exercise: think about the single correct answer

Input

Write a letter to a 5-year-old boy from Santa Clause explaining that Santa is not real. Convey gently so as not to break his heart

Target

Exercise: think about the single correct answer

Input

Implement logistic regression with gradient descent in Python

Target

```
class LogisticRegression:
    …
```

Observations

Increasingly we want to teach models more abstract behaviors

Observations

Increasingly we want to teach models more abstract behaviors

Objective function of instruction finetuning seems to be the “bottleneck” of teaching these behaviors

Observations

Increasingly we want to teach models more abstract behaviors

Objective function of instruction finetuning seems to be the “bottleneck” of teaching these behaviors

The maximum likelihood objective is “predefined” function (i.e. no learnable parameter)

Observations

Increasingly we want to teach models more abstract behaviors

Objective function of instruction finetuning seems to be the “bottleneck” of teaching these behaviors

The maximum likelihood objective is “predefined” function (i.e. no learnable parameter)

Can we parameterize the objective function and ***learn*** it?

RL provides one way to use a learned objective

In RL, we try to maximize the expected reward function

$$\max_{\theta} \mathbb{E}[R]$$

RL provides one way to use a learned objective

In RL, we try to maximize the expected reward function

$$\max_{\theta} \mathbb{E}[R]$$

Reward is the objective function. We can *learn* the reward: reward model.

RL provides one way to use a learned objective

In RL, we try to maximize the expected reward function

$$\max_{\theta} \mathbb{E}[R]$$

Reward is the objective function. We can *learn* the reward: reward model.

We know how to do supervised learning with neural network well. Let’s use neural net to represent the reward model.

Reward Model (RM) training

Reward Model (RM) training data: which completion is better?

Input

Explain the moon landing to a 6 year old in a few sentences

Completion 1

The Moon is a natural satellite of the Earth. It is the fifth largest moon in the Solar System and the largest relative to the size of its host planet.

<

Completion 2

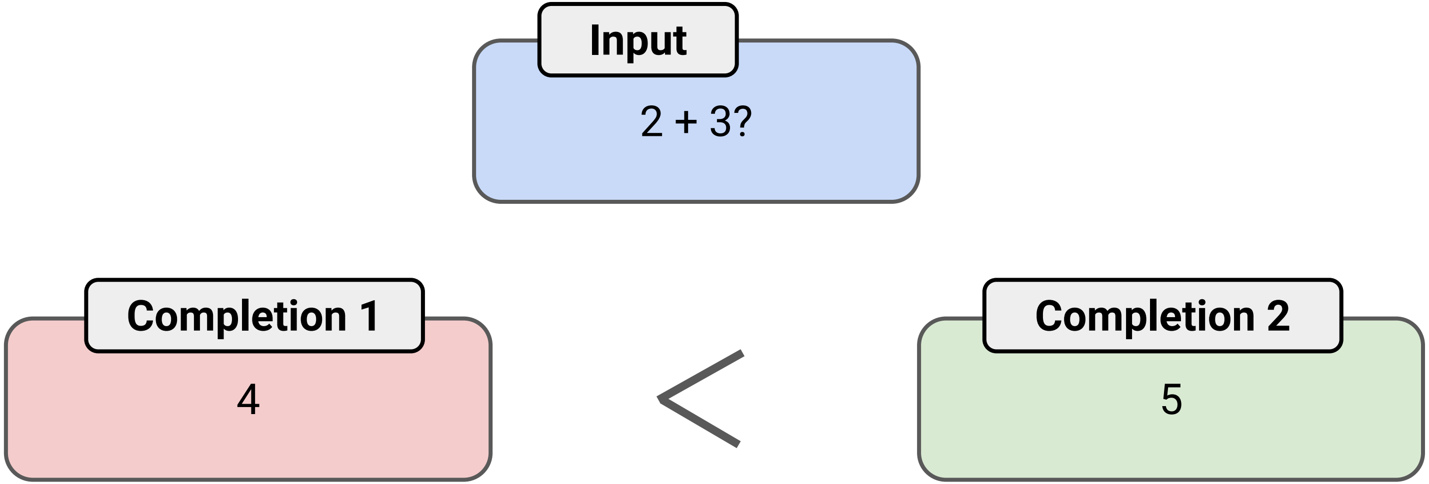
People went to the moon, and they took pictures of what they saw, and sent them back to the earth so we could all see them.

Humans label which completion is preferred.

