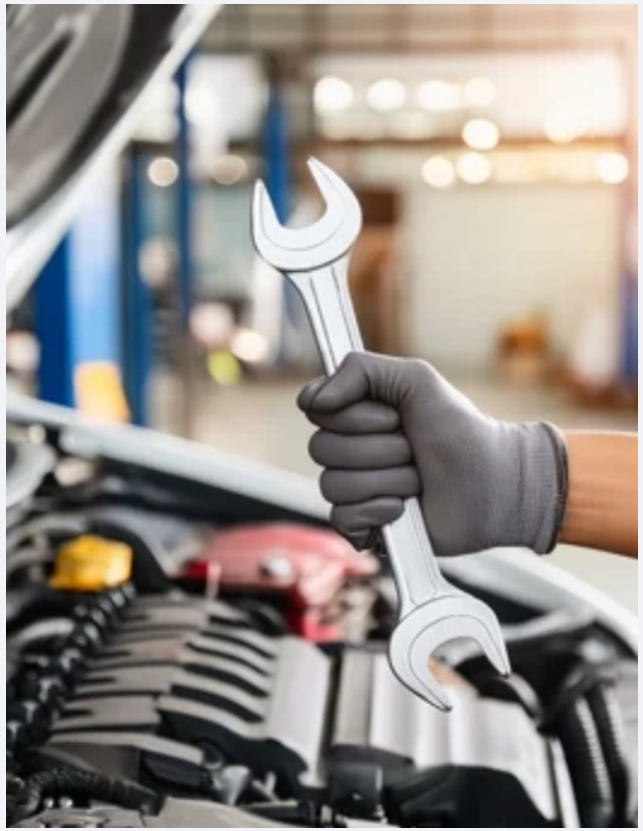
LLM Mechanics

- Dr Maria Prokofieva

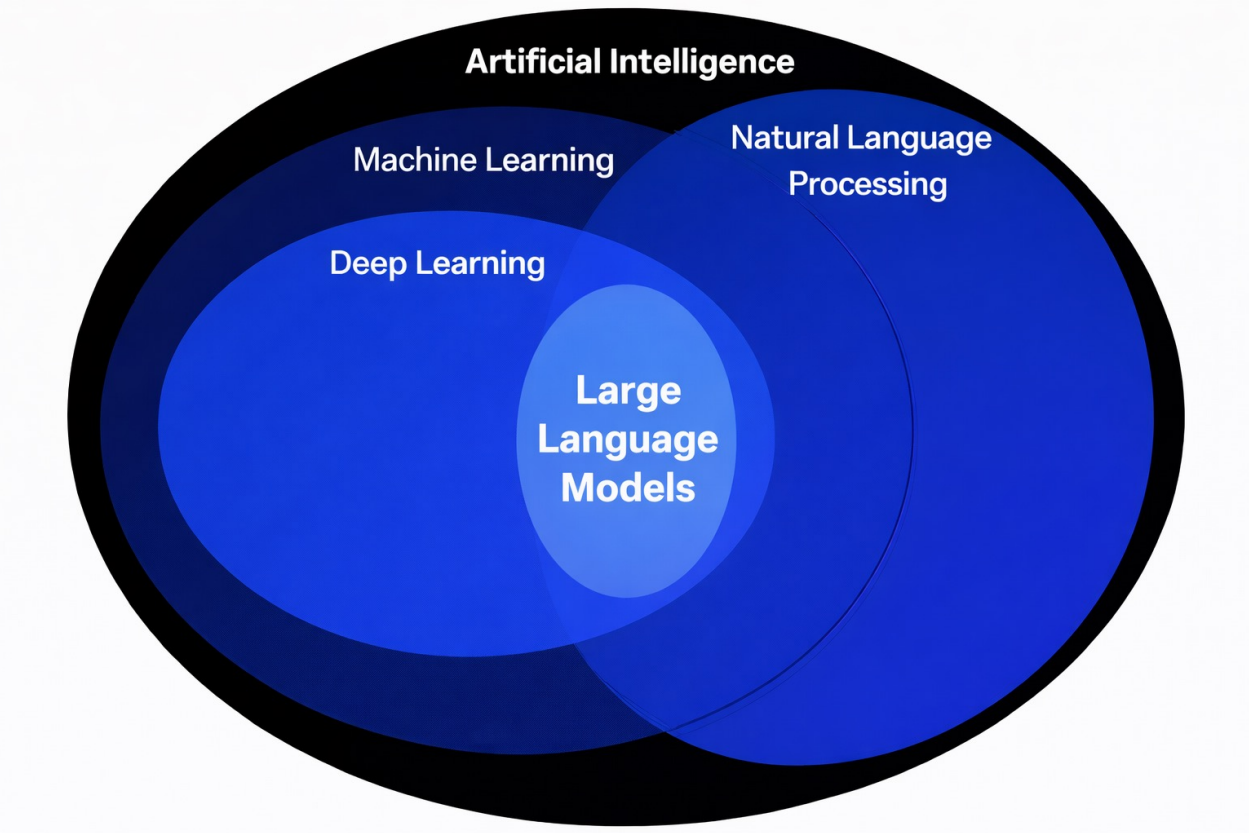
What We Will Cover

- DL basics
- How LLMs are different
- Modeling stages
- Tokenization
- Intro to embeddings
- Transformers architecture



Deep Learning → Large Language Models

- Deep learning (DL): learning functions from data using neural networks
- Large Language Models (LLMs): deep learning models specialized for token sequences
- Same training machinery, different data + architecture



Deep Learning Basics

- Data → model → training → testing
- Labeled vs unlabeled data
- Train / validation / test splits
- Generalization as the goal
- Loss ≠ evaluation metric

Modeling

- Modeling is at the heart of Machine Learning/DL
- The **aim of modeling** is to make predictions or decisions from data:
  - i.e. approximate the relationship between inputs and outputs
- A **model** is a mathematical function:
  - **x** = input (image, text, audio, tabular data).
  - **y** = target (label, category, numerical value).

$$f_\theta(\text{tokens}_{1:t}) \rightarrow P(\text{token}_{t+1})$$

- **Models generalize:**
  - given new, unseen data, the model should make accurate predictions without human intervention

Data → Model

Data (What LLMs Learn From)

Unlabeled text

Semi-structured text

Post-training data

Data quality, diversity, and coverage dominate model capability

Biases, duplication, and contamination propagate directly into behavior

No labels → evaluation difficulty

Model (What Is Being Trained)

