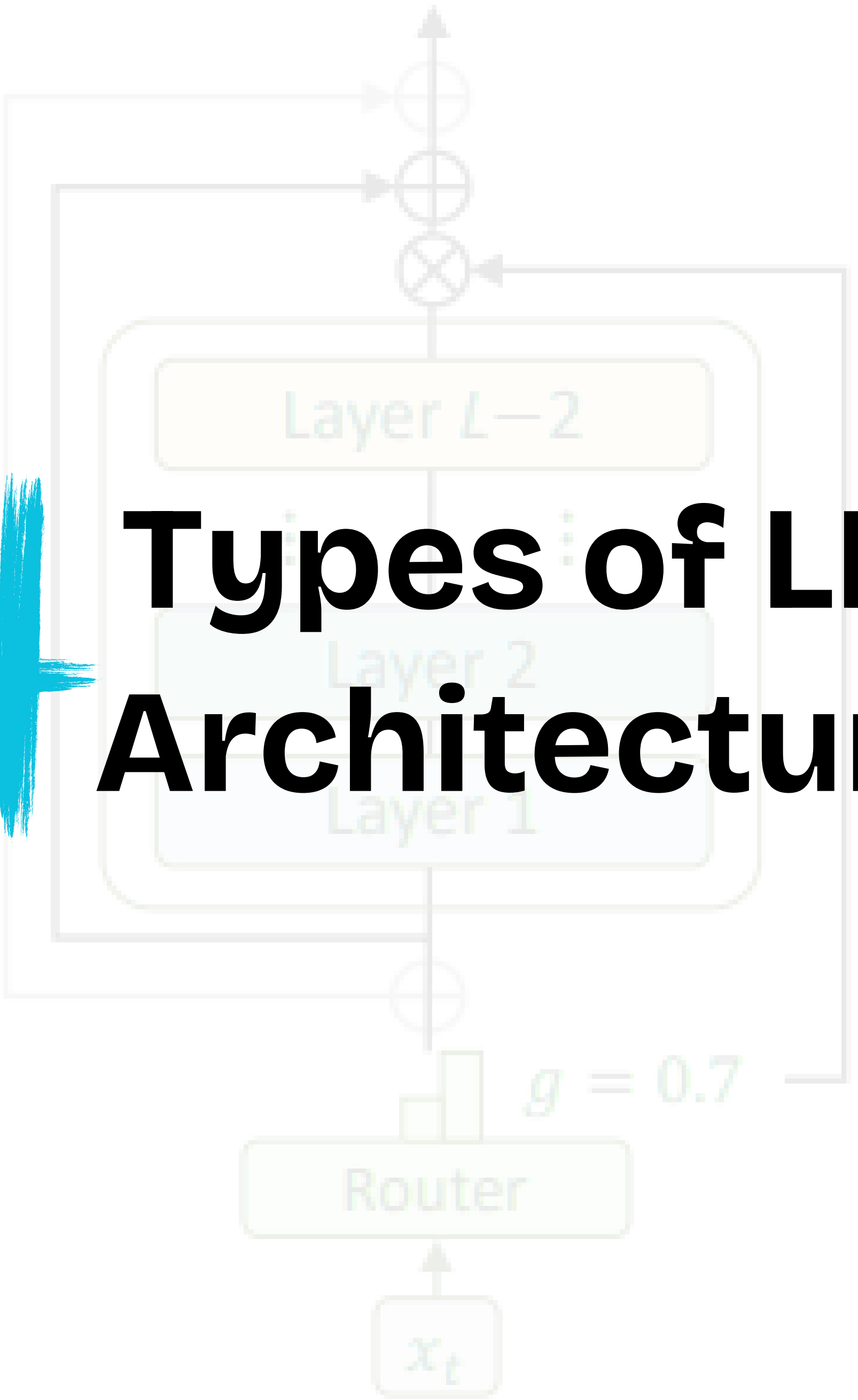
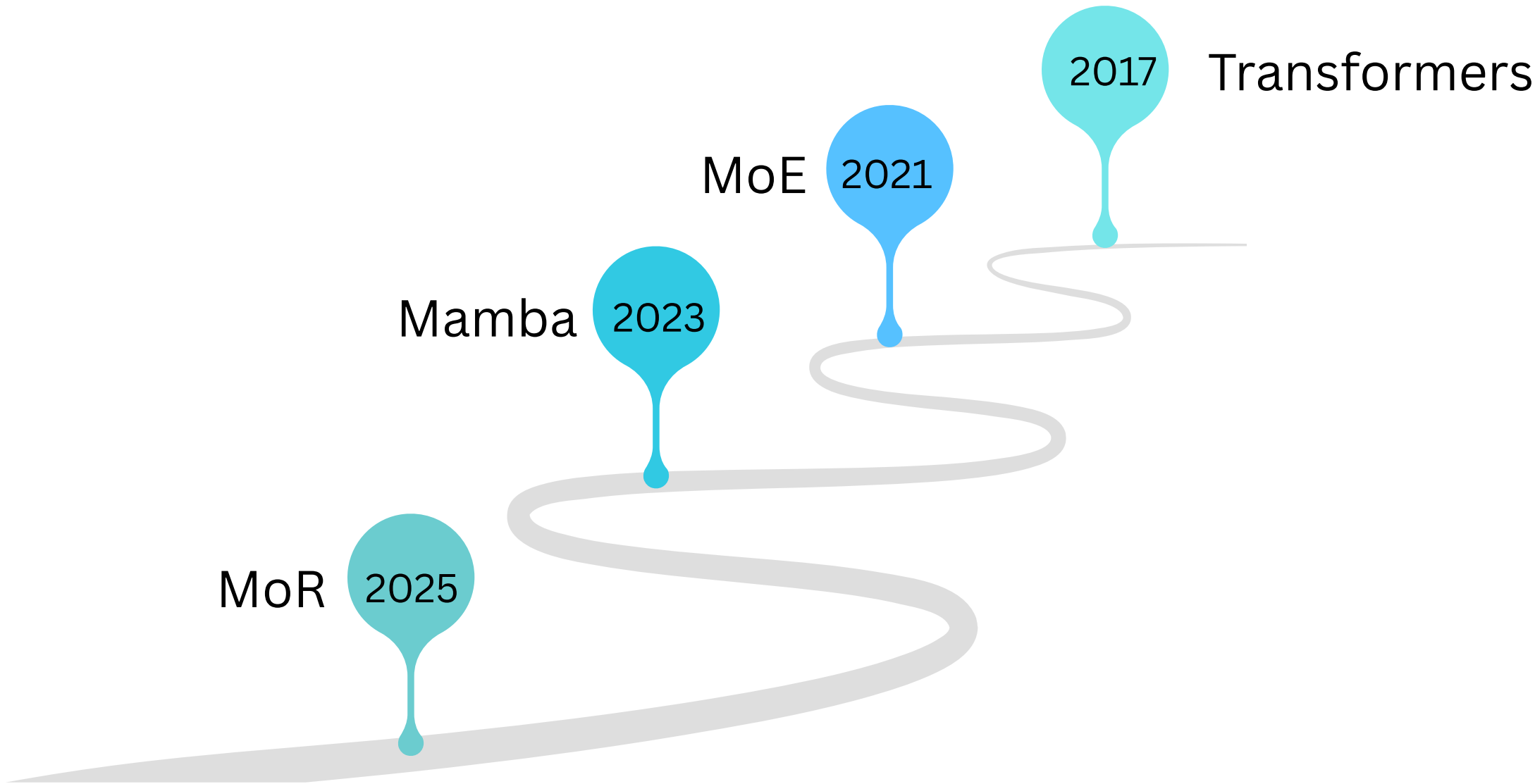