This setup aims to align models to the human preference

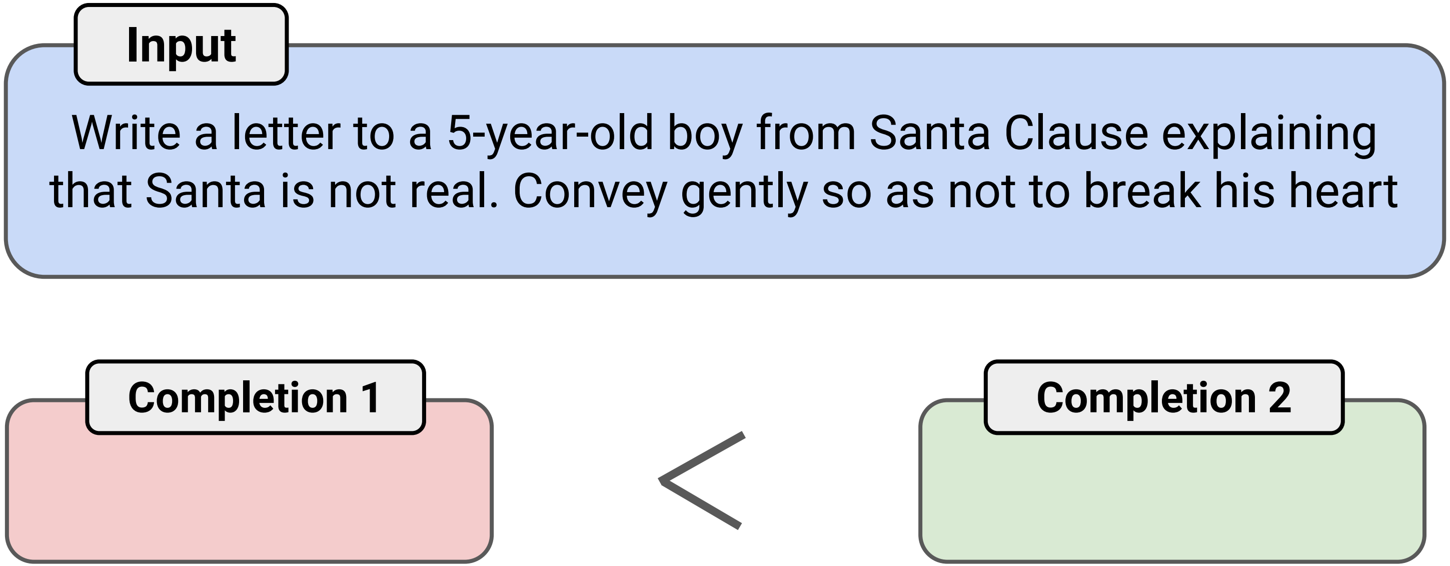
Why use comparison for RM?

For an easy prompt where a clear answer exists, comparison may not be useful



Why use comparison for RM?

But for more open-ended generations, it is easier to compare relatively



Reward Model (RM) training objective function

Let p_{ij} be the probability that completion y_i is better than completion y_j

Bradley–Terry model (1952): log odds that completion y_i is favored over y_j is modeled as difference in the rewards

$$\log \frac{p_{ij}}{1 - p_{ij}} = r(x, y_i; \phi) - r(x, y_j; \phi)$$

Reward Model (RM) training objective function

Let p_{ij} be the probability that completion y_i is better than completion y_j

Bradley–Terry model (1952): log odds that completion y_i is favored over y_j is modeled as difference in the rewards

$$\log \frac{p_{ij}}{1 - p_{ij}} = r(x, y_i; \phi) - r(x, y_j; \phi)$$

$$p_{ij} = \frac{e^{r(x,y_i;\phi) - r(x,y_j;\phi)}}{1 + e^{r(x,y_i;\phi) - r(x,y_j;\phi)}} = \sigma(r(x, y_i; \phi) - r(x, y_j; \phi))$$

Reward Model (RM) training objective function

Let p_{ij} be the probability that completion y_i is better than completion y_j

Bradley–Terry model (1952): log odds that completion y_i is favored over y_j is modeled as difference in the rewards

$$\log \frac{p_{ij}}{1 - p_{ij}} = r(x, y_i; \phi) - r(x, y_j; \phi)$$

$$p_{ij} = \frac{e^{r(x,y_i;\phi) - r(x,y_j;\phi)}}{1 + e^{r(x,y_i;\phi) - r(x,y_j;\phi)}} = \sigma(r(x, y_i; \phi) - r(x, y_j; \phi))$$

$$\max_{\phi} \sum_{x, y_i, y_j \in D} \log p_{ij}$$

Policy model training

Policy model objective function

Once we have a reward model, we can use it in RL to learn the language model parameters that maximizes the expected reward

$$J(\theta) = \mathbb{E}_{(X,Y)\sim D_{\pi_\theta}}[r(X,Y;\phi)]$$

where $X = (X_1, \ldots, X_S)$ is the prompt and $Y = (Y_1, \ldots, Y_T)$ is the completion sampled from the policy model.