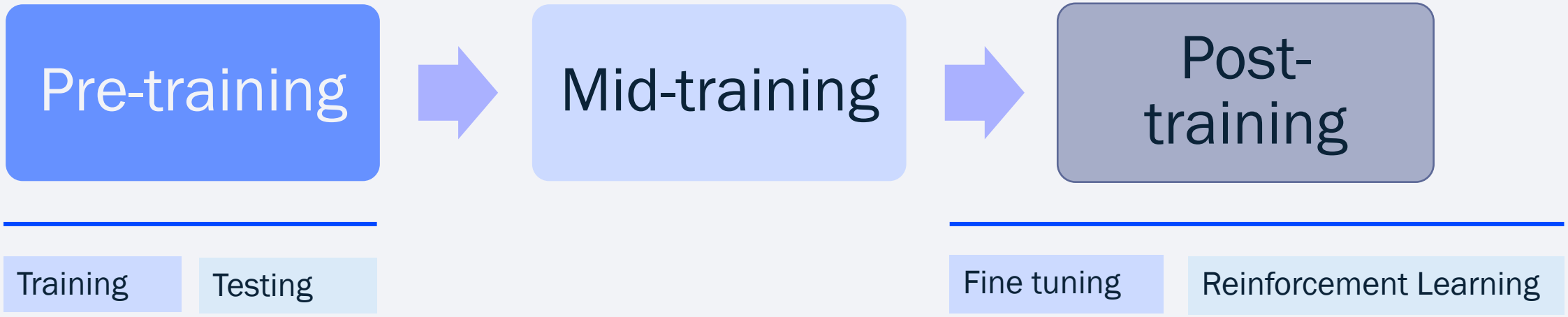
$$f_\theta(\text{tokens}_{1:t}) \rightarrow P(\text{token}_{t+1})$$

LLM specific model architecture:

- Token embedding layer
- Stacked transformer blocks (attention + feed-forward)
- Output projection to vocabulary logits

- The model does **not store facts explicitly**
- It stores **statistical regularities** in its parameters
- Architecture is mostly fixed once chosen; learning happens via parameters

LLM model workflow



What happens inside the DL model training?

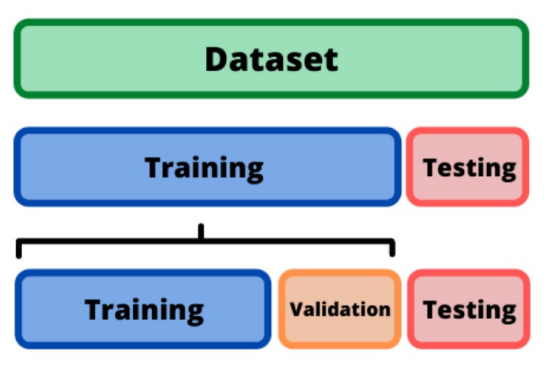
Problem:
Task
Expected outcome
DL?

Data preparation:
Collect
Preprocess

Model development:
Model selection
Design
Implementation

Model training and tuning

Deployment and maintenance



Model development:
Train = learning patterns.
Test = unseen → checks generalization.

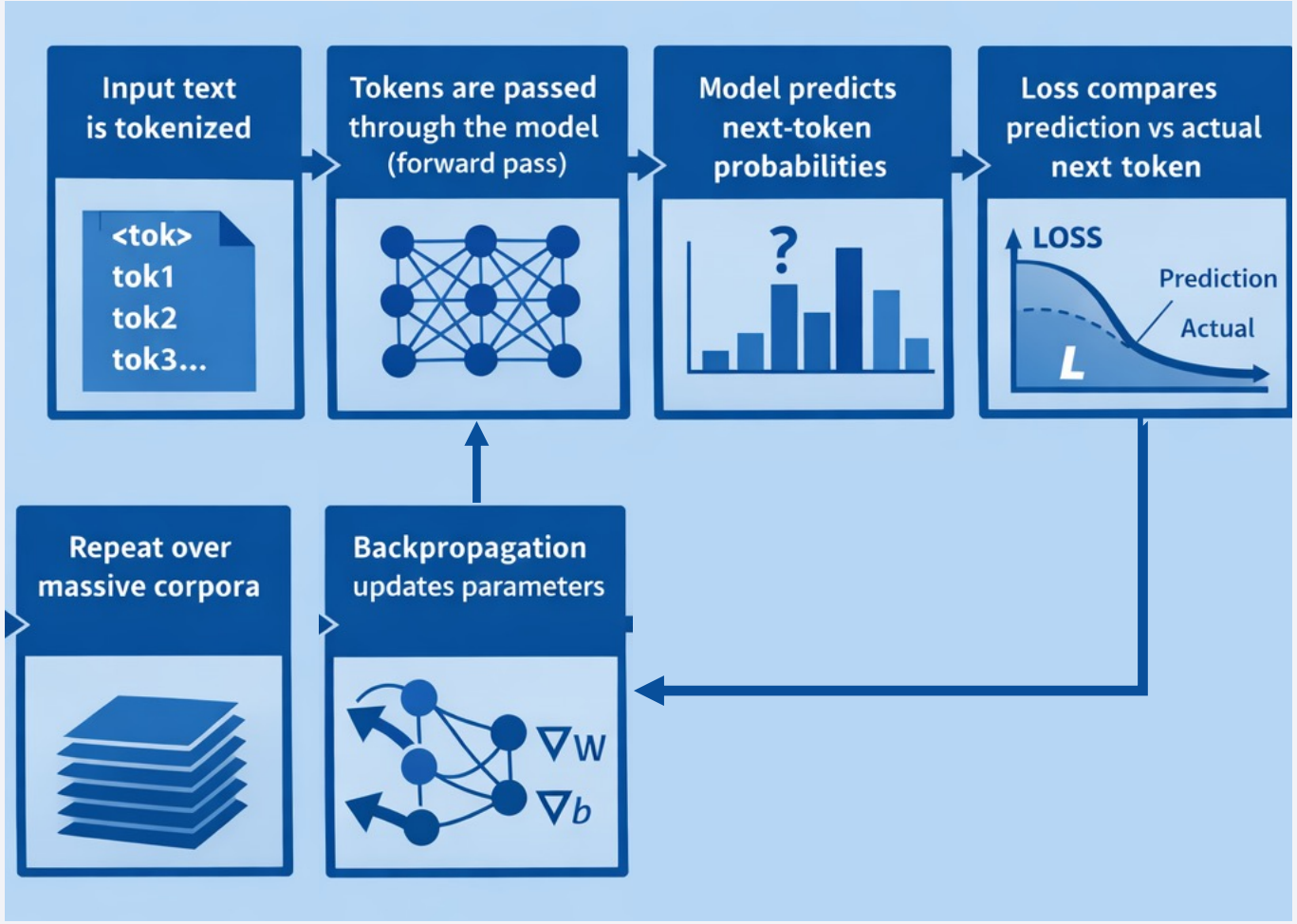
Inference:
All new data

Training → Testing

Teaches the model to **predict the next token** given context.

Training

1. Input text is tokenized
2. Tokens are passed through the model (forward pass)
3. Model predicts next-token probabilities
4. Loss compares prediction vs actual next token
5. Backpropagation updates parameters
6. Repeat over massive corpora



Training → Testing

Testing / Evaluation (How We Measure LLMs)

Traditional testing

- Held-out datasets
- Perplexity, accuracy on benchmarks

Why this is insufficient

Linguistic fit ≠ reasoning, reliability, or usefulness

- Benchmarks leak into training data
- Performance depends on prompt framing
- IID assumptions often fail

- **Held-out data**
  Unseen text used only for evaluation (assumes same distribution)
- **Perplexity**
  How well the model predicts the next token
  ↓ perplexity = better linguistic prediction
- **Benchmark accuracy**
  Correct / incorrect on fixed tasks (QA, MCQ, classification)
- **Core limitation**

Training → Testing

Modern evaluation adds

- Task-based evaluation (QA, summarization, classification)
- Robustness checks (paraphrases, domain shifts)
- Human evaluation (helpfulness, safety, preference)
- Stress tests (long context, adversarial prompts)

Critical point

High benchmark scores ≠ reliable real-world behavior.