4 Types of LLM Architectures

Introduction

Transformers are everywhere. They're the groundwork for nearly every major language model today.



In this post, we'll walk through the evolution of four key architectures that shaped modern AI models:

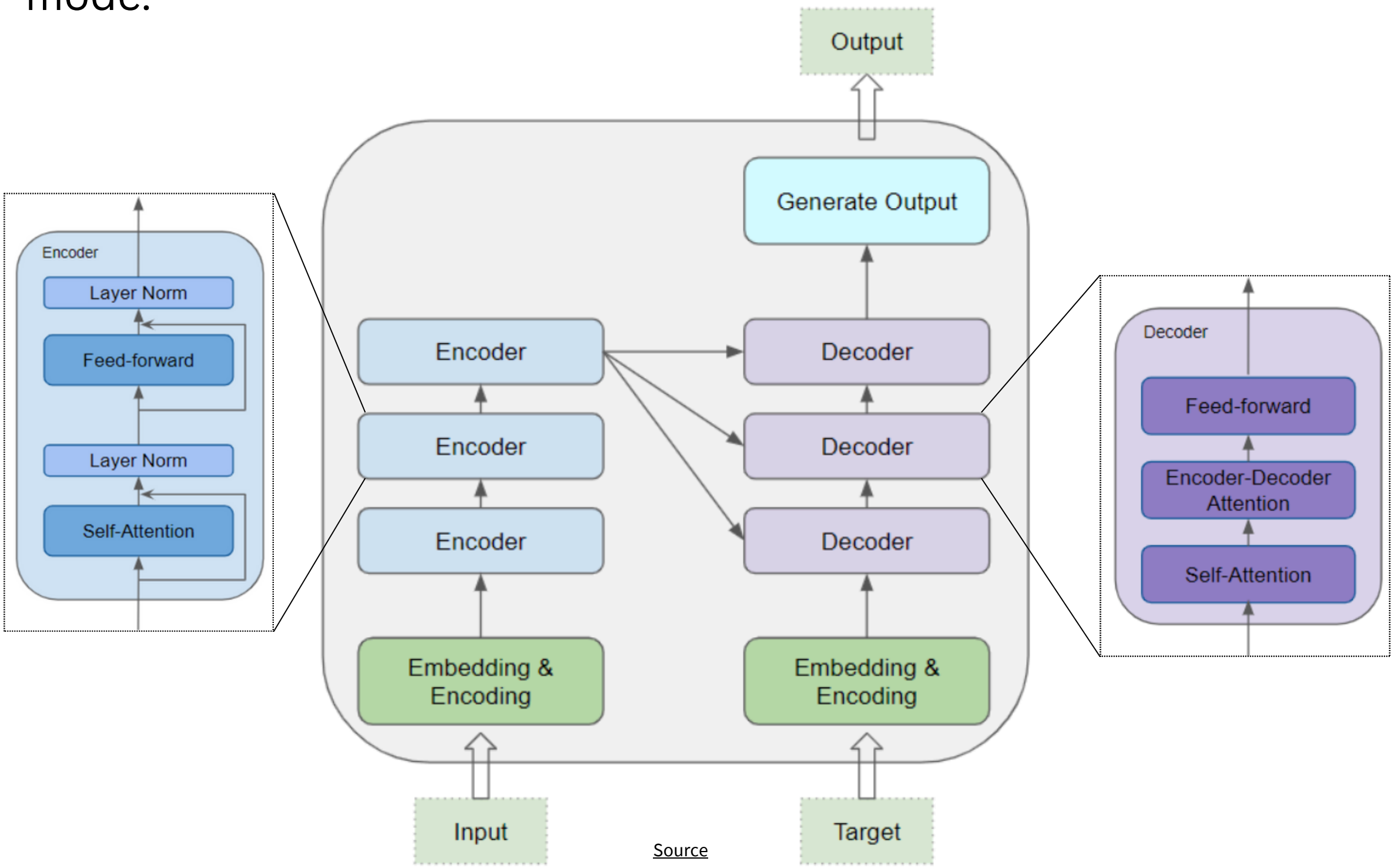
- Transformers
- Mixture of Experts (MoE)
- Mamba
- Mixture of Recursion (MoR)

Each step builds on the last, and together they show where the future of model design is heading. Let's begin with the foundation of it all, **the Transformer**.

