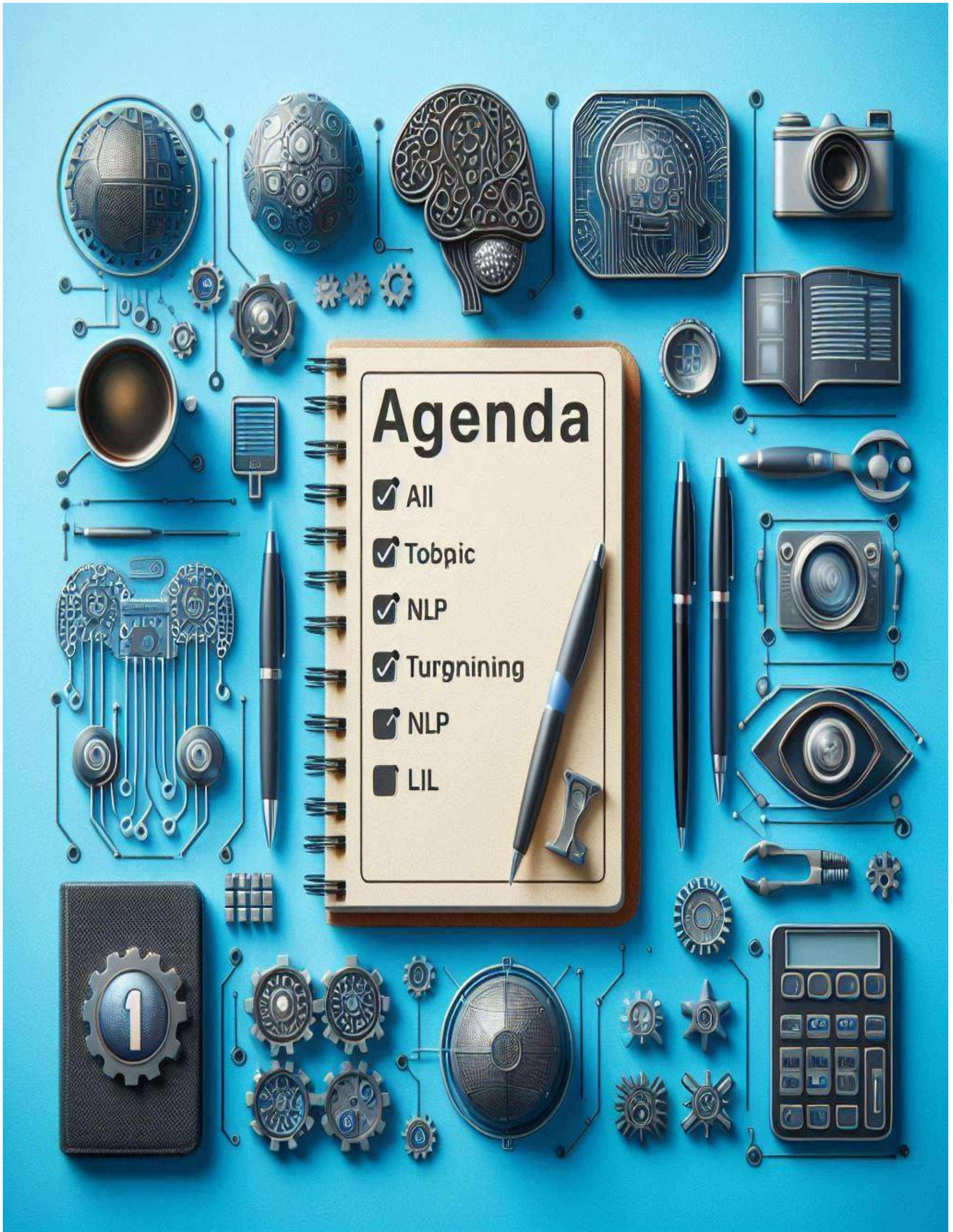
NTI
National Telecommunication Institute

What are your expectations for the course?

Agenda



- Introduction to Sequential Data
- Limitations of Feedforward Networks
- Introduction to RNNs
- Deep Dive into LSTM
- Common Use Cases of LSTM
- Introduction to GRU
- Comparison: LSTM vs GRU vs RNN

NTI
National Telecommunication Institute

What is Sequential Data?

- Ordered data (sentences, time-series, audio)
- Requires context and memory

Examples:

Text/Sentences:

The sentence "The cat sat on the mat" has a meaning because of word order. Shuffle the words, and the meaning may change.

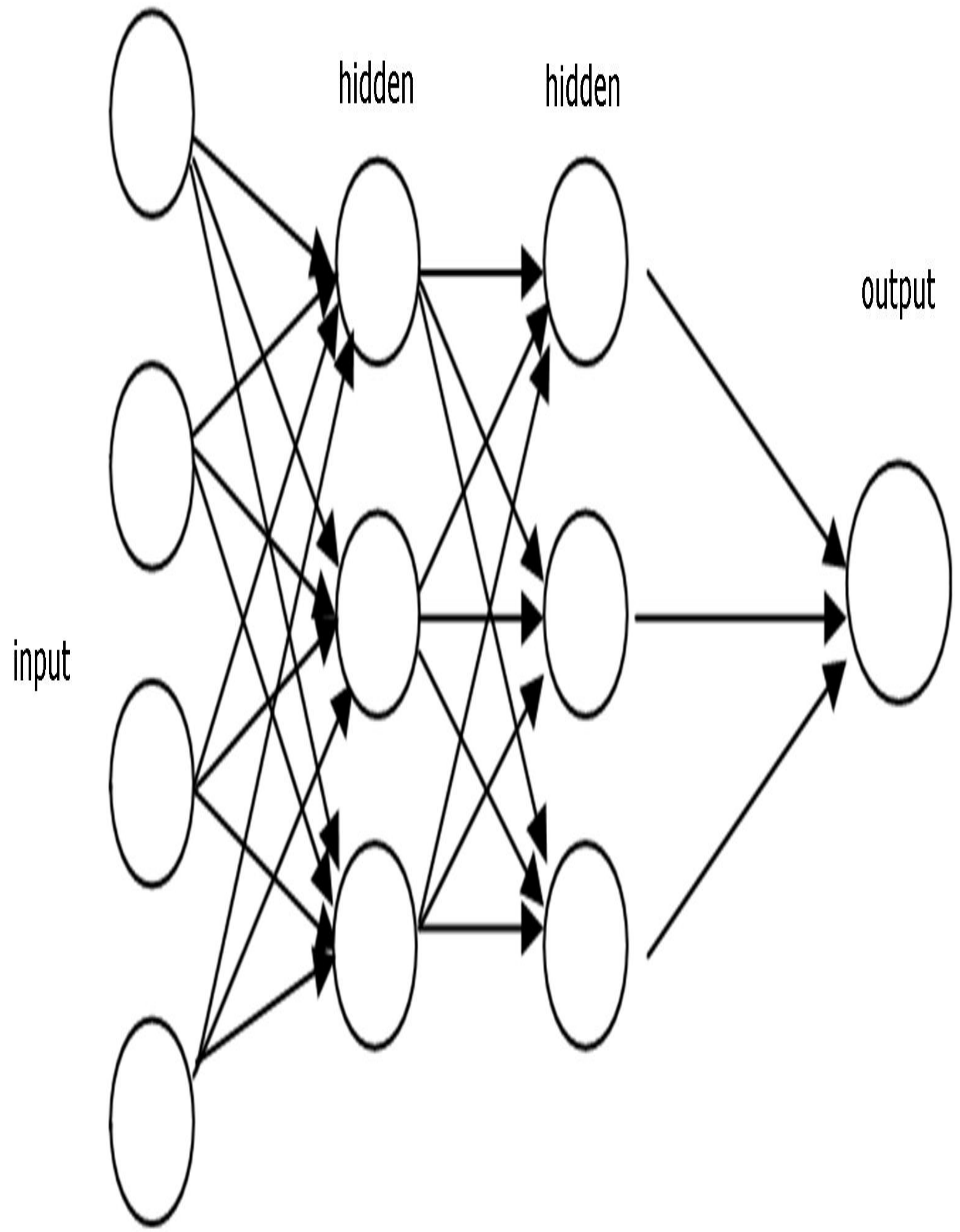
Time-Series Data:

Stock prices, temperature readings — each value depends on the time before it.

Audio Signals:

Speech is made of waveforms that change over time. The order of sound frequencies determines the spoken word.

Why Not Feedforward Networks?