Unlabeled vs Labeled Data

Unlabeled Data (Pretraining Data)

Raw text with **no human-assigned labels**

No tags like sentiment, topic, or correct answer

How learning happens without labels

The *label is implicit*: the next token in the sequence

What the model learns

Grammar and syntax

Semantic regularities

World correlations and patterns

Statistical structure of language

Examples

Books, articles, websites, code repositories

Conversations without ratings or instructions

Unlabeled vs Labeled Data

Labeled Data (Post-Training / Fine-Tuning Data)

Text paired with **explicit human intent or judgment**

Labels are often *implicit in the format*, not numeric

What labels provide

Task definition

Behavioral guidance

Alignment with human preferences

How labels are used

Supervised fine-tuning

Preference optimization (e.g., RLHF-style methods)

Evaluation and benchmarking

Examples

Instruction → ideal response

Question → verified answer

Prompt → preference ranking

Input → safety classification

Data splits:

IID Assumption in DL

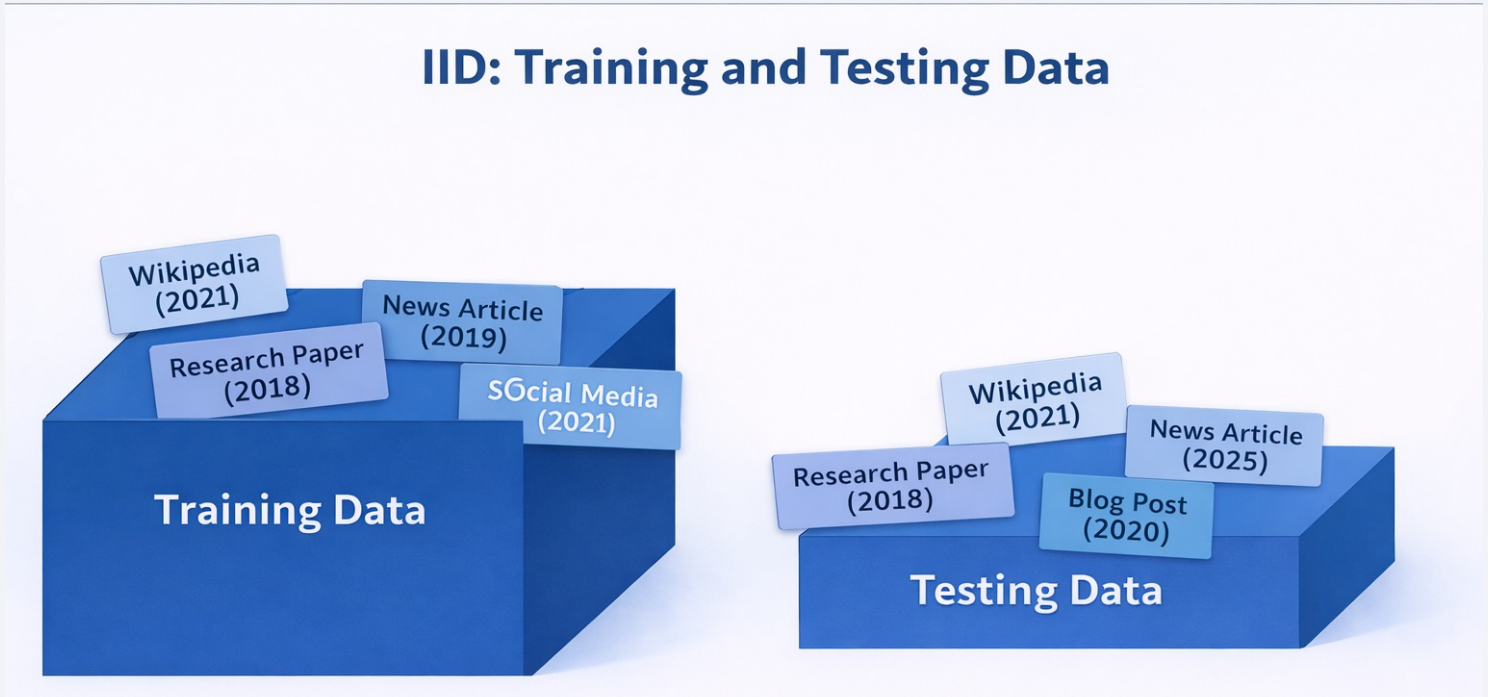
IID: Independently Identically Distributed

IID: train and test look the same

Non-IID: train and test differ

- Time
- Domain
- Author

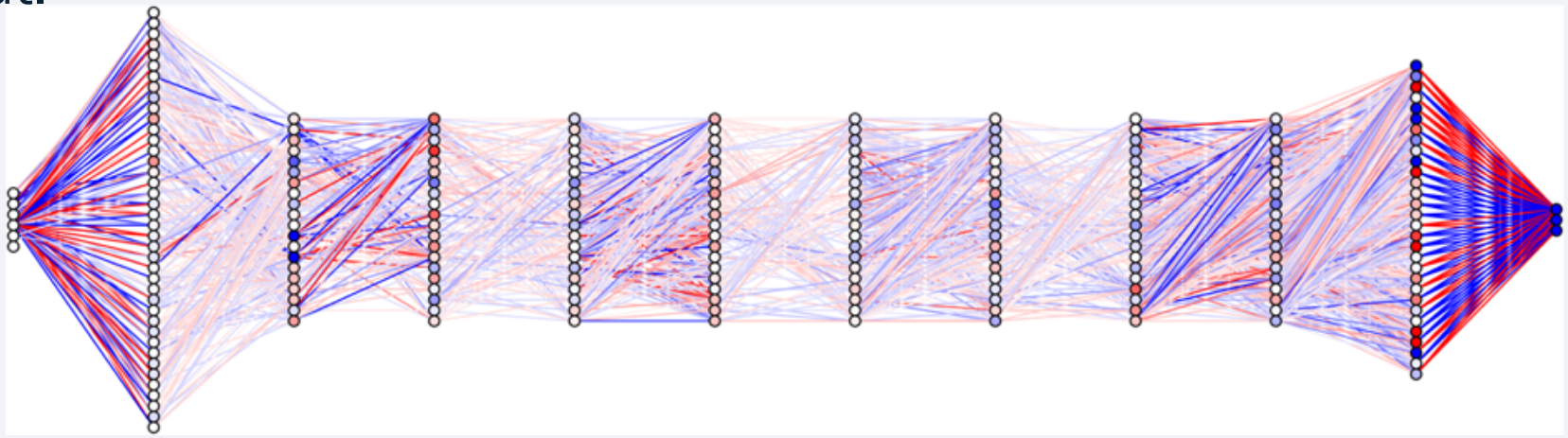
Problem: real use is non-IID



LLM Training Mechanics: How Parameters Learn

Neural Network Basics

- A neural network is a stack of layers (simple mathematical functions).
- Each layer transforms data step by step ⟶ from raw input to meaningful output.