Policy model training

The optimization problem is then

$$\max_{\theta} J(\theta) = \max_{\theta} \mathbb{E}_{(X,Y)\sim D_{\pi_\theta}}[r(X, Y; \phi)]$$

Policy model training

The optimization problem is then

$$\max_{\theta} J(\theta) = \max_{\theta} \mathbb{E}_{(X,Y)\sim D_{\pi_\theta}}[r(X,Y;\phi)]$$

We use iterative algorithm such as gradient ascent to solve this

$$\theta := \theta + \alpha \nabla J(\theta)$$

Policy model training

The optimization problem is then

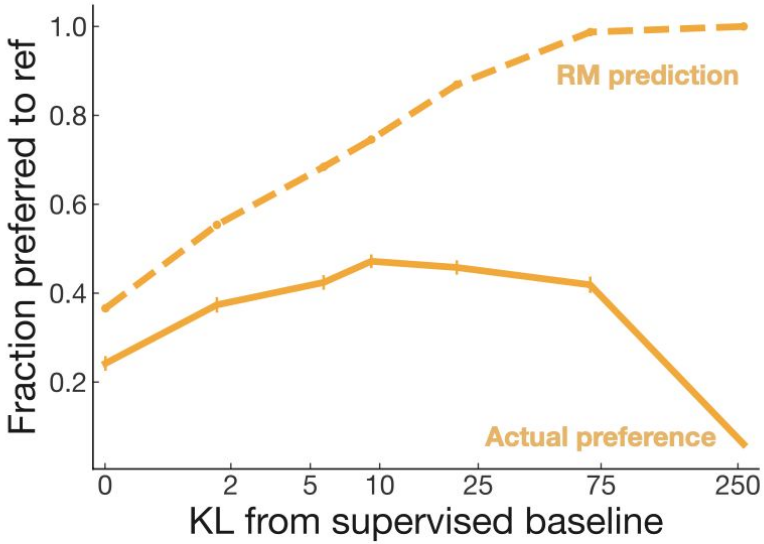
$$\max_{\theta} J(\theta) = \max_{\theta} \mathbb{E}_{(X,Y)\sim D_{\pi_\theta}}[r(X, Y; \phi)]$$

We use iterative algorithm such as gradient ascent to solve this

$$\theta := \theta + \alpha \nabla J(\theta)$$

We can use policy gradient algorithms such as PPO to compute the gradient.

RLHF is tricky to get right



Reward model is susceptible to “reward hacking”.

When policy is over-optimized, actual human preference can be negatively correlated with RM prediction

Steinnon et al. (2020)

Why should we keep studying RLHF?

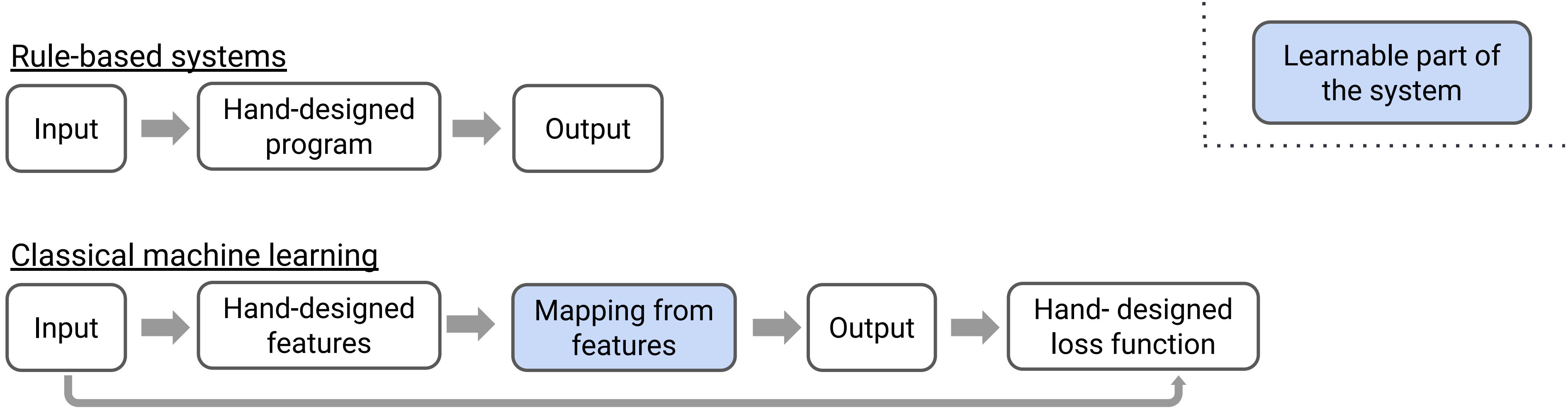
Maximum likelihood is too strong of an inductive bias