1. Transformers

Instead of passing information step by step like RNNs or LSTMs, transfomers introduced **self-attention** and **feed-forward layers** so tokens could look at each other all at once. That introduced te concept of **parallelism**.

Think of Transformers as the moment deep learning hit turbo mode.



Source

This shift unlocked two big things:

- Training on massive datasets without crawling speeds
- Scaling models from millions to billions of parameters

1. Transformers Contd.

How do they work:

- **Self-attention** computes relationships between all tokens in a sequence, allowing each token to weigh the importance of others.
- **Feed-forward layers** transform token representations into higher-level features.
- **Stacked layers** progressively refine these representations, enabling deep contextual understanding.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Why they became so powerful:

- **Parallel training** means they don't have to process words one by one like RNNs did. Huge speed boost.
- They **handle long-range context** really well—perfect for understanding long documents or conversations.
- Most importantly, they **scale** beautifully.

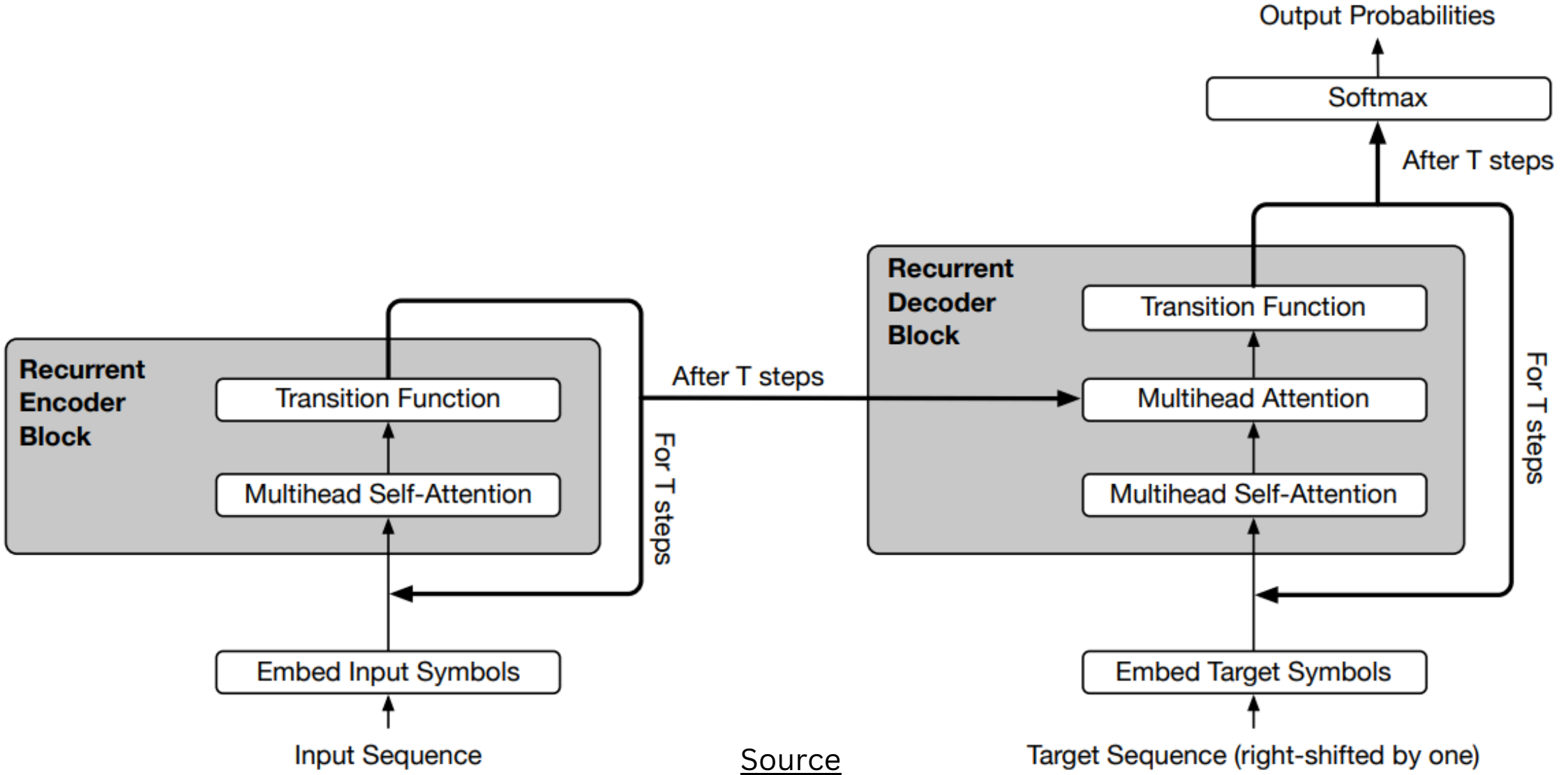
But here's the catch:

- Every token, whether it's a rare scientific term or a common word,goes through the same heavy computation.
- This makes inference slow and costly, since memory and compute pile up.