- Feedforward Neural Networks (FNNs) are excellent for many tasks like classification and regression on structured/tabular data, but they fail when it comes to sequential data. Why?

Limitations:

No Memory of the Past:

FNNs process each input independently.
They can’t retain information from earlier parts of the input.
For example, if you're translating a sentence, FNNs can't remember what was said earlier.

No Sense of Order:

In sequential data, order matters.
"She is not happy" ≠ "She is happy not"
→ FNNs treat both as unrelated since they lack positional context.

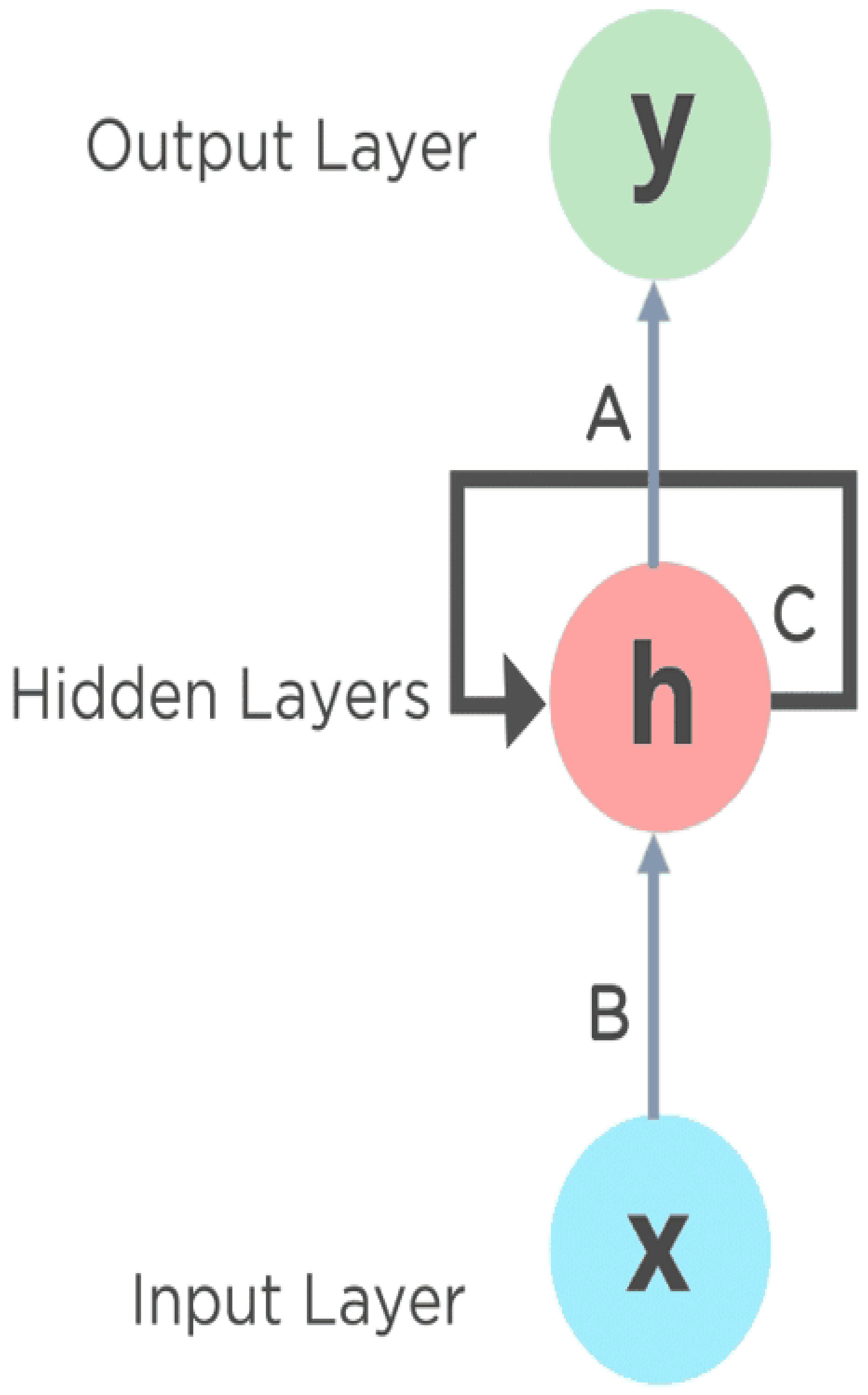
Contextual Dependency is Ignored:

If you’re analyzing time-series data like stock prices, the next value heavily depends on the previous ones.
FNNs assume inputs are independent, so they can’t model such dependencies.

Fixed Input Size:

FNNs usually require a fixed-size input.

National Telecommunication Institute



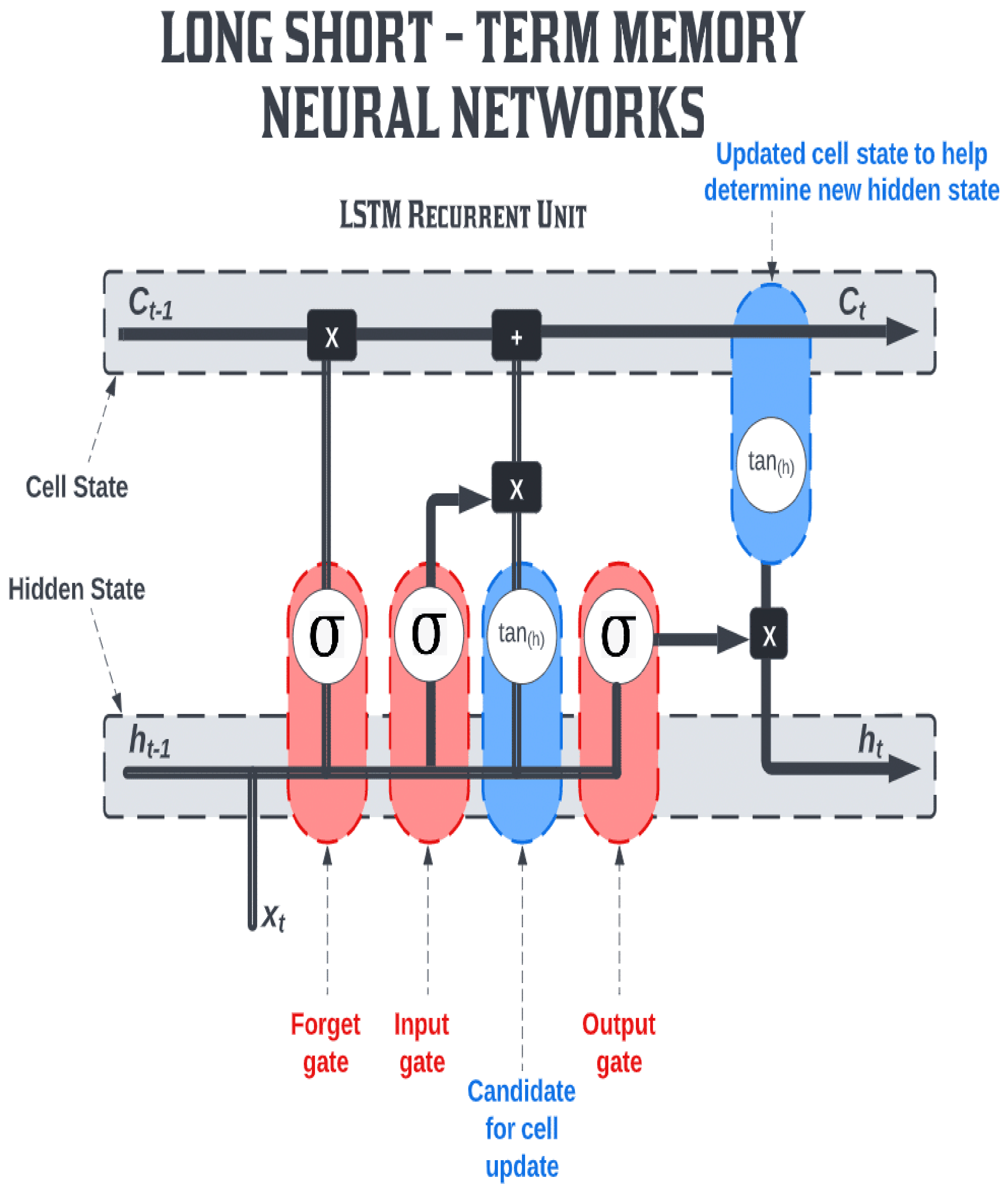
A, B and C are the parameters

Recurrent Neural Networks (RNN)

- Maintains hidden state
- Designed for sequences
- Struggles with long-term dependencies

  (Vanishing Gradient Problem)

What is LSTM?