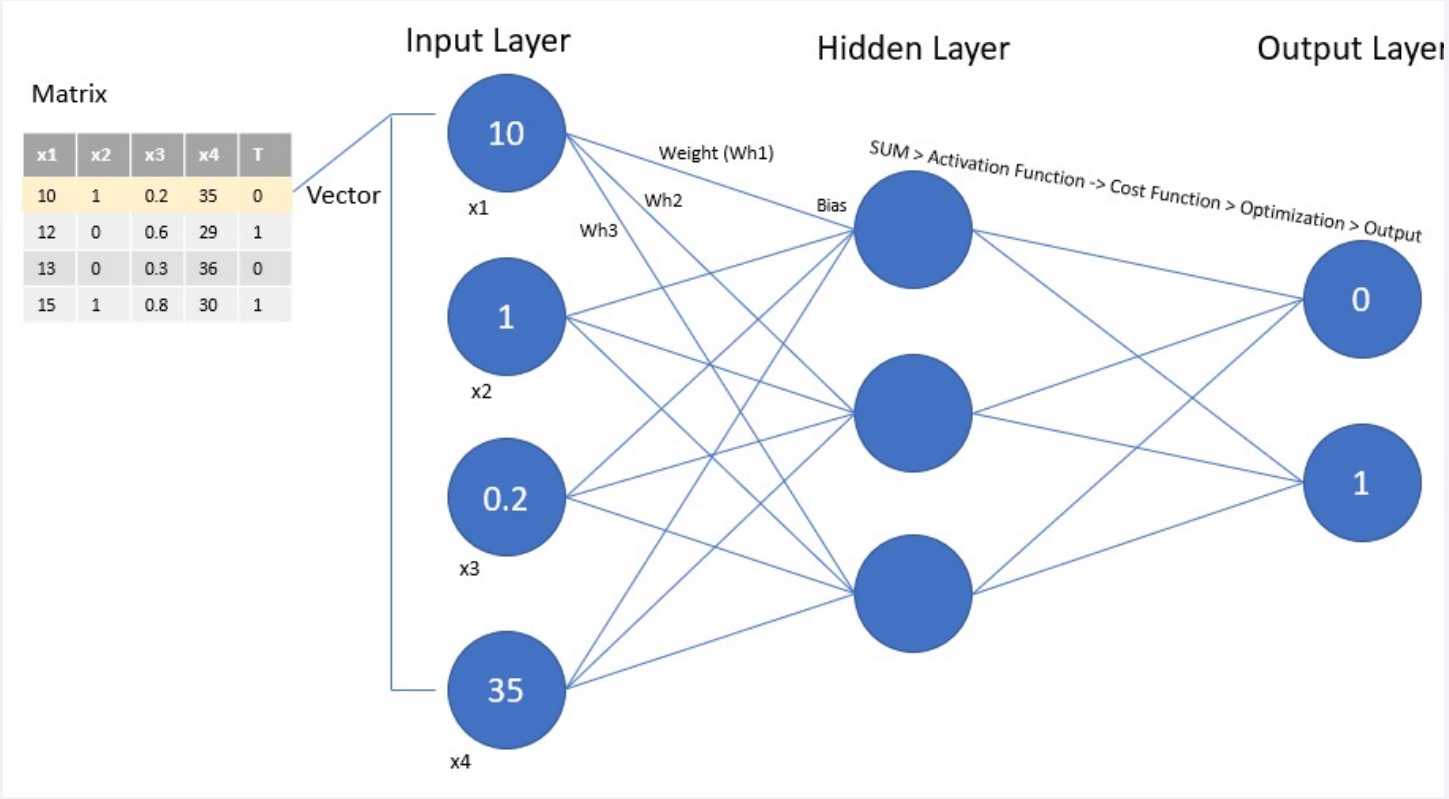


What makes the model learn is the weights, which adjust to fit patterns in the data.

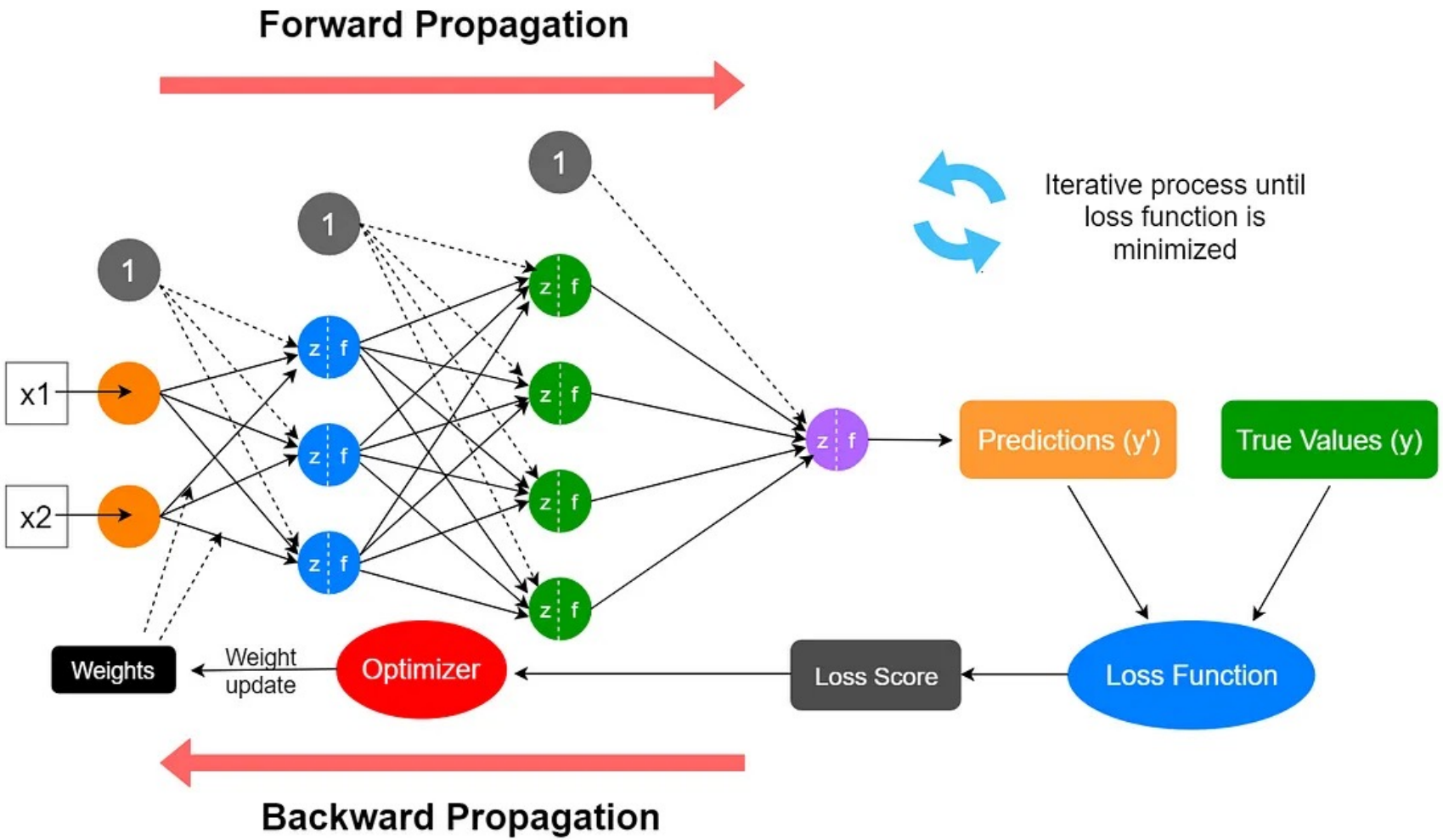
What Are Weights?