Universal Transformers

Instead of stacking dozens of separate layers, **Universal Transformers** reuse the same layer again and again. It's like giving tokens multiple passes through one smart filter.

They tied weights across depth and applied the same block repeatedly to refine token states, like an iterative loop. It introduced the idea of per-position iterative processing.

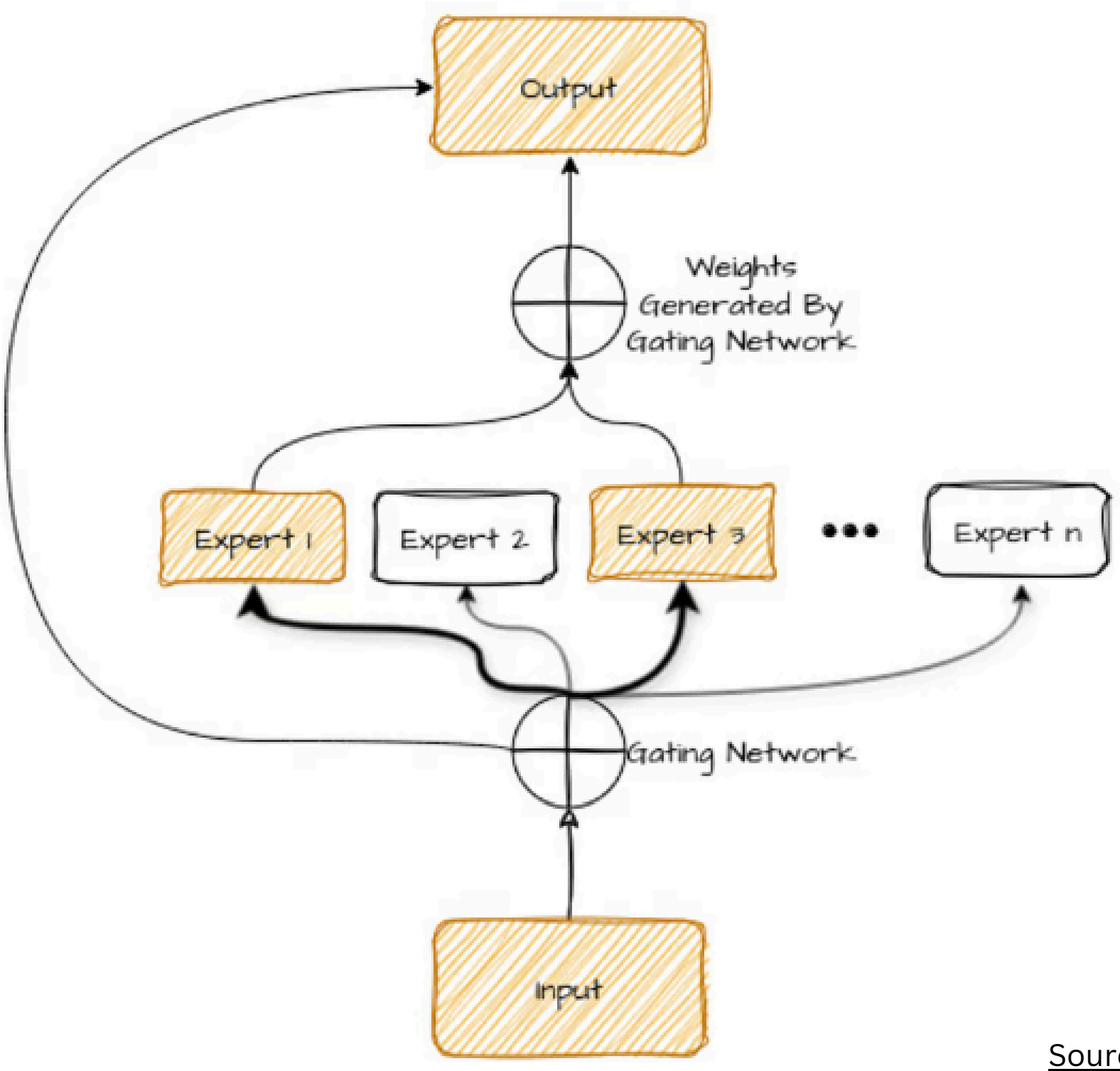


Source

- It also explored adaptive computation time (ACT): let some tokens stop early if they're "done." This gave a proof-of-concept for per-token compute.
- The architecture improves expressivity and efficiency compared to vanilla Transformers.
- It reduces redundancy by sharing parameters across layers.

2. Mixture of Experts

- MoE puts many experts in a layer and routes each token to a few experts. You get huge parameter capacity while keeping per-token compute low when routing is sparse.