LSTM (Long Short-Term Memory) is a special type of Recurrent Neural Network (RNN) architecture designed to **learn long-term dependencies** in sequential data.

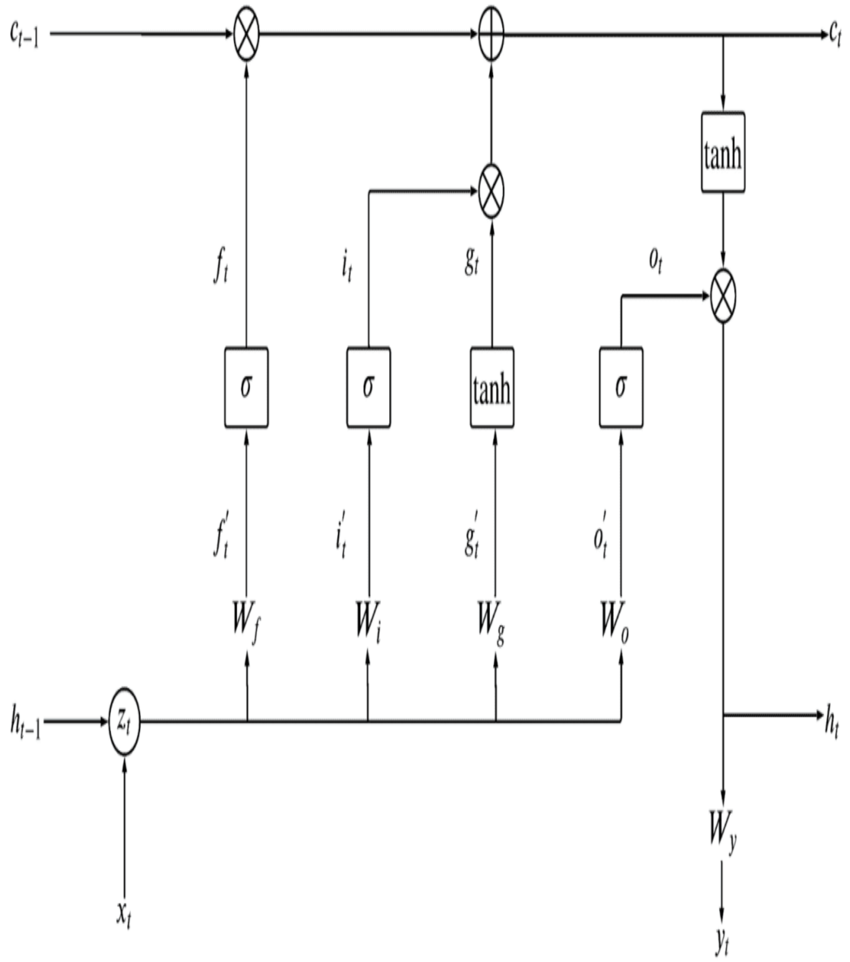
Proposed by *Hochreiter & Schmidhuber (1997)*.

Unlike vanilla RNNs, it has an **internal memory (cell state)** that can carry information across many time steps.

Uses **gates** (forget, input, output) to control the flow of information.

Why “Long” and “Short”-Term?

Long Short-Term Memory Networks



•The name reflects its ability to manage **short-term memory** (current context) **and long-term memory** (important past context).

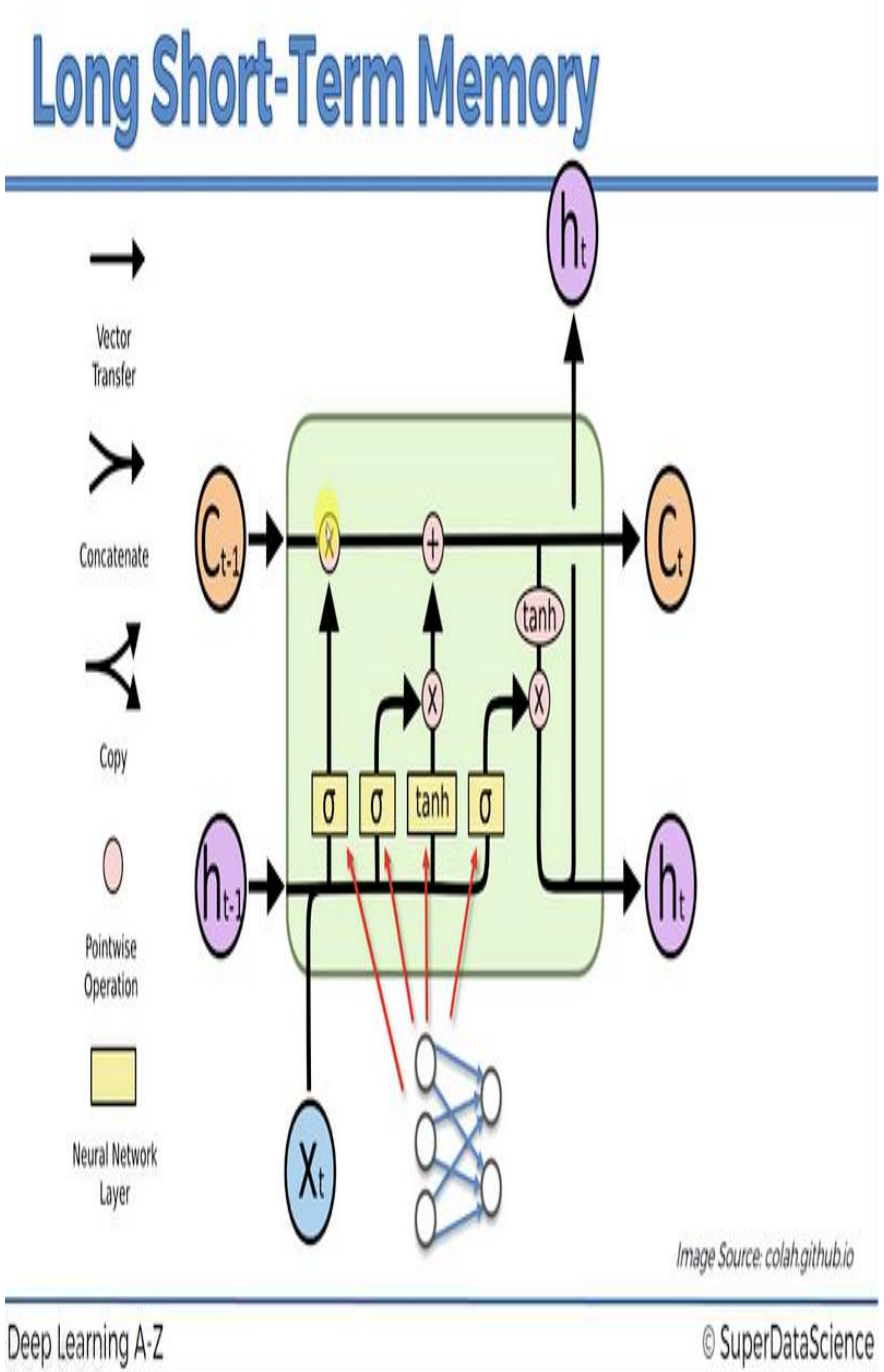
For example, in the sentence:

“The **cat** that was chased by the **dog** ran under the **car**,”

The network might need to remember “cat” at the beginning to understand what “ran” refers to at the end.

LSTM Solves:

- **Vanishing gradient problem** in long sequences
- **Maintains memory over time**
- **Learns complex patterns** across long texts, audio, and time-series data



LSTM Architecture Overview

Think of LSTM like a **smart notebook**:

- It writes down **important things** (input gate).
- It erases **unnecessary things** (forget gate).
- It decides what to **share with the n1ext layer or step** (output gate).

This makes LSTMs excellent at remembering **context** over long sequences — something vanilla RNNs fail at due to the *vanishing gradient problem*.