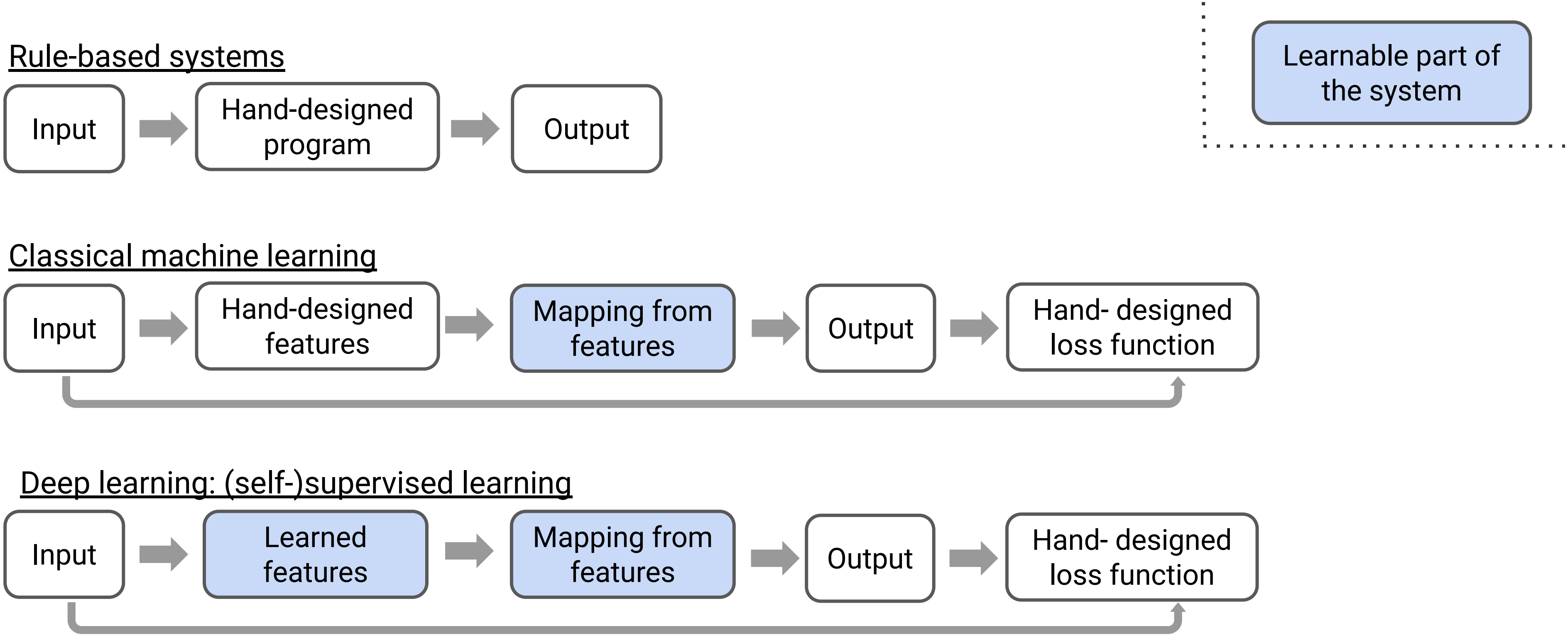
Learning the objective function is a different paradigm and there is a lot of room for improvement