Source

Two main components define a MoE:

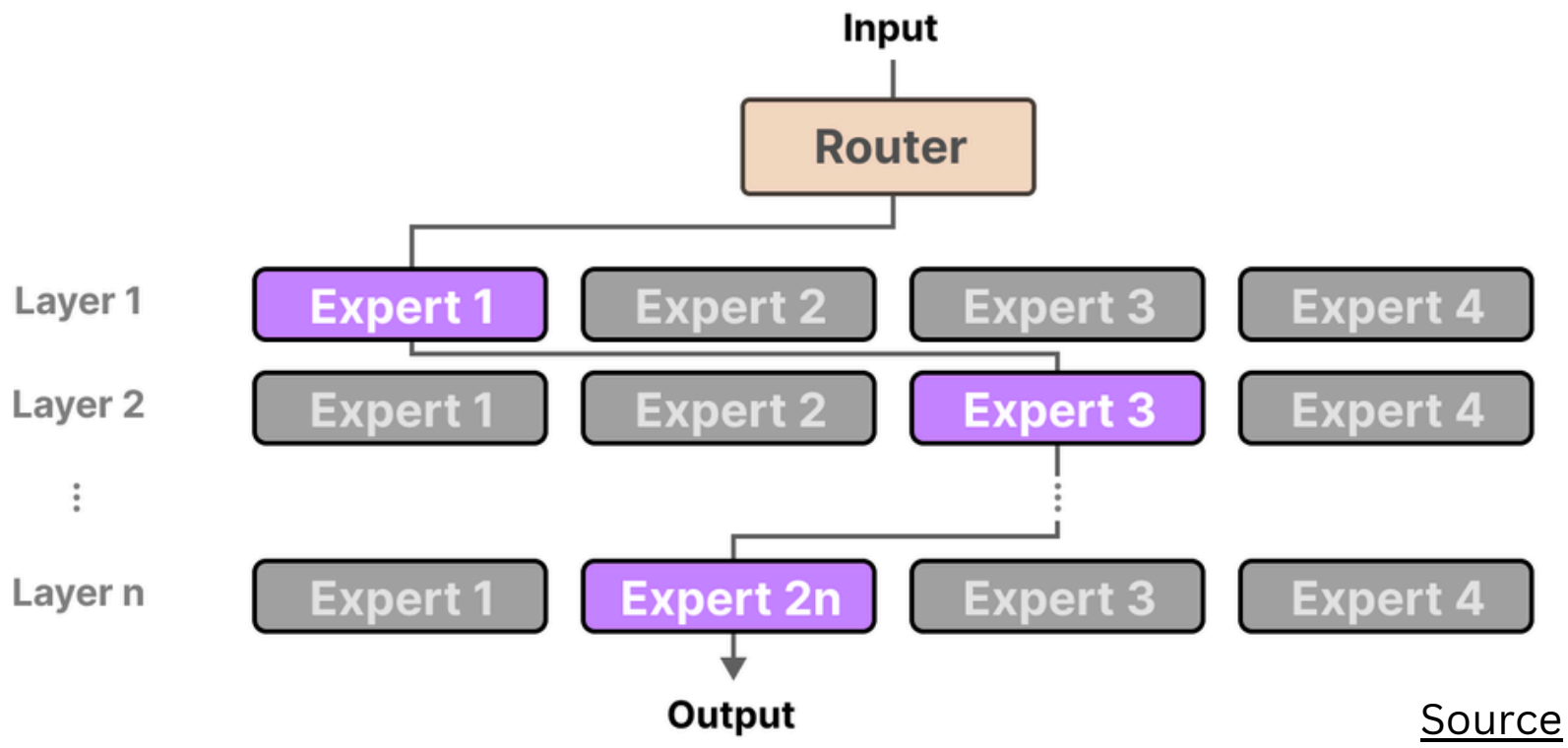
- **Experts** - Each FFNN layer now has a set of “experts” of which a subset can be chosen. These “experts” are typically FFNNs themselves.
- **Router** or **gate network** - Determines which tokens are sent to which experts.

How it works?

Here is how MoE works:

- At runtime, a router decides which 1 or 2 experts a token should go to.
- That means the model can have billions of parameters in total, but each token only activates a tiny fraction of them.
- You get the benefit of huge model capacity without paying the full compute cost on every step.
- This made MoE a big deal in scaling LLMs. It showed that you don’t always need to throw every parameter at every token. Instead, you scale smartly, not just heavily.

This made MoE a big deal in scaling LLMs.