Imagine predicting the next word in this sentence:

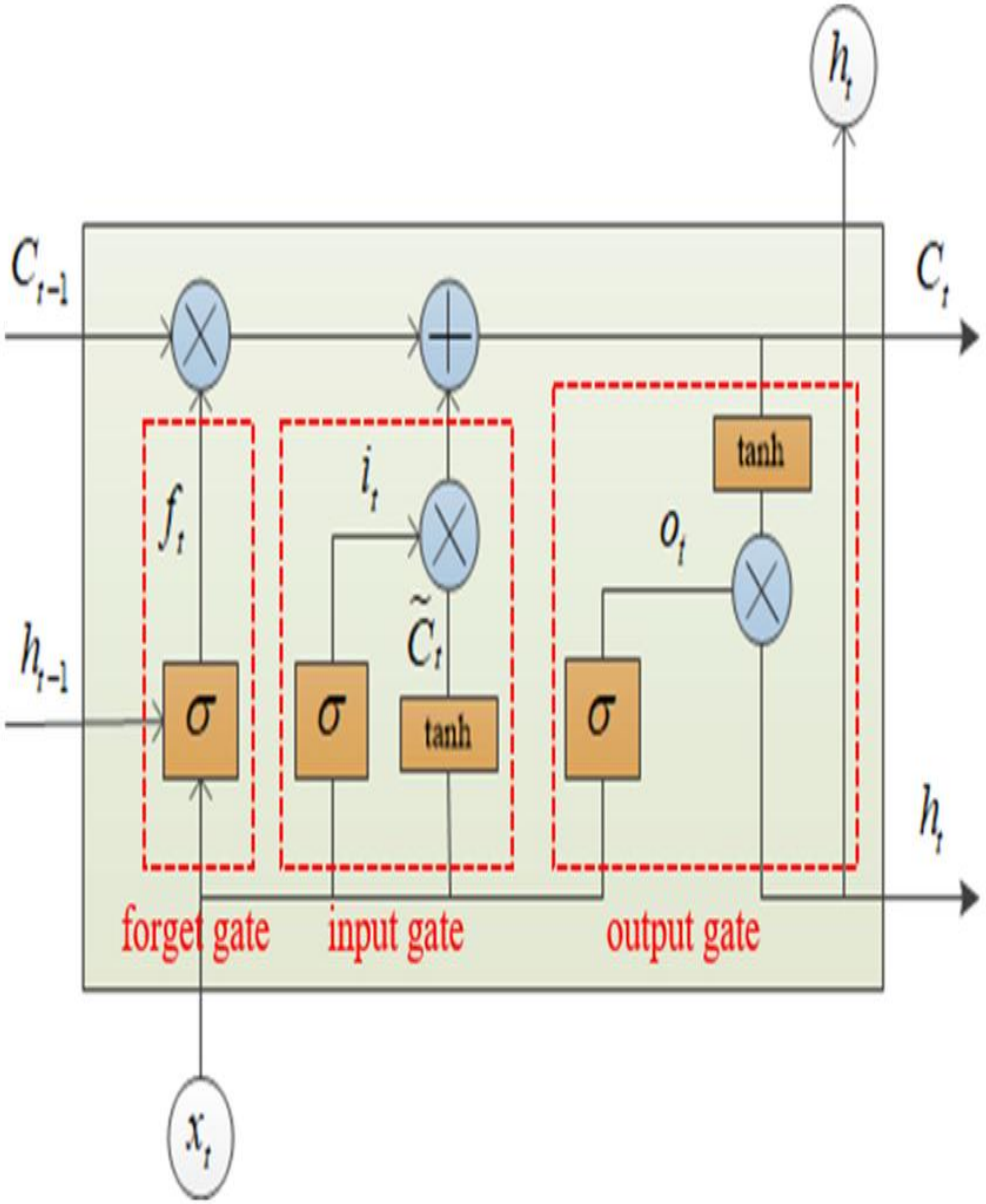
"I grew up in **France**… I speak fluent ___."

A vanilla RNN might forget "France" after a few steps.

An LSTM **remembers** the country information across many words and predicts **French** correctly.

➢ LSTM = RNN + *memory cell* + *gates* → allows learning **long-term dependencies** in sequential tasks.

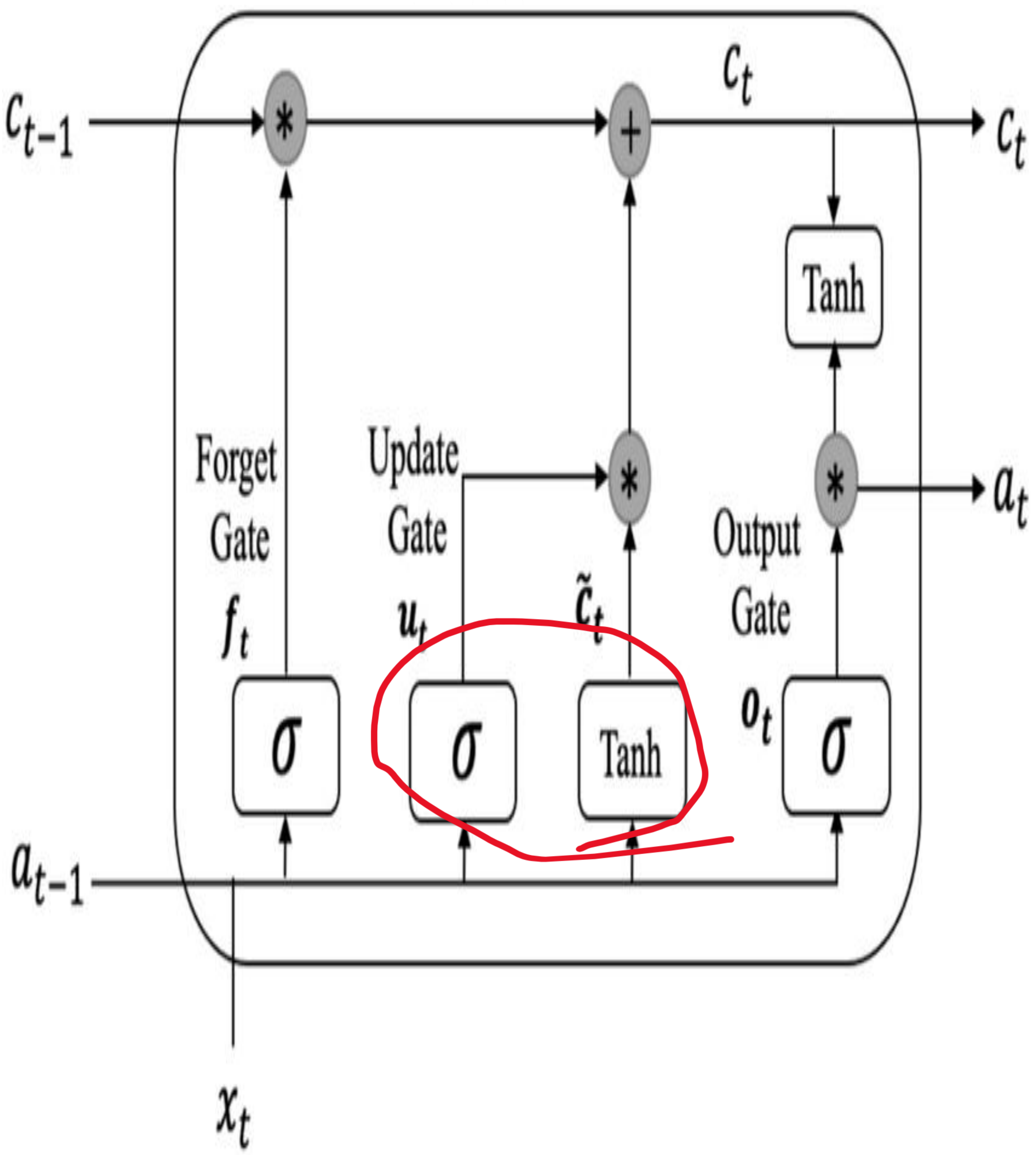
Forget Gate



- The **Forget Gate** is the first decision point inside the LSTM cell.
- Its job is to **decide which information from the previous cell state should be removed or kept**.
- Sometimes, not all past information is useful for future predictions.
- So, the LSTM needs to "forget" parts of its memory that are no longer relevant.