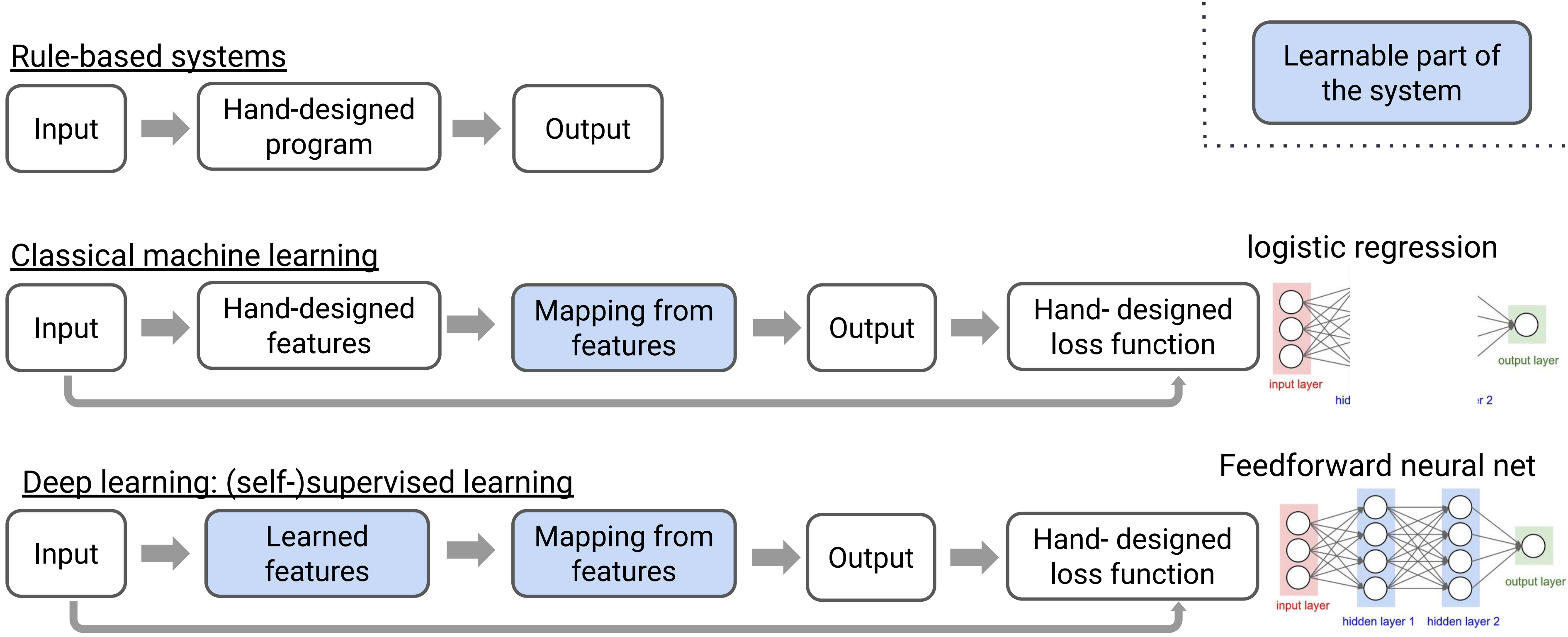
If something is so principled, we should keep at it until it works