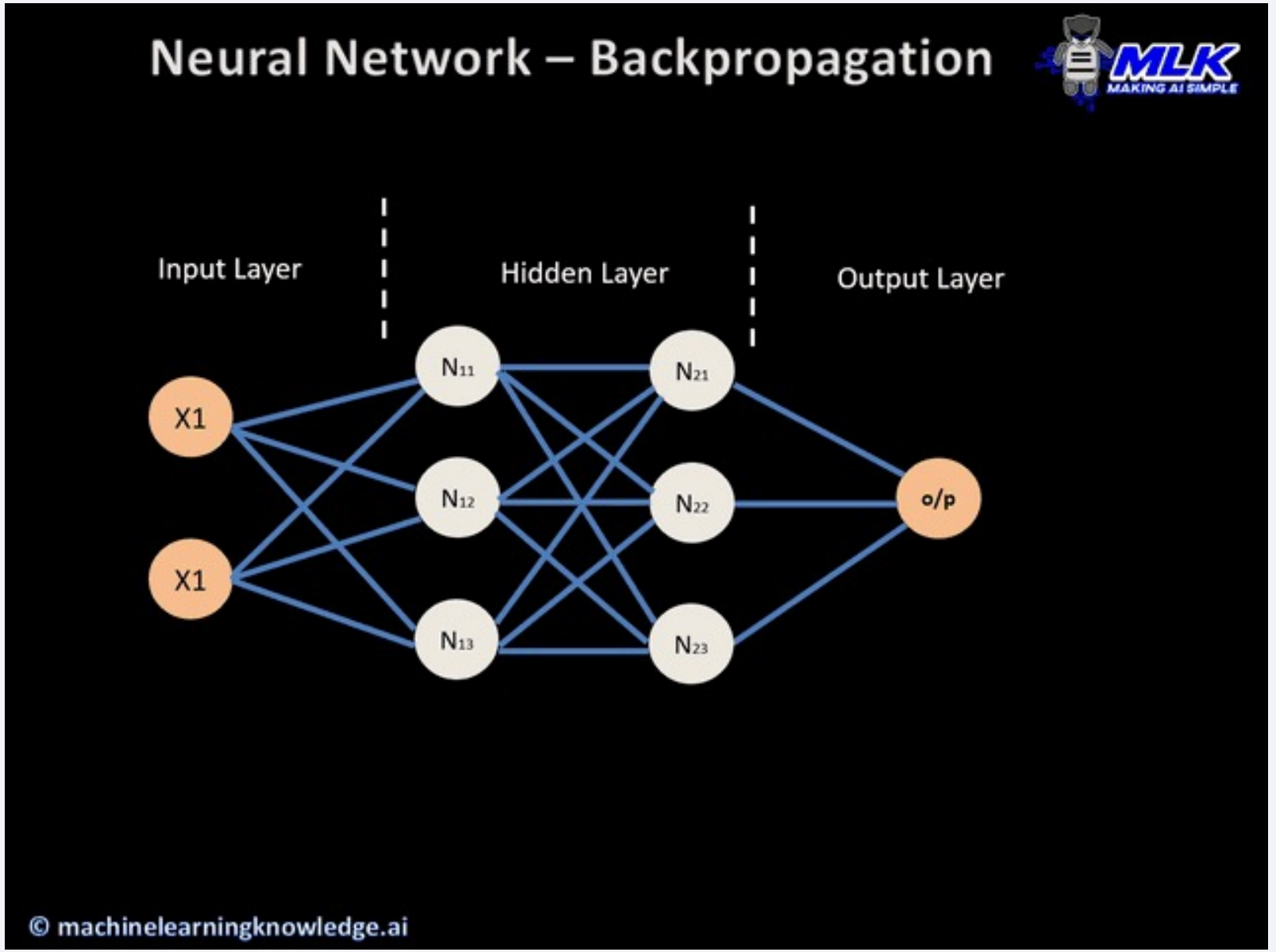
The weights ARE the model

- Each connection between layers has a **weight = strength of influence.**
- At first: random weights → random guesses.
- **During training:** weights are adjusted to reduce error.
- **End result:** the set of all weights = the trained knowledge of the model.



Training


Forward Propagation
1
1
1
x1
x2
z f
Iterative process until loss function is minimized
Predictions (y')
True Values (y)
Loss Function
Loss Score
Optimizer
Weight update
Weights
Backward Propagation