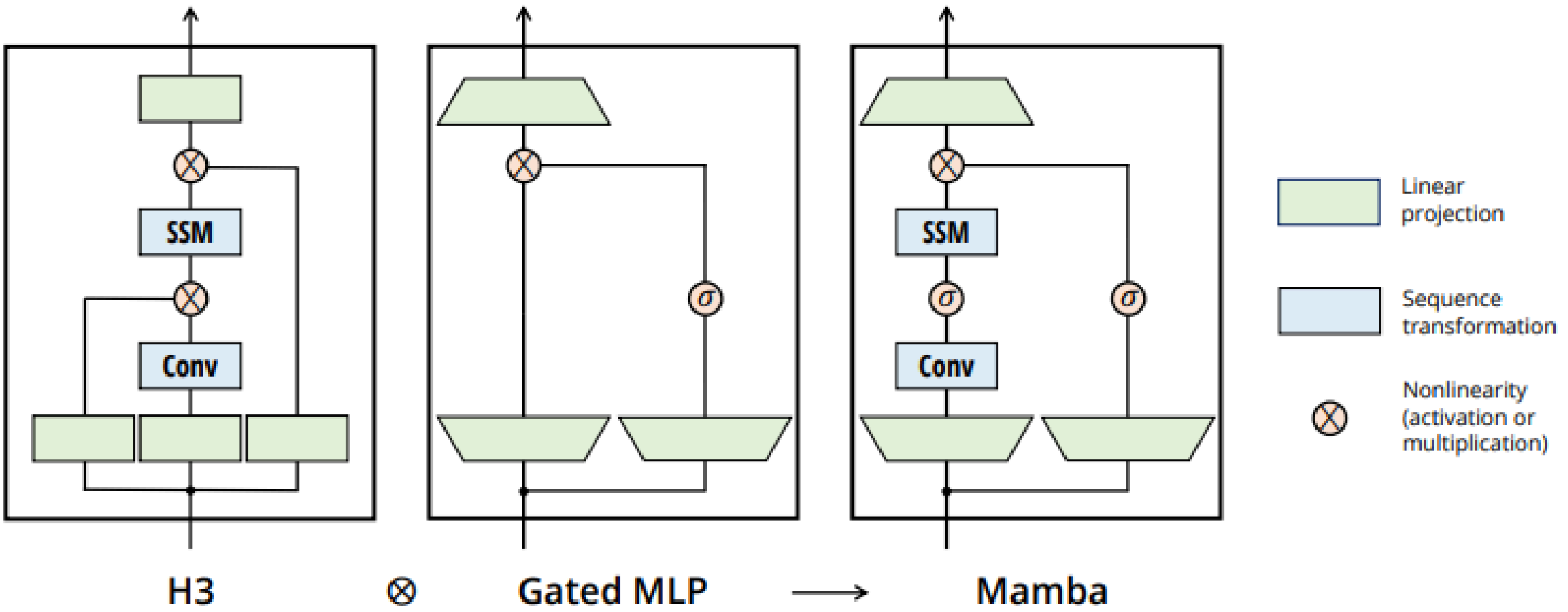


Source

- It showed strong scale efficiency but added routing complexity, load balancing issues, and infra overhead (many experts to manage).
- MoE answered which specialist to use; recursion asks how many times to reuse the same block. Both are conditional compute but with different practical tradeoffs.

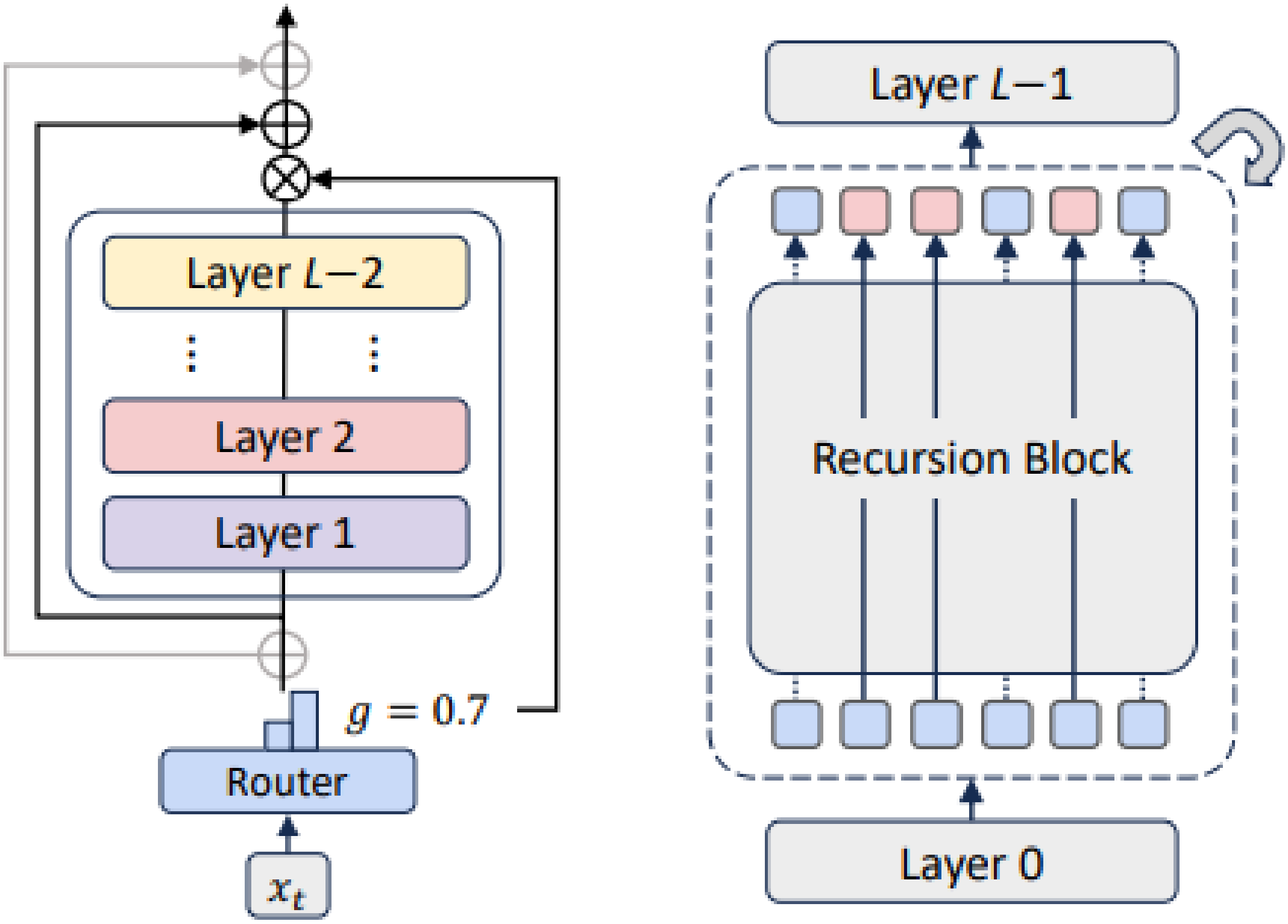
3. Mamba

- Meet Mamba, a **state-space model** (a mathematical framework that tracks and updates an internal state over time to process sequential data efficiently) that's making waves as a Transformer alternative.



- Unlike Transformers, Mamba processes sequences with **linear time complexity**, making it blazing fast for long contexts.
- It **doesn't rely on attention**. Instead, it uses a selective state space mechanism to decide what to keep and what to forget.
- This means **lower memory use**, **faster inference**, and **better scaling** for massive datasets.
- While Mamba shines in efficiency, it's not yet as universally dominant as Transformers in all tasks.