Learnable part of the system

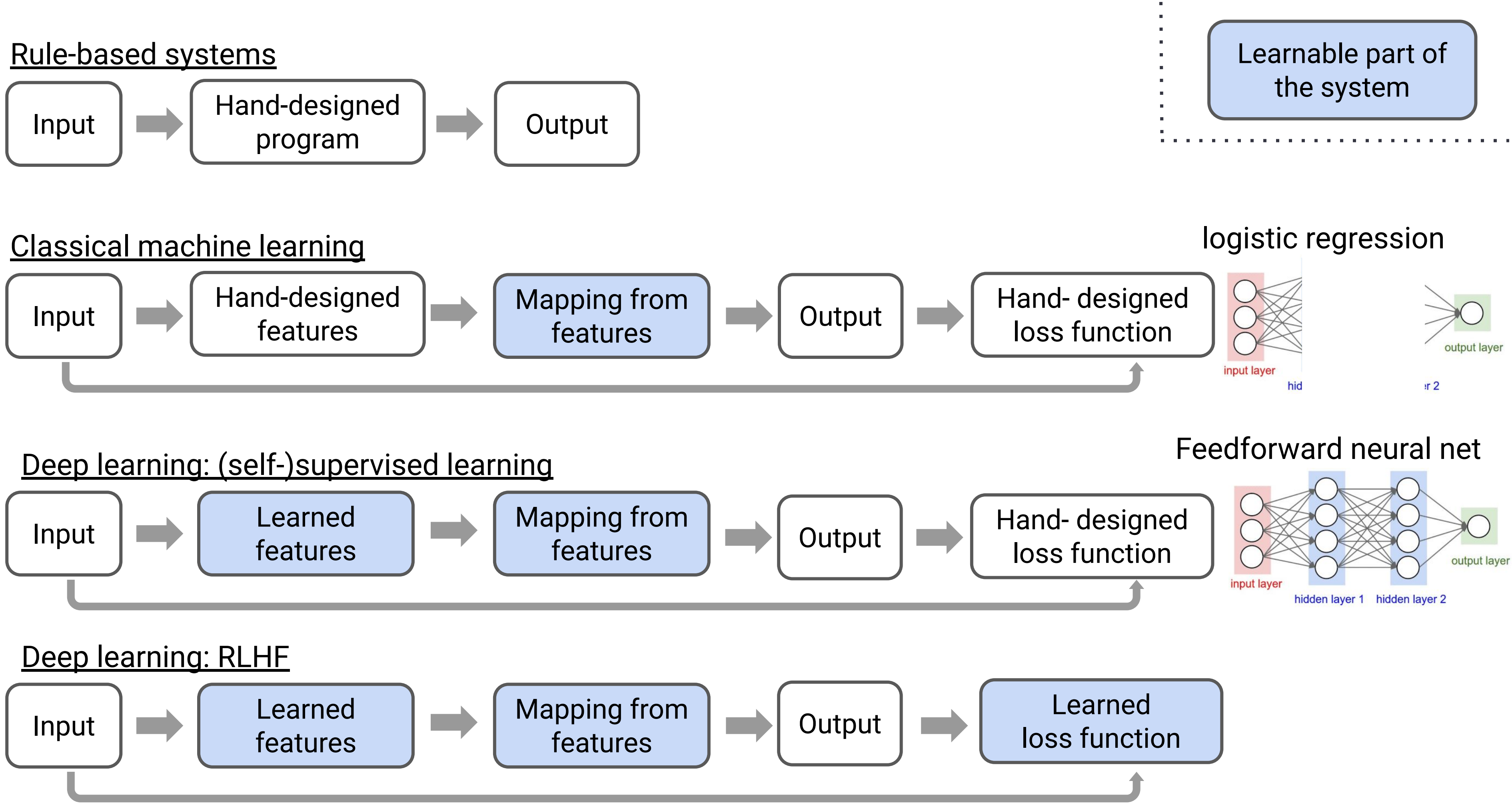
Rule-based systems

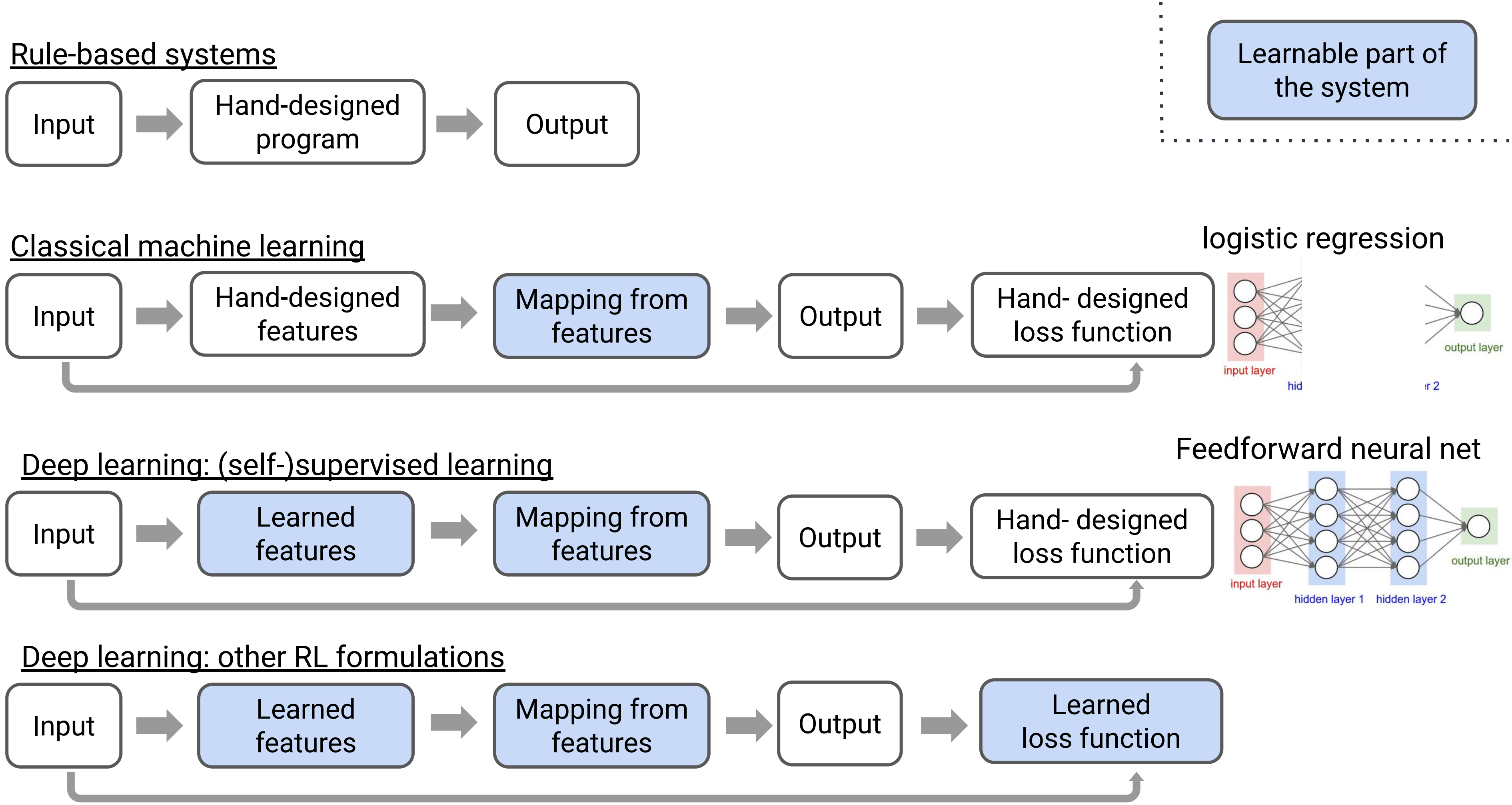

Rule-based systems
Input
Hand-designed program
Output
Classical machine learning
Input
Hand-designed features
Mapping from features
Output
Hand- designed loss function
Learnable part of the system


Rule-based systems
Input
Hand-designed program
Output
Learnable part of the system
Classical machine learning
Input
Hand-designed features
Mapping from features
Output
Hand- designed loss function
Deep learning: (self-)supervised learning
Input
Learned features
Mapping from features
Output
Hand- designed loss function