Formula:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$



Input Gate

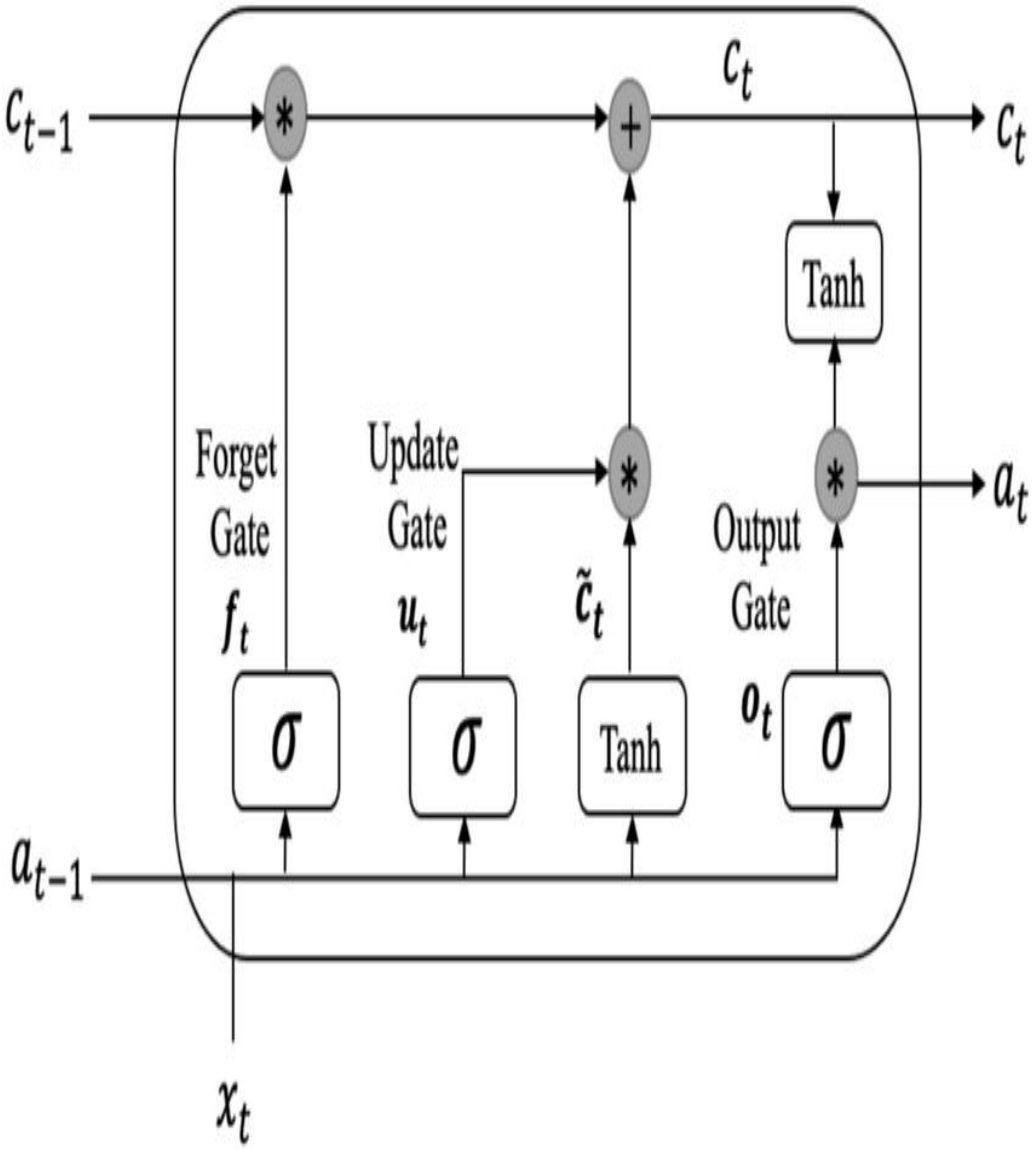
After deciding what to forget, the LSTM now asks:
“What new information should I store in my memory?”
That’s the role of the **Input Gate**.
The **Input Gate** controls which parts of the current input (and the previous hidden state) are important enough to **add to the cell state (memory)**.

Gate Activation:

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

Candidate Memory:

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$



Update Gate

Now that we know what to forget (Forget Gate) and what new information to add (Input Gate),

it's time to **actually update the memory** — this happens in the **Cell State**.

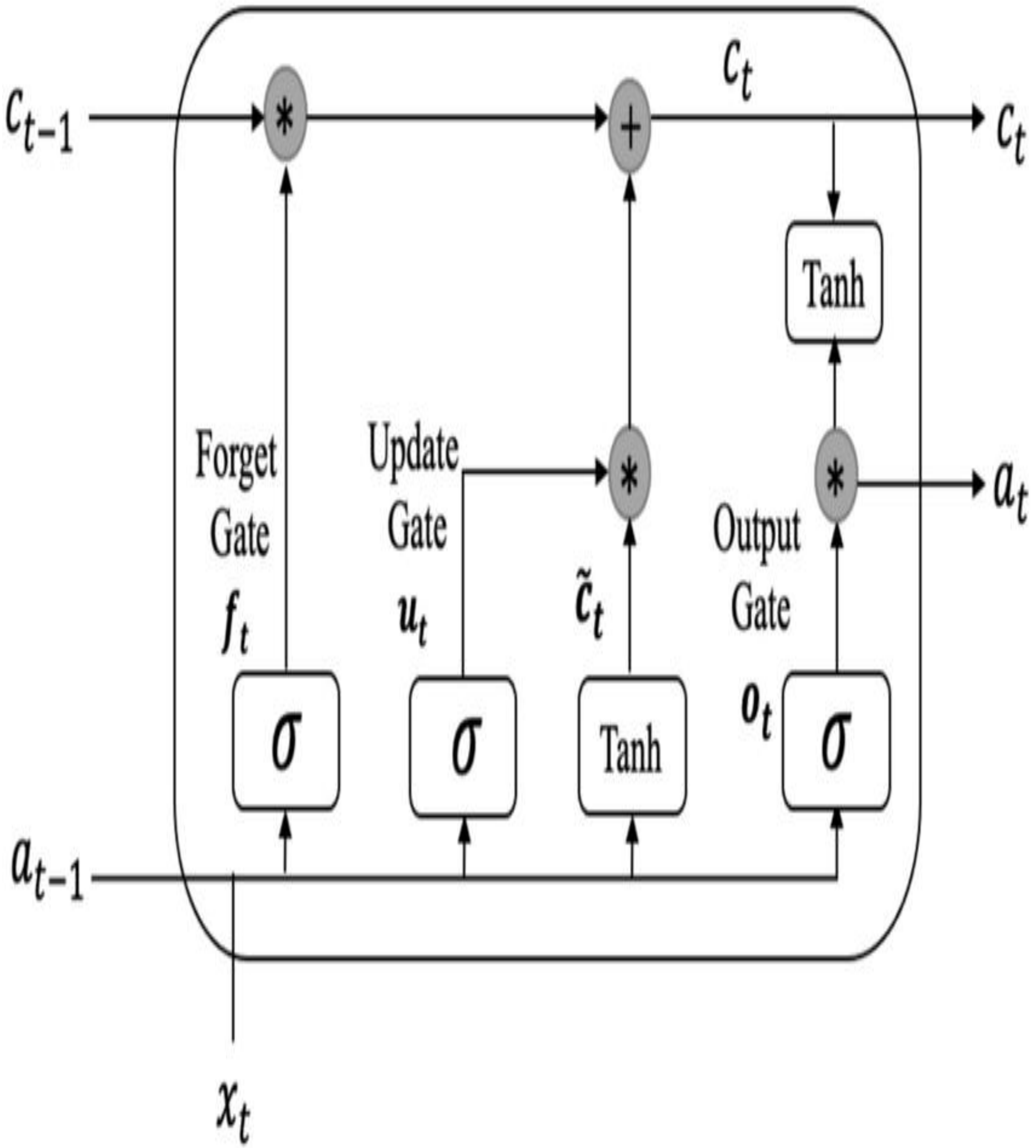
The **cell state** is the internal memory of the LSTM.

It flows through time, carrying important information forward — updated at every step.

Formula:

$$C_t = \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t)$$

Output Gate



After updating the memory (cell state), the LSTM now decides:

What should be output at this time step?

That's the job of the **Output Gate**.

The **Output Gate** controls what information from the updated cell state should become the **hidden state** , which will be:

Passed to the next LSTM unit

Used to produce any final output (e.g., predictions)

Formula:

$$h_t = \tanh(C_t) * o_t$$

Advantages of LSTM

1. **Solves the Vanishing Gradient Problem**
   •Can preserve information over **long sequences**.
   •Unlike vanilla RNNs, LSTM doesn't "forget" quickly.
2. **Captures Long-Term Dependencies**
   •Keeps track of **context** across many time steps.
   •Useful in tasks where earlier input influences much later output (e.g., machine translation).
3. **Selective Memory with Gates**
   •Forget, input, and output gates act as **filters**.
   •The network decides what to **keep, update, or discard**.
4. **Better Accuracy in Sequence Tasks**
   •Proven to outperform vanilla RNNs in **NLP, speech recognition, and time-series forecasting**.
5. **Flexible for Different Data Types**
   •Works with **text, speech, video, stock prices, sensor data**, etc.
6. **Well-Researched & Widely Used**
   •Huge community support, tutorials, and pre-trained models available.