4. Mixture of Recursion



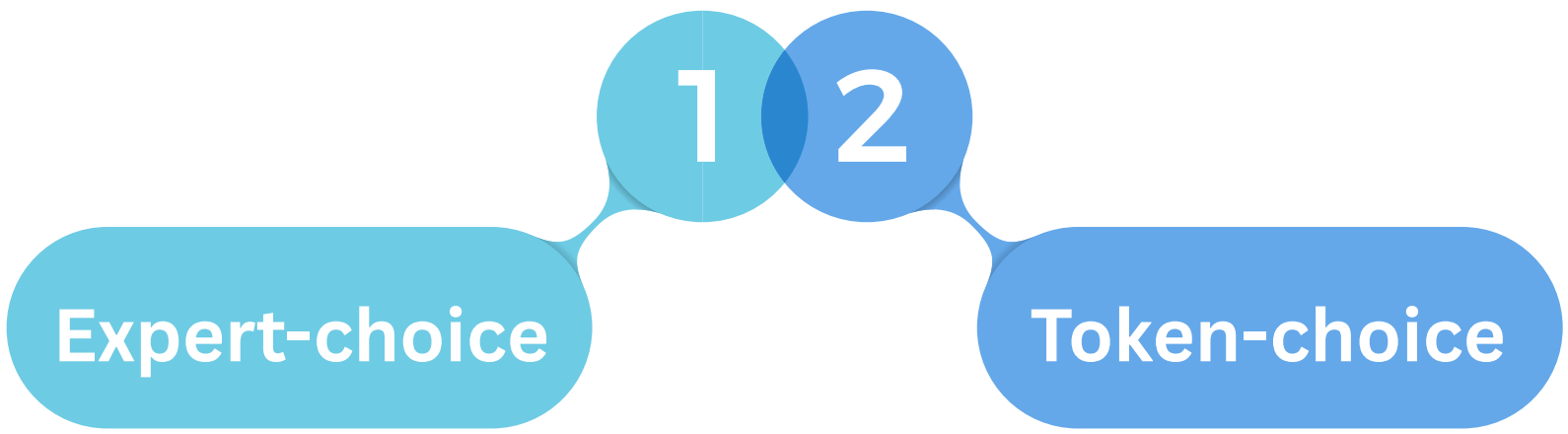
MoR repeatedly applies a shared transformer block while learned **routers decide per token** whether to loop or exit, key-value(KV) caching is **selective for active tokens**.

That yields parameter efficiency (fewer unique weights), compute efficiency (tokens stop early), and lower KV overhead (cache only what's needed).

It's trained end-to-end so routing decisions at inference match what the model learned.

How MoR Works?

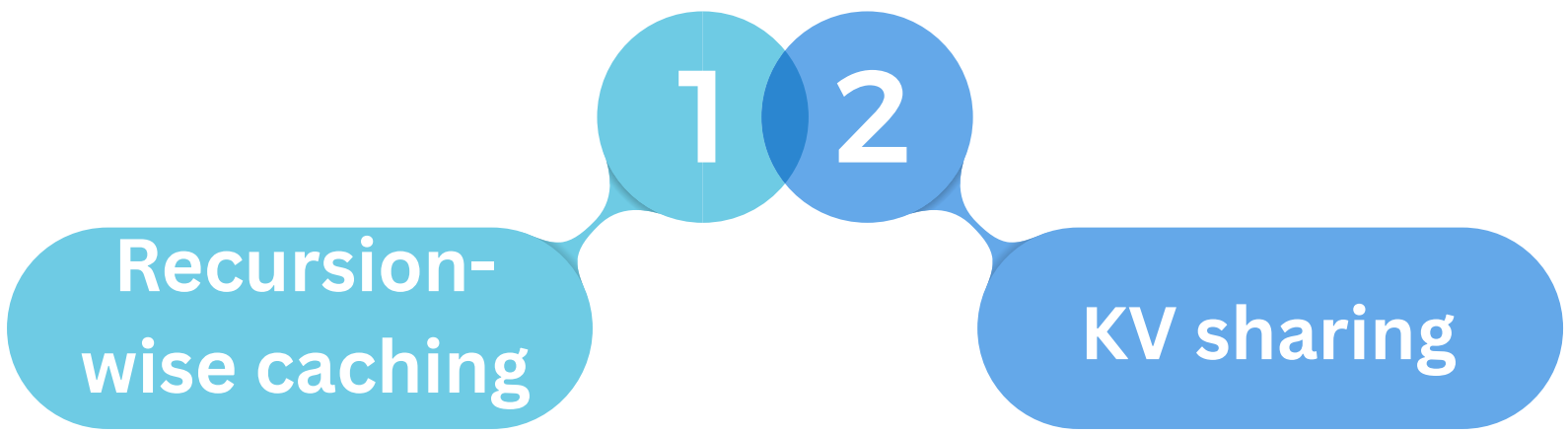
Routing Strategies



At each recursion step, routers pick the top-k tokens that continue. The active set narrows with depth; it's flexible and dynamic.

Predict full recursion depth per token up front and follow that path. It's simpler to batch but needs a reliable depth predictor.

KV Caching Tactics



Store only the keys/values for tokens still in play at each step, cutting down memory reads.

KV sharing → Reuse the first pass's KV pairs in later passes, slashing prefill latency and memory costs.

Routing decides how deep each token goes and caching makes sure you don't blow your memory budget doing it. Together, they're the reason MoR can scale without rewriting entire decoding stacks.