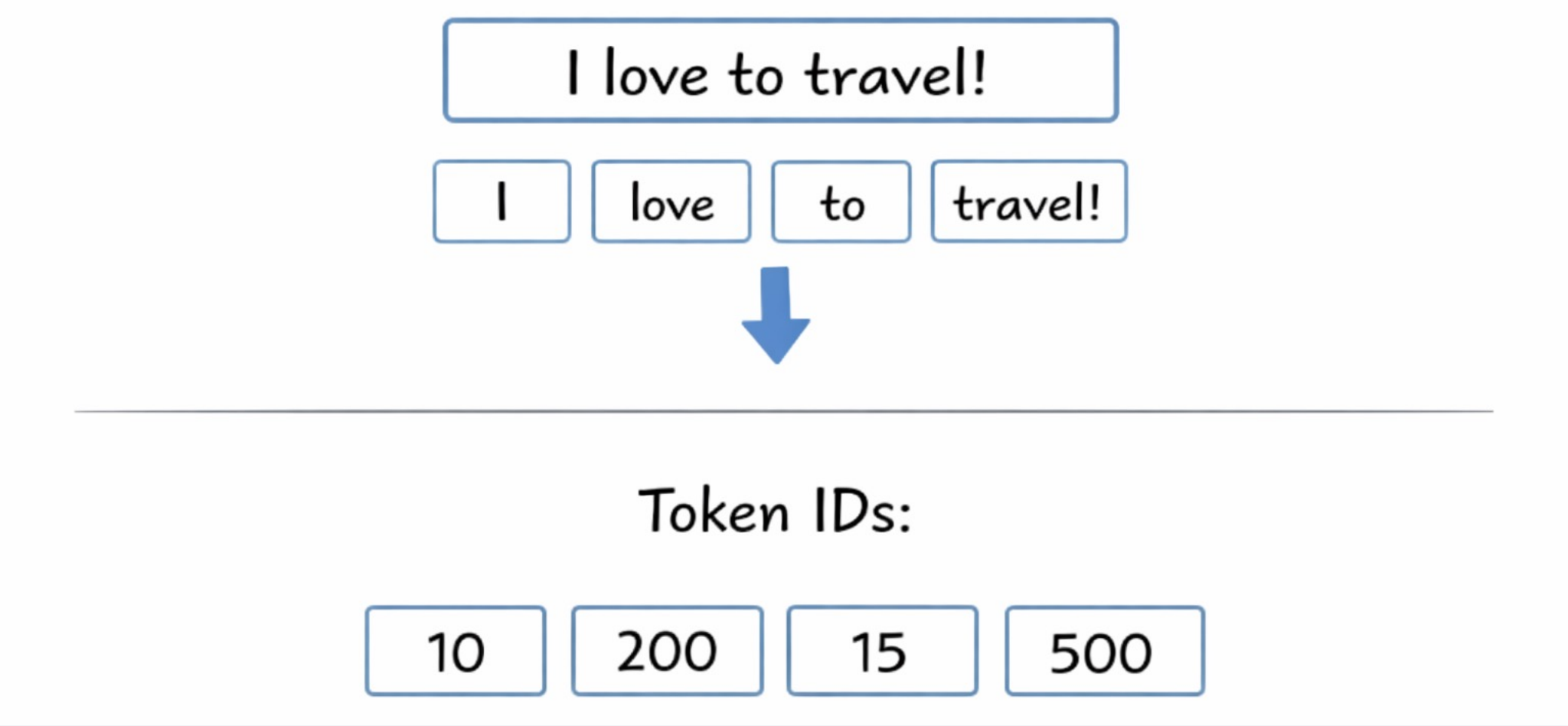

Neural Network – Backpropagation
MLK
MAKING AI SIMPLE
Input Layer
Hidden Layer
Output Layer
X1
X1
N11
N12
N13
N21
N22
N23
o/p
© machinelearningknowledge.ai

Parameters & Model Size

- **Parameters** = all the learnable **weights** and **biases** in the model.
- Model size is measured in **number of parameters** (e.g., 110M, 1B).
- Bigger ≠ always better :
  - more parameters = more capacity but
  - more compute + more data needed.
- **Model cards** report this so you can gauge complexity and hardware needs.

But how do these parameters actually get used and updated?

How DL Basics Manifest in LLMs



From Text to Numbers: Tokenization

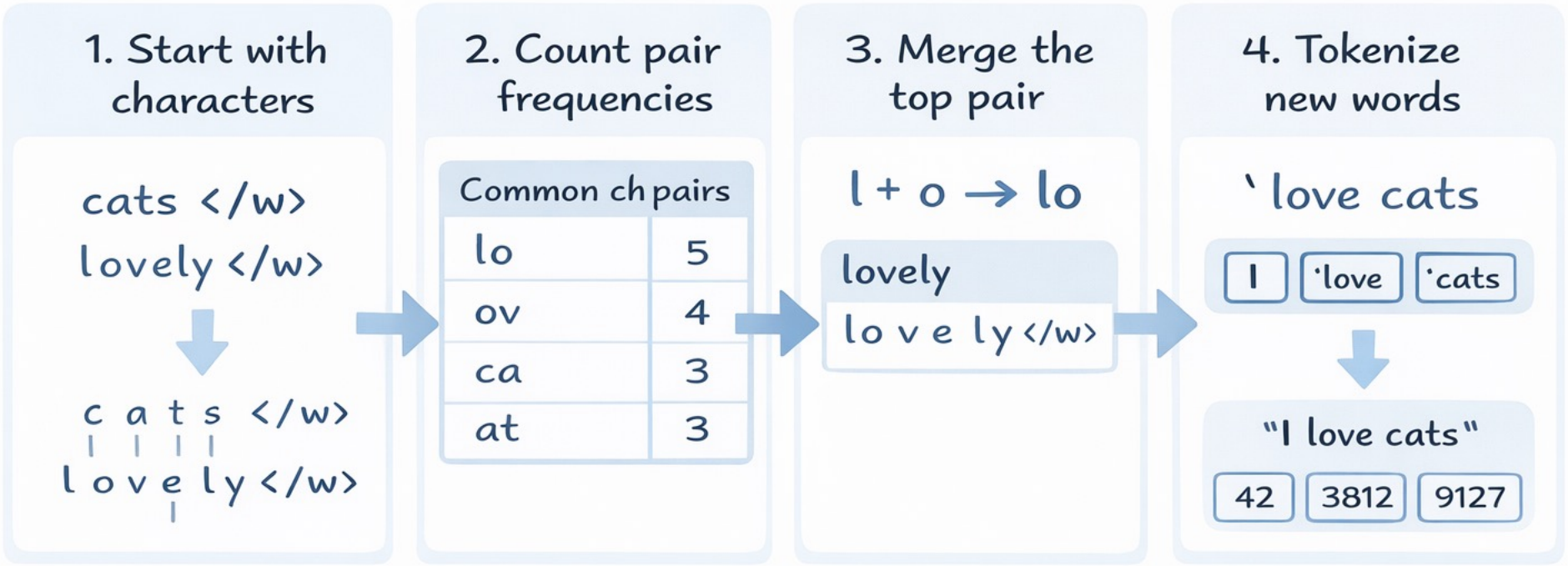
A **token** is a basic unit like a word, subword, or character.

Each token is assigned an integer ID and then turned into a vector (embedding) that the model actually works with.

Common tokenization styles

- **Word-based**: splits on spaces, so each word is a token; simple but breaks on unknown words and different spellings.
- **Character-based**: each character (including spaces) is a token; handles anything but sequences become long.
- **Subword-based**: splits words into frequent pieces using methods like BPE, WordPiece, or Unigram; this is what most modern LLMs use.

Subword Tokenization (BPE)



When tokenizer is constructed (ONCE)
At the final stage, the vocabulary becomes fixed and IDs are assigned

Why BPE is popular

- Solves out-of-vocabulary words
- Balances vocabulary size
- Language-agnostic

```
"tokenization" → ["token", "ization"]
"antidisestablishmentarianism"
→ ["anti", "dis", "establish", "ment", "arian", "ism"]
```

```
"I love cats"
→ ["I", " love", " cats"]

"catlike"
→ ["cat", "like"]

"cat-ownership"
→ ["cat", "-", "owner", "ship"]
```

```
English:   unbelievable → ["un", "believe", "able"]
German:    unglaublicher → ["unglaub", "lich", "er"]
French:    malheureusement → ["mal", "heureuse", "ment"]
```

Tokenization → Embeddings → Model Input

- Raw text is split into **subword tokens** using a fixed tokenizer (e.g. BPE)
- Each token is mapped to a **unique integer ID**
- Token IDs index rows in a **learned embedding matrix**
- Each token becomes a **dense vector** (token embedding)