Limitations of LSTM

1. **High Computational Cost**
   - More complex than vanilla RNNs.
   - Each LSTM cell has **multiple gates** → more matrix multiplications per step.
2. **Slower Training**
   - Requires **more resources and longer training time** compared to simpler RNNs or GRUs.
3. **Large Memory Usage**
   - Storing gate activations and gradients consumes **a lot of memory**.
   - Not ideal for very long sequences on limited hardware.
4. **Difficult to Tune**
   - Many hyperparameters: number of layers, hidden units, dropout, learning rate.
   - Sensitive to initialization and optimization choices.
5. **Overkill for Short Sequences**
   - When the sequence length is short, LSTMs may be **unnecessarily heavy**.
   - Simpler models (GRU, even vanilla RNN) may work just as well.
6. **Interpretability**
   - Internal working (gates + cell state) is **hard to interpret** compared to simpler models.

Motivation for GRU

- We’ve seen how **LSTMs solve the vanishing gradient problem** using **gates and memory cells**.
- But… LSTMs are **computationally heavy**:
  - Multiple gates (forget, input, output).
  - Large number of parameters.
  - Slower training and inference.

Researchers asked:

❓ *“Can we design a simpler architecture that works almost as well as LSTM, but is faster and lighter?”*

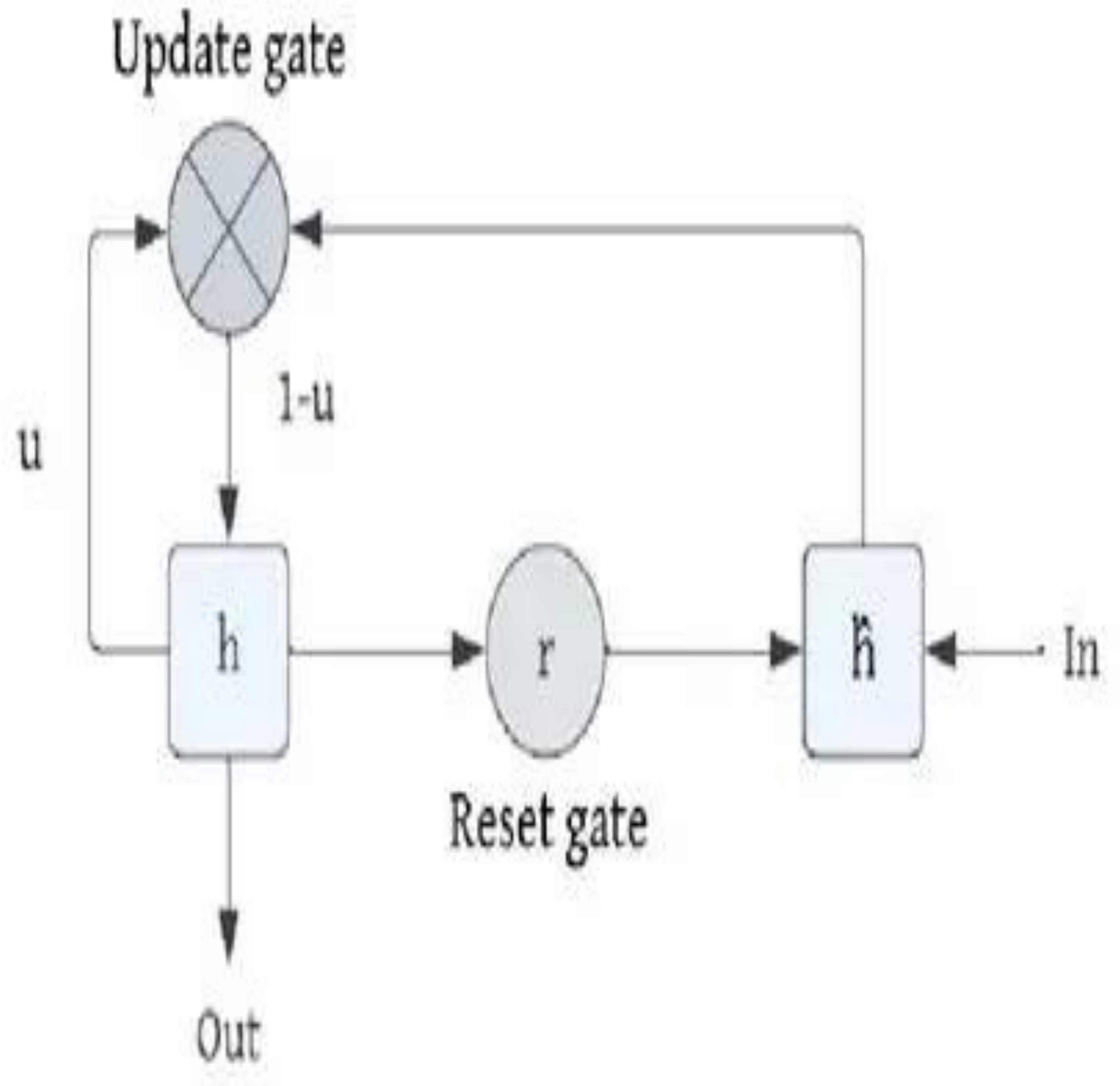
Answer: **GRU (Gated Recurrent Unit)** → introduced in **2014 (Cho et al.)**.

GRU simplifies the LSTM by:

Combining **forget & input gate → update gate**.

Removing the **separate cell state** → only hidden state.

What is GRU?



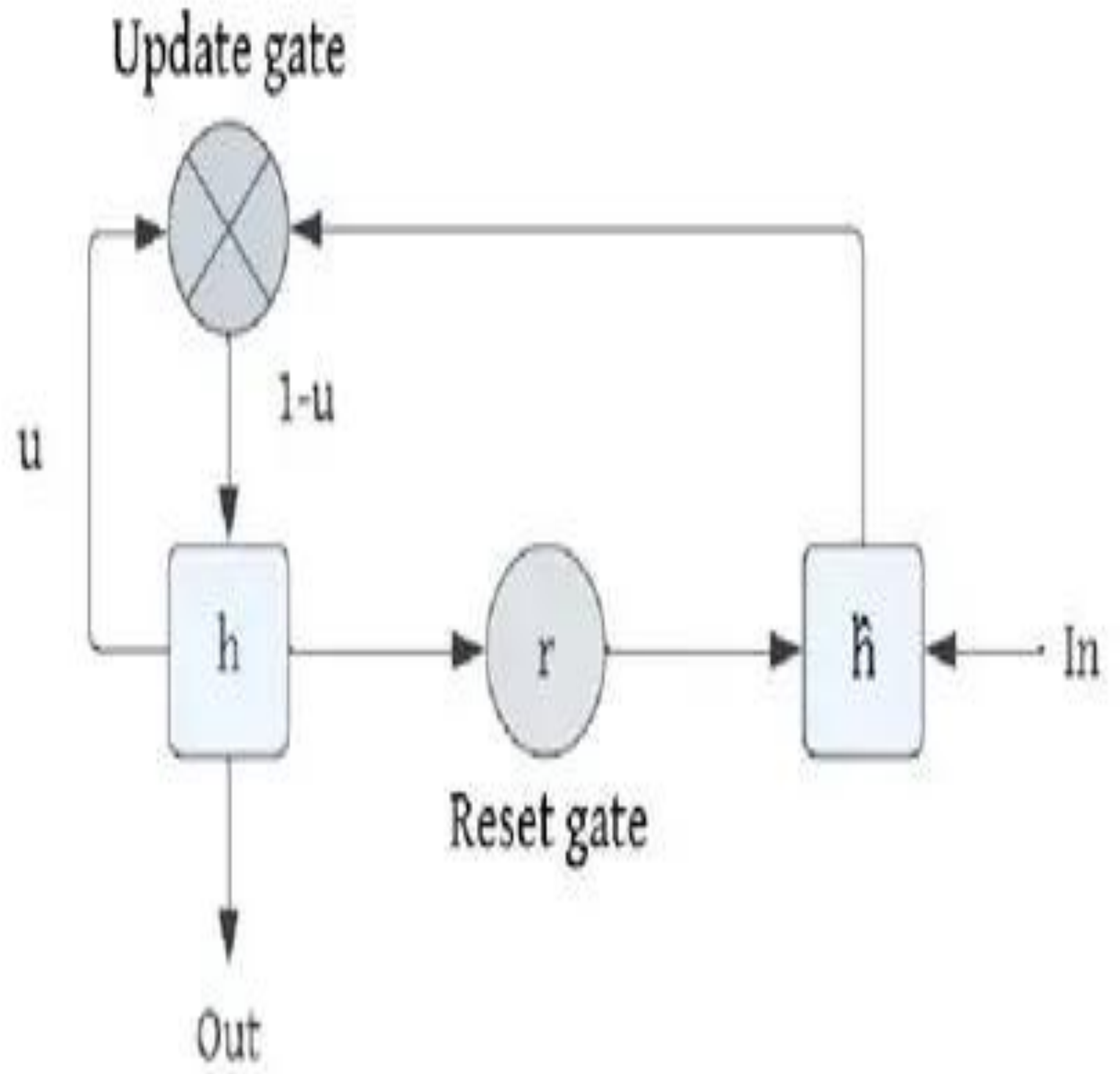
•GRU is a **gated architecture** like LSTM.
•It **controls information flow** with fewer gates.
•Unlike LSTM, it has **no separate cell state** → only maintains a **hidden state**.