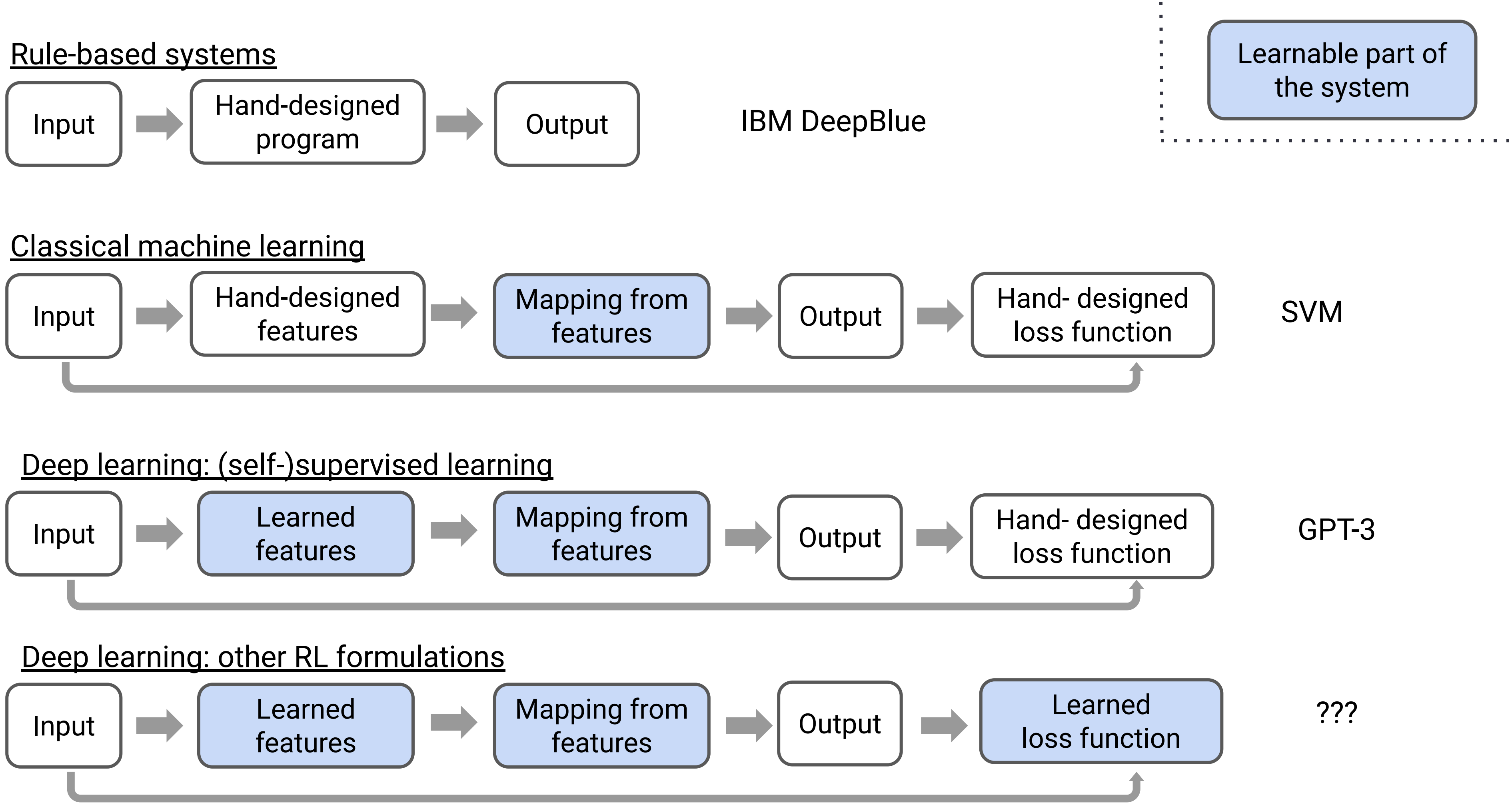


Rule-based systems
Input
Hand-designed program
Output
Learnable part of the system
Classical machine learning
Input
Hand-designed features
Mapping from features
Output
Hand- designed loss function
logistic regression
input layer
hid
output layer
Deep learning: (self-)supervised learning
Input
Learned features
Mapping from features
Output
Hand- designed loss function
Feedforward neural net
input layer
hidden layer 1
hidden layer 2
output layer


Learnable part of the system
Rule-based systems
Input
Hand-designed program
Output
Classical machine learning
Input
Hand-designed features
Mapping from features
Output
Hand- designed loss function
logistic regression
input layer
output layer
Deep learning: (self-)supervised learning
Input
Learned features
Mapping from features
Output
Hand- designed loss function
Feedforward neural net
input layer
hidden layer 1
hidden layer 2
output layer
Deep learning: RLHF
Input
Learned features
Mapping from features
Output
Learned loss function



Learnable part of the system
Rule-based systems
Input
Hand-designed program
Output
Classical machine learning
Input
Hand-designed features
Mapping from features
Output
Hand- designed loss function
logistic regression
input layer
output layer
hid
Deep learning: (self-)supervised learning
Input
Learned features
Mapping from features
Output
Hand- designed loss function
Feedforward neural net
input layer
hidden layer 1
hidden layer 2
output layer
Deep learning: other RL formulations
Input
Learned features
Mapping from features
Output
Learned loss function


Learnable part of the system
Rule-based systems
Input
Hand-designed program
Output
IBM DeepBlue
Classical machine learning
Input
Hand-designed features
Mapping from features
Output
Hand- designed loss function
SVM
Deep learning: (self-)supervised learning
Input
Learned features
Mapping from features
Output
Hand- designed loss function
GPT-3
Deep learning: other RL formulations
Input
Learned features
Mapping from features
Output
Learned loss function
???

Bitter lesson: don't get in the way of scaling

The biggest progress in the past 10 years (or even more) can be summarized as

- Create weaker inductive biases and scale up
- Do not teach machines how we think we think. Let it learn in a machine's way

It is humbling to accept these

http://www.incompleteideas.net/IncIdeas/BitterLesson.html

Internals of Transformers do not matter as much

Many Transformer variants have been proposed but almost all fancy variations don't scale well

More useful to abstract away Transformer as sequence of functions and think about input and output shapes and types

Do Transformer Modifications Transfer Across Implementations and Applications?
Sharan Narang, Hyung Won Chung, Yi Tay, William Fedus, et al. (2021)

Large Language Models (in 2023)

Hyung Won Chung

OpenAI

Twitter: @hwchung27