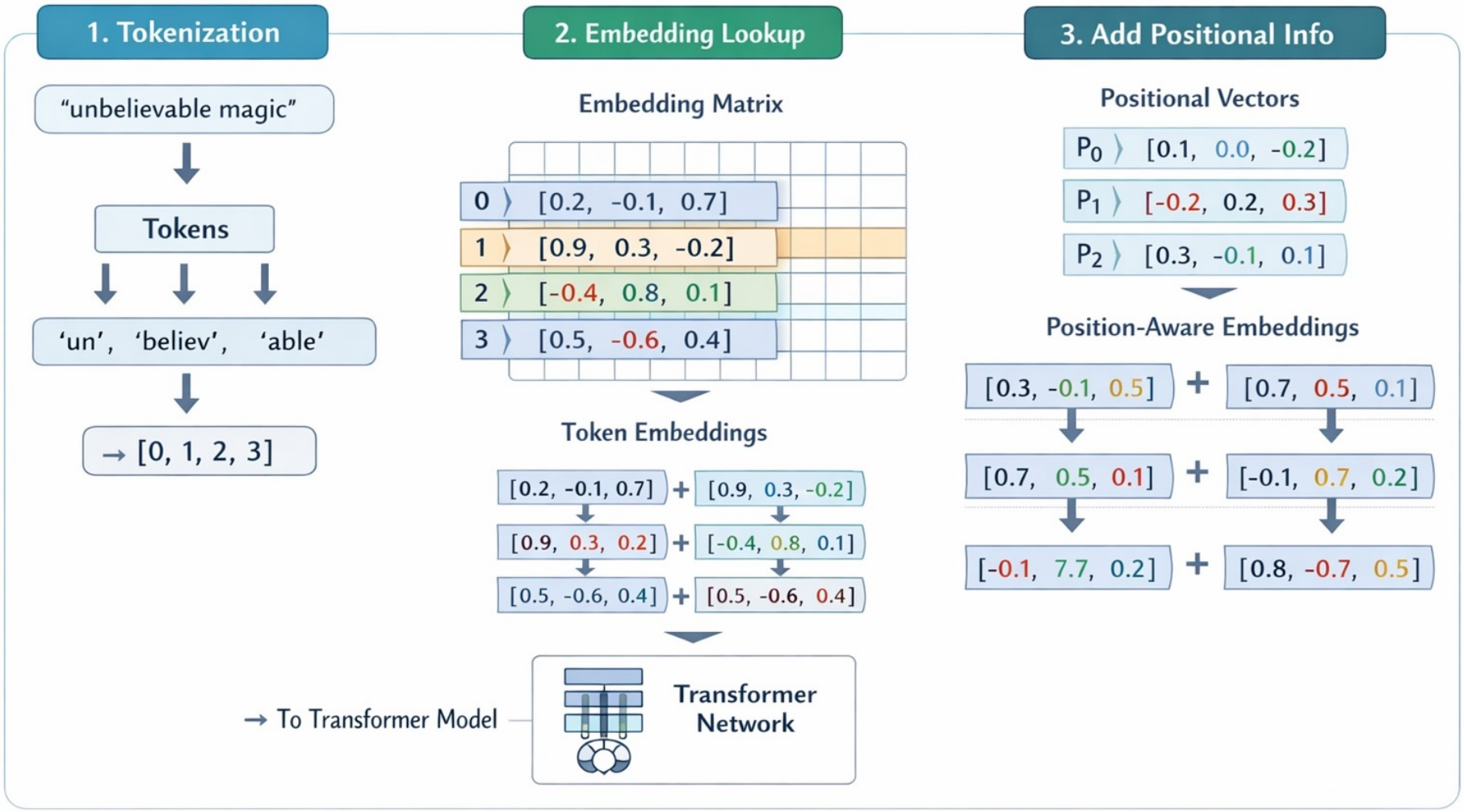
Positional vectors are added to encode order

- Resulting vectors are the **input to the first Transformer layer**

During training (what shapes embeddings)

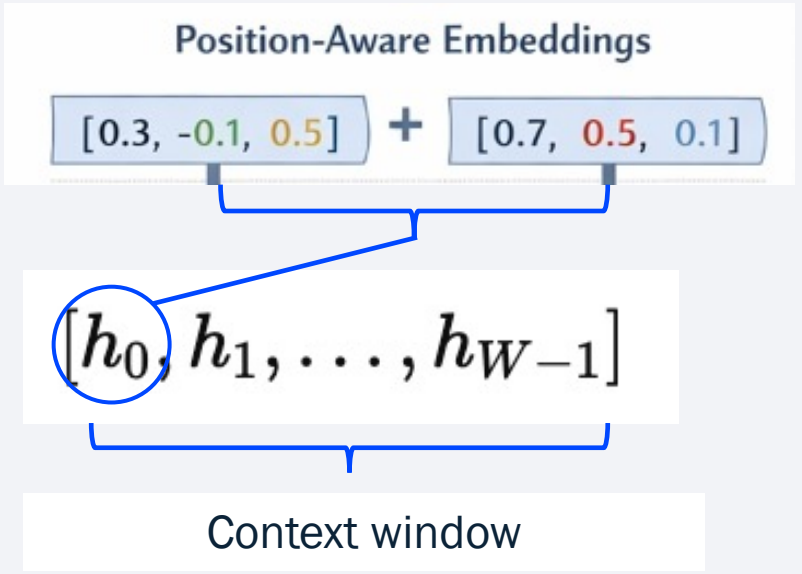
- Model predicts the **next token** in the sequence
- Loss compares predicted vs actual next token
- Gradients backpropagate through the network
- **Only embeddings of tokens seen in the batch are updated**
- Tokens used in similar contexts move **closer in embedding space**

Embedding Process Example



Context window

- Maximum number of tokens processed at once
- Defines the model's active working set
- Tokens outside the window do not exist to the model
- Independent of token meaning or position



Transformer Block: Minimal Components

- A Transformer block is a **repeated computation unit** applied to the whole sequence.

Includes

- Self-attention: tokens share information
- Feed-forward: tokens refine information
- Residuals: preserve earlier signals
- Normalization: stabilize computation
- Stacking blocks increases capacity, not context

Self-Attention = Weighted Mixing of Token Vectors

1. Compare

"unbelievable magic"

Token 0	Token 1
[1.0, 0.0]	[0.8, 0.2]
"believ"	"magic"

Token 0: [1.0,0] · 0.8, 0.2]
[1,0,0] · 0.0, 1.0]

2. Calculate Weights

Scores	Weights
0.8	1.0
0.0	0.0

Score	Weights
0.8	1.0
0.0	0.0

Scores: 0.8 → 0.0 [1,0.0]
Weights: 1.0 → 0.0 [0.0.0]

3. Generate New Vector

$w_1 \times$ Token 1 + $w_2 \times$ Token 2

1.0 × [0.8.2]

= w_1 [0.8.2] + 0.0 × [0,0,1.0]

0.8, 2.2]