GRU was designed to:

- Be **faster and simpler** than LSTM
- Use **fewer gates** (only 2 instead of 3)
- Have **fewer parameters**, which helps it train quicker

GRU Has Two Gates:

- ➢ Update Gate
- ➢ Reset Gate



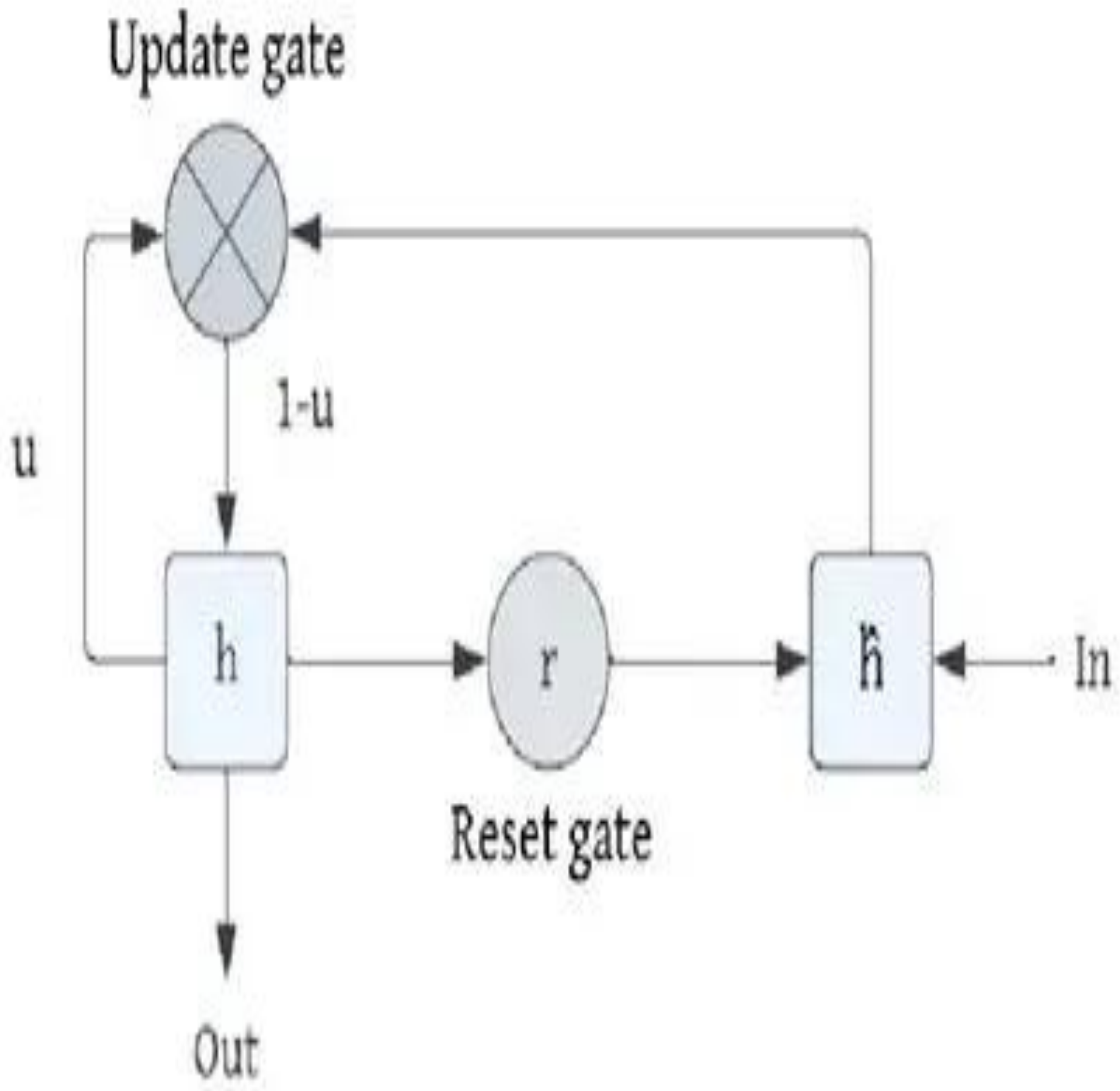
Update Gate

In GRU, the **Update Gate** is responsible for controlling how much of the past information should be **carried forward** and how much of the new information should be **used to update** the current hidden state.

Formula:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

Reset Gate



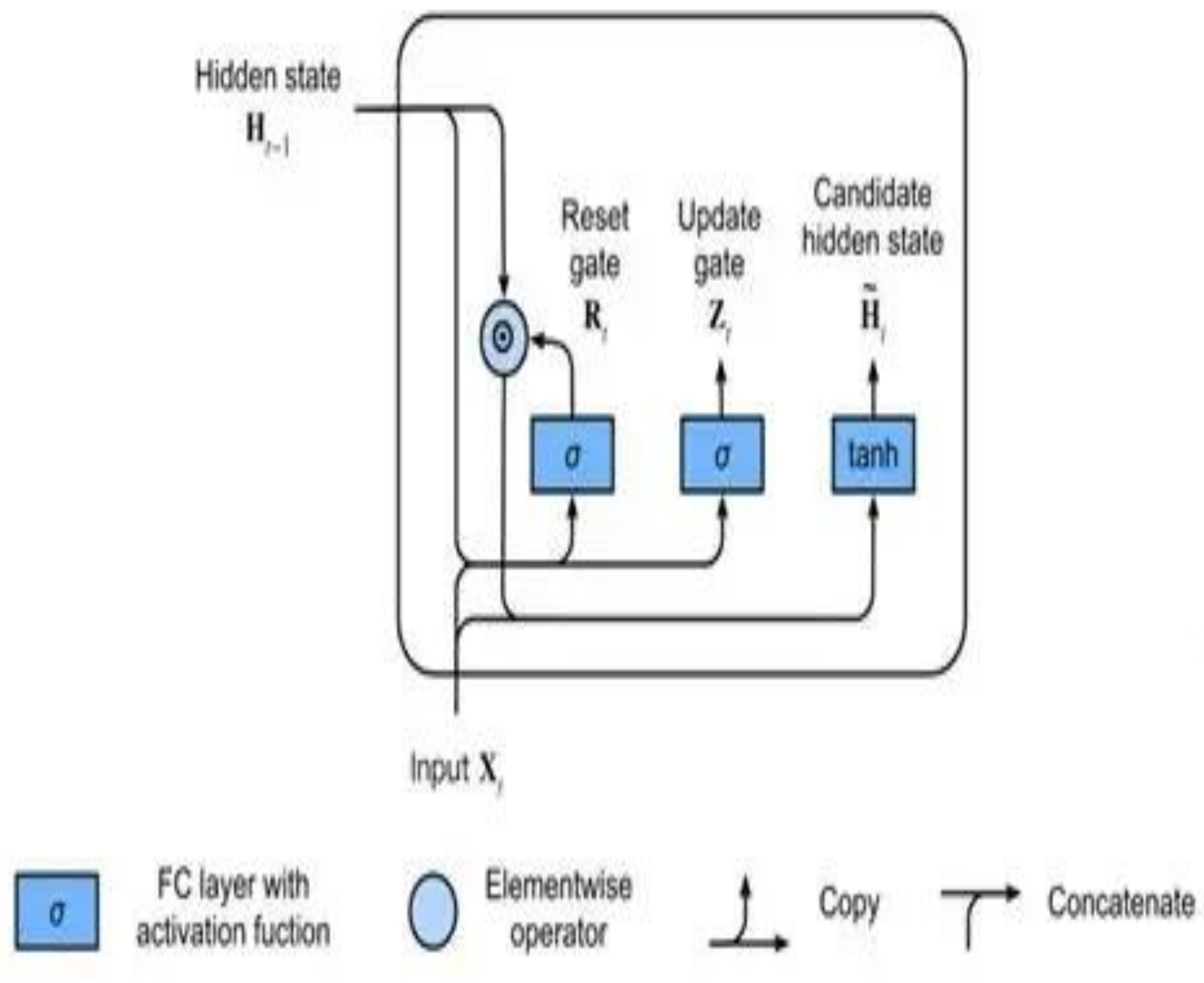
In the GRU, the **Reset Gate** decides how much of the past hidden state should be forgotten **when computing the new memory**.
It helps the model to **reset or ignore irrelevant past information**.