Bhavishya Pandit

Let’s Have a Comparision

Model	Core Idea	Advancement Over Previous	Limitations
Vanilla Transformers	Replace recurrence with self-attention + feed-forward layers, enabling massive parallelism.	Scales training to billions of tokens; foundation of modern LLMs.	Uniform compute for all tokens; large KV caches; high inference cost.
Universal Transformers	Reuse the same block recursively so tokens can adapt their depth of processing.	Adaptive depth vs fixed layers in Transformers.	Sequential per recursion and slower training.
Mixture of Experts (MoE)	Route each token to a small subset of experts instead of using all parameters.	Massive scaling without activating full model each step.	Routing can be unstable; training complexity increases.
Mamba	State-space model replacing attention with efficient recurrence.	Handles very long sequences without huge KV caches.	Still new, ecosystem and tooling less mature compared to Transformers.
Mixture of Recursion (MoR)	Tokens take different recursive depths, exiting early or going deeper as needed.	Saves compute on easy tokens; boosts accuracy at smaller scale.	Requires careful recursion control; hardware support still evolving.

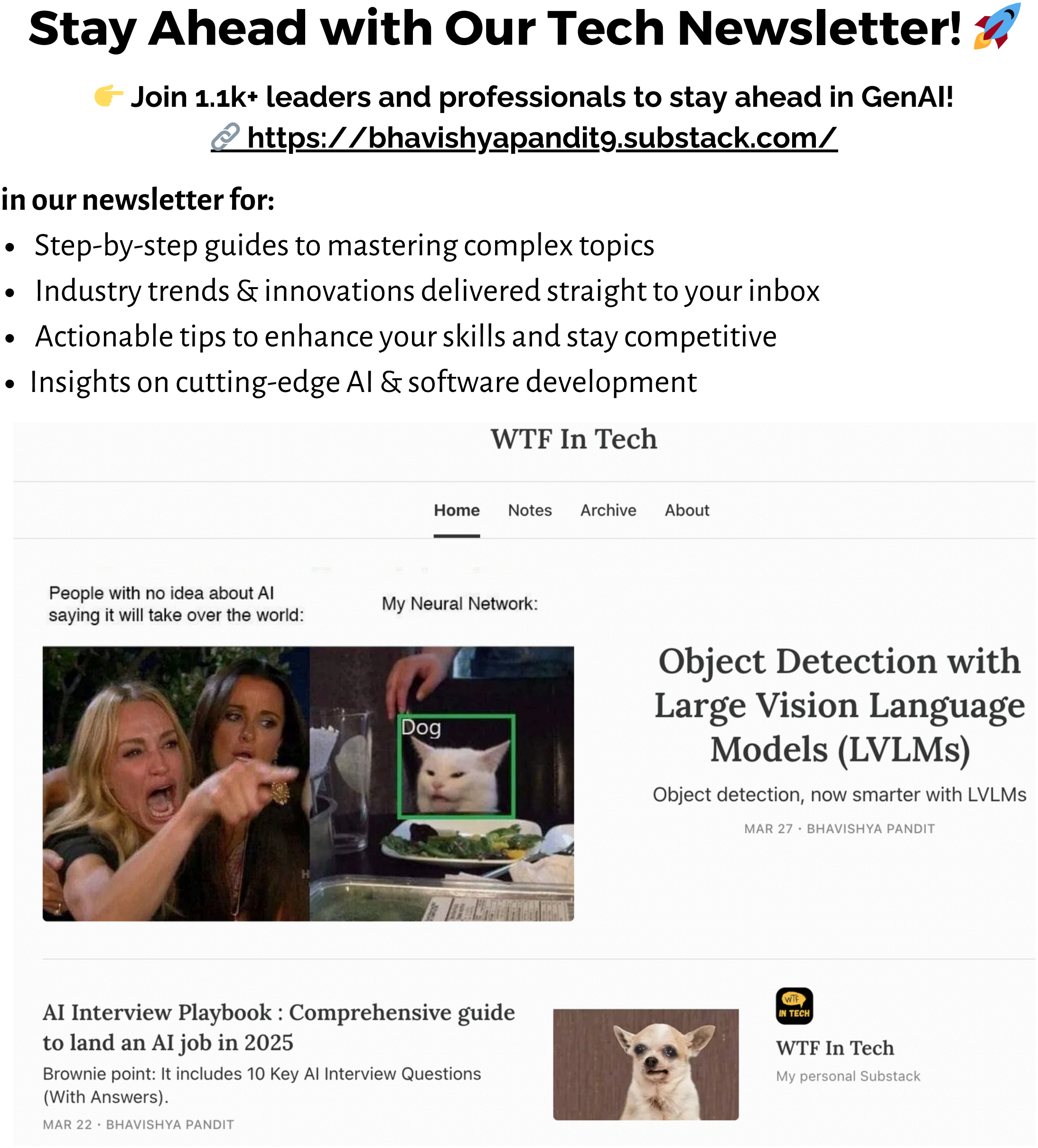
Stay Ahead with Our Tech Newsletter! 🚀

👉 **Join 1.1k+ leaders and professionals to stay ahead in GenAI!**

🔗 **https://bhavishyapandit9.substack.com/**

Join our newsletter for:

- Step-by-step guides to mastering complex topics
- Industry trends & innovations delivered straight to your inbox
- Actionable tips to enhance your skills and stay competitive
- Insights on cutting-edge AI & software development



💡 Whether you're a developer, researcher, or tech enthusiast, this newsletter is your shortcut to staying informed and ahead of the curve.

Bhavishya Pandit

Follow to stay updated on Generative AI

LIKE

COMMENT

REPOST