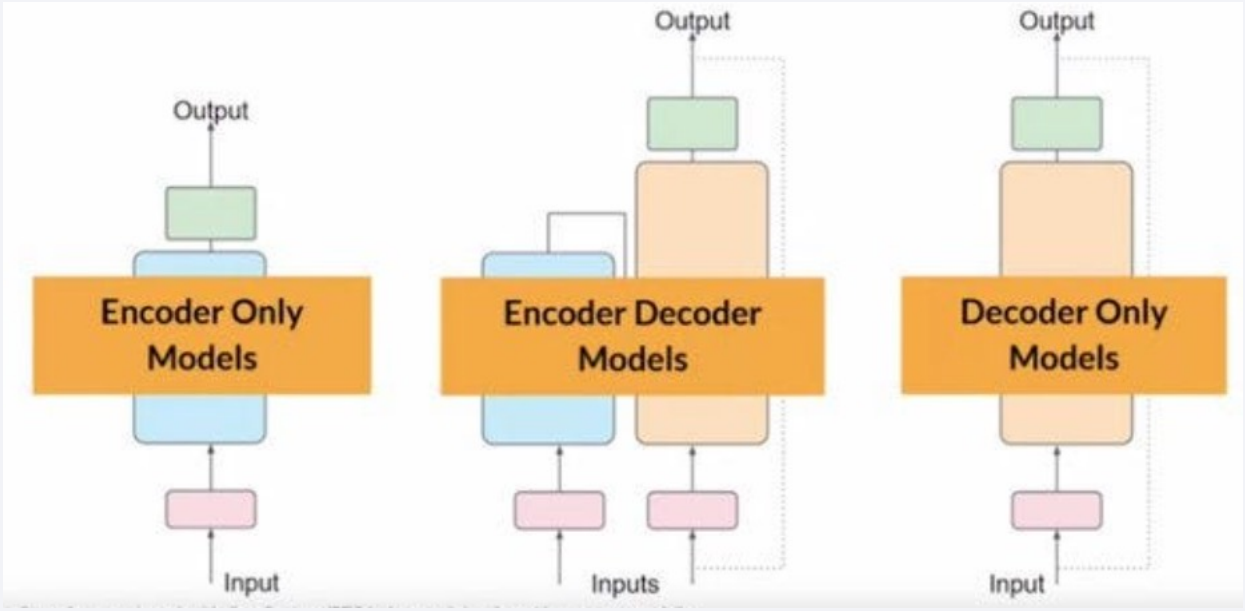
0.8 × [0.8.2] + 0.0 × [0.01,1,10]

Maria Aise

Modern neural architectures

Based on how data flows through the model.

- Encoder-only
- Decoder-only, and
- Encoder–decoder.



Encoder

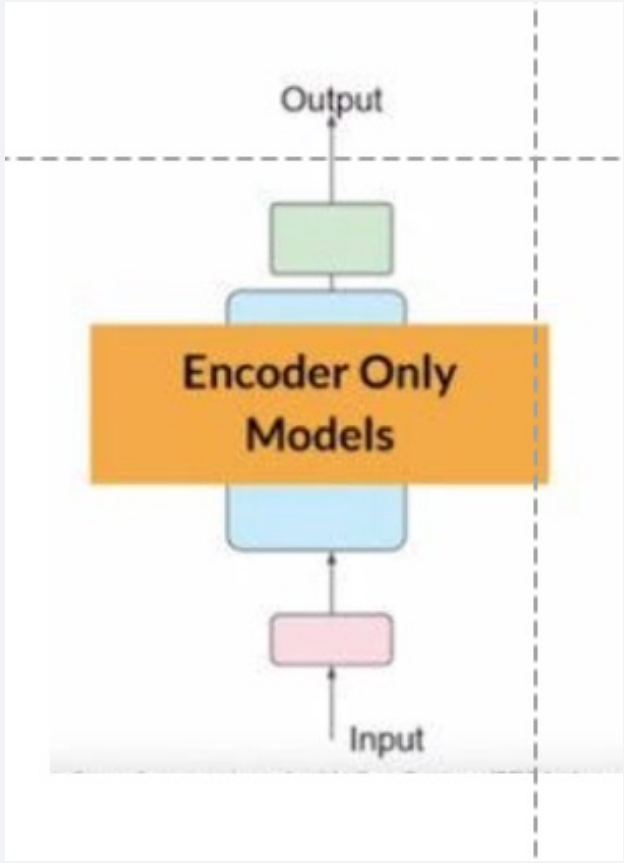
A skilled **note-taker** → reads a whole book and produces a clear summary.

- An **encoder** is the part of a model that **reads the input** and builds an internal summary (representation).
- Think of it as a **compressor**: it takes raw messy data and turns it into a compact, meaningful code.

Examples:

- BERT (text): turns sentences into embeddings used for classification or search.
- ViT (vision): turns image patches into a feature vector for classification.

Good for: classification, clustering, feature extraction.



Decoder

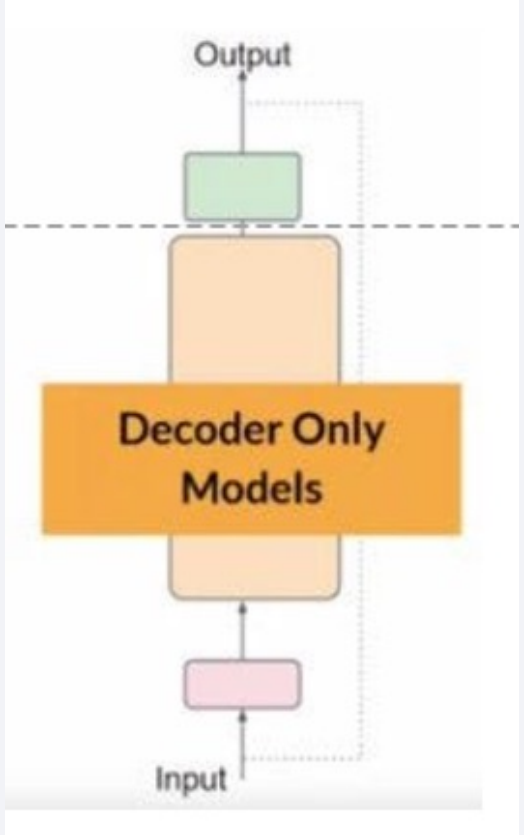
A skilled **storyteller** → takes a summary and tells the full story in words

- A **decoder** is the part of a model that **generates new outputs** from a representation.
- It expands a compressed meaning back into a sequence (text, audio, image).

Examples:

- GPT (text): generates next words one by one.
- TTS models: generate audio waveforms from text.

Good for: translation, text generation, image captioning, TTS.



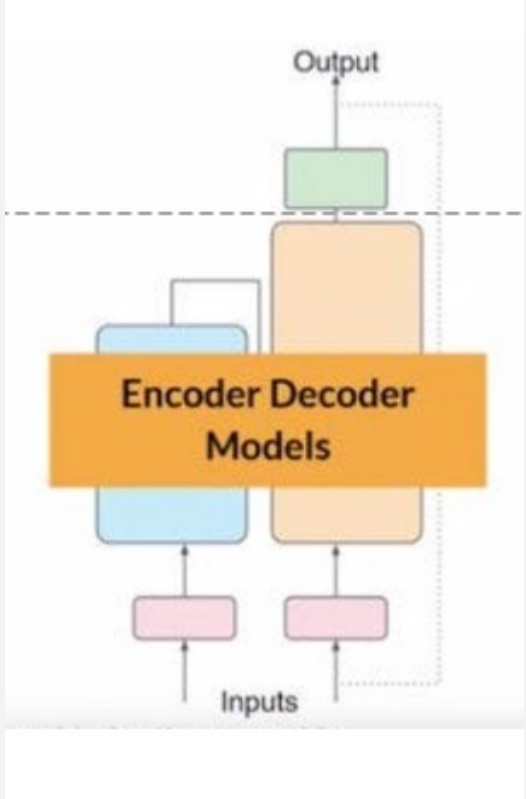
Encoder–Decoder Together

The **seq2seq pipeline** combines both:

1. **Encoder** compresses input (e.g. French sentence).
2. **Decoder** generates new sequence (English translation).

Examples:

- Whisper (speech ⟶ text): audio in ⟶ encoder ⟶ text out via decoder.
- Machine translation: one language in ⟶ encoder ⟶ another language out via decoder.



Practice