Formula:

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

Candidate hidden state update

$$\tilde{H}_t = \tanh(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h)$$



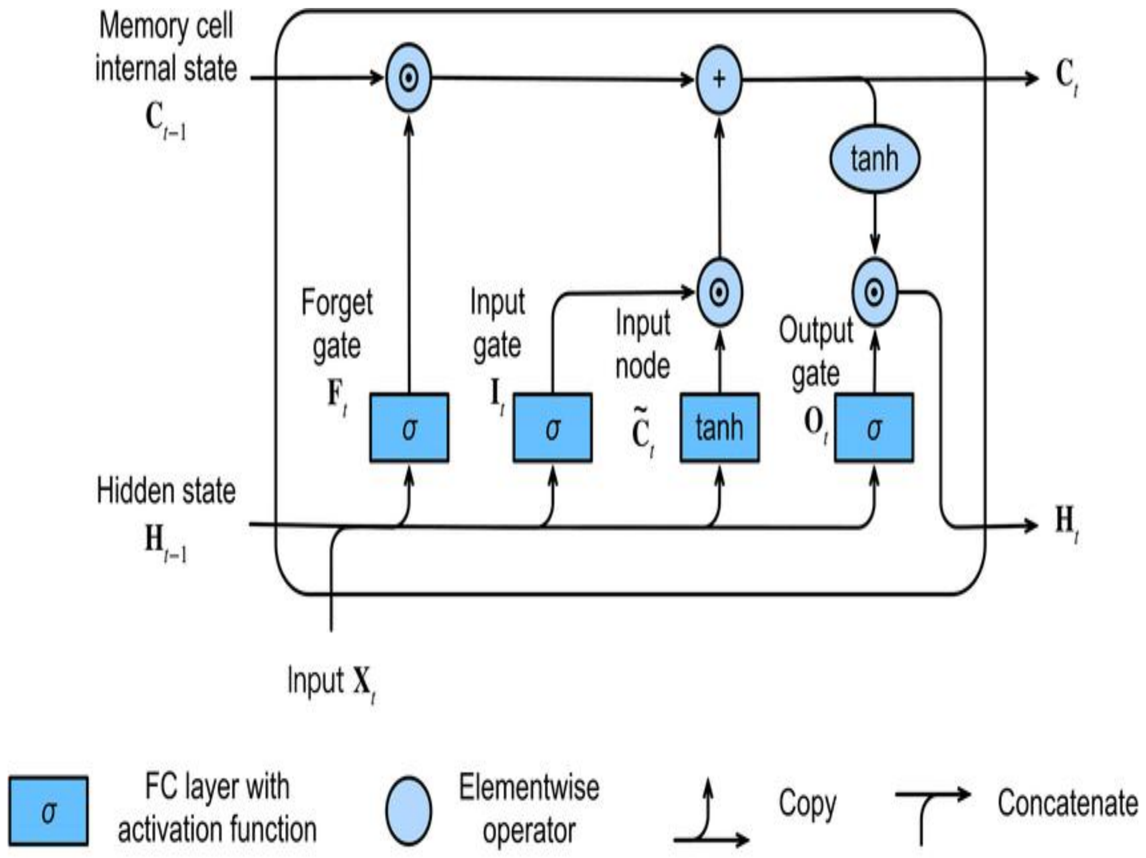
In a GRU, before updating the hidden state, we compute a **candidate hidden state** ($\tilde{h}_t$ (that represents the **new information** the network might want to add.

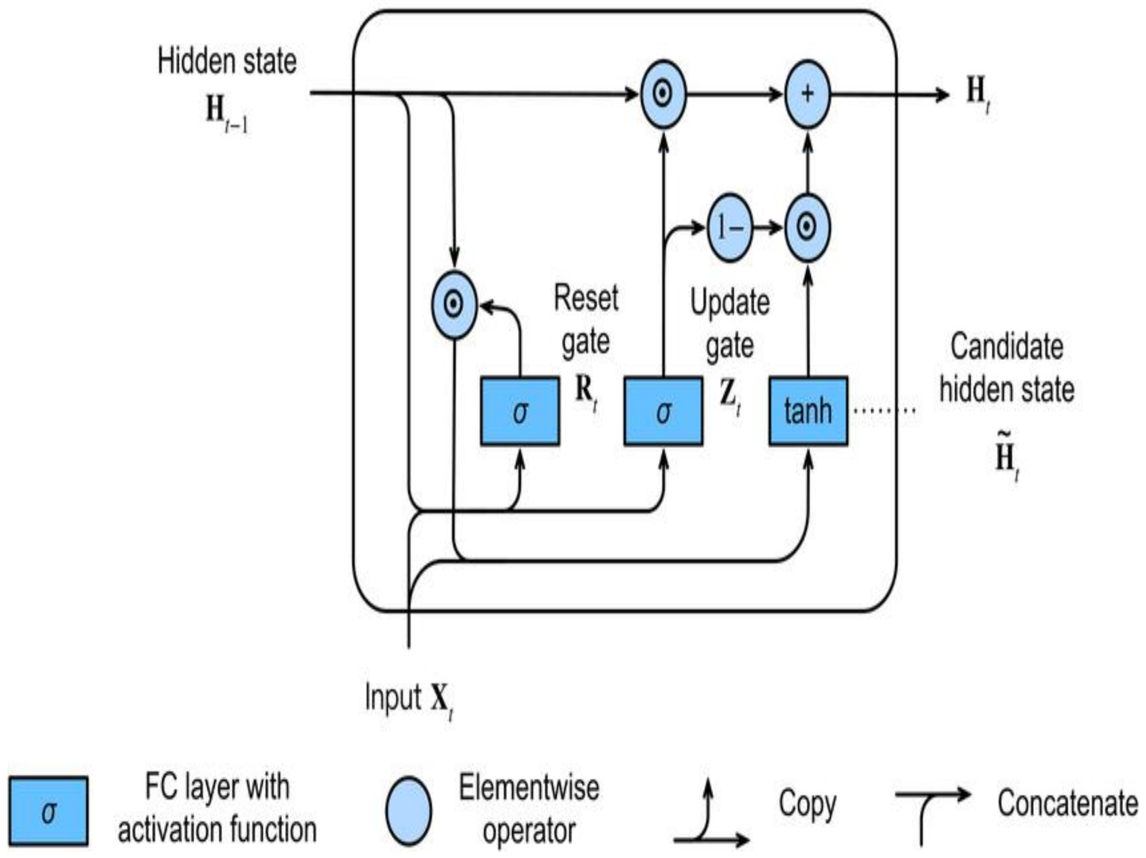
How it works:

- **Reset gate** r_t decides how much of the **past memory** to use.
  - If $r_t \approx 0$ →ignore most of the past (focus on new input).
  - If $r_t \approx 1$ →keep past information.
- The network combines **filtered past info** + **current input**.
- Passes them through a tanh activation → candidate hidden state.

Comparison Table

Feature	LSTM	GRU
Long-term Memory	True	True
Training Speed	Slow	Fast
Complexity	High	Low





When to choose GRU over LSTM

1. **When speed matters**
   •GRUs are **faster to train** (fewer gates → fewer parameters).
   •Better for **real-time applications** like chatbots or online recommendation.
2. **When resources are limited**
   •GRUs use **less memory & computation**.
   •Good for **mobile/embedded devices** or when using smaller GPUs.
3. **When data is small or moderate**
   •GRUs have **lower risk of overfitting** due to simpler architecture.
   •More suitable for tasks with limited training data.
4. **When sequences are not very long**
   •For short/medium sequences, GRUs perform as well as LSTMs, but more efficiently.
5. **When experimenting quickly**
   •Faster training allows **rapid prototyping** and trying different architectures.

Thanks