Artificial Intelligenc... · Follow publication

I Finally Understood “Attention is All You Need” After So Long. Here’s How I Did It.

8 min read · Jul 12, 2025

Olubusolami Sogunle Follow

Listen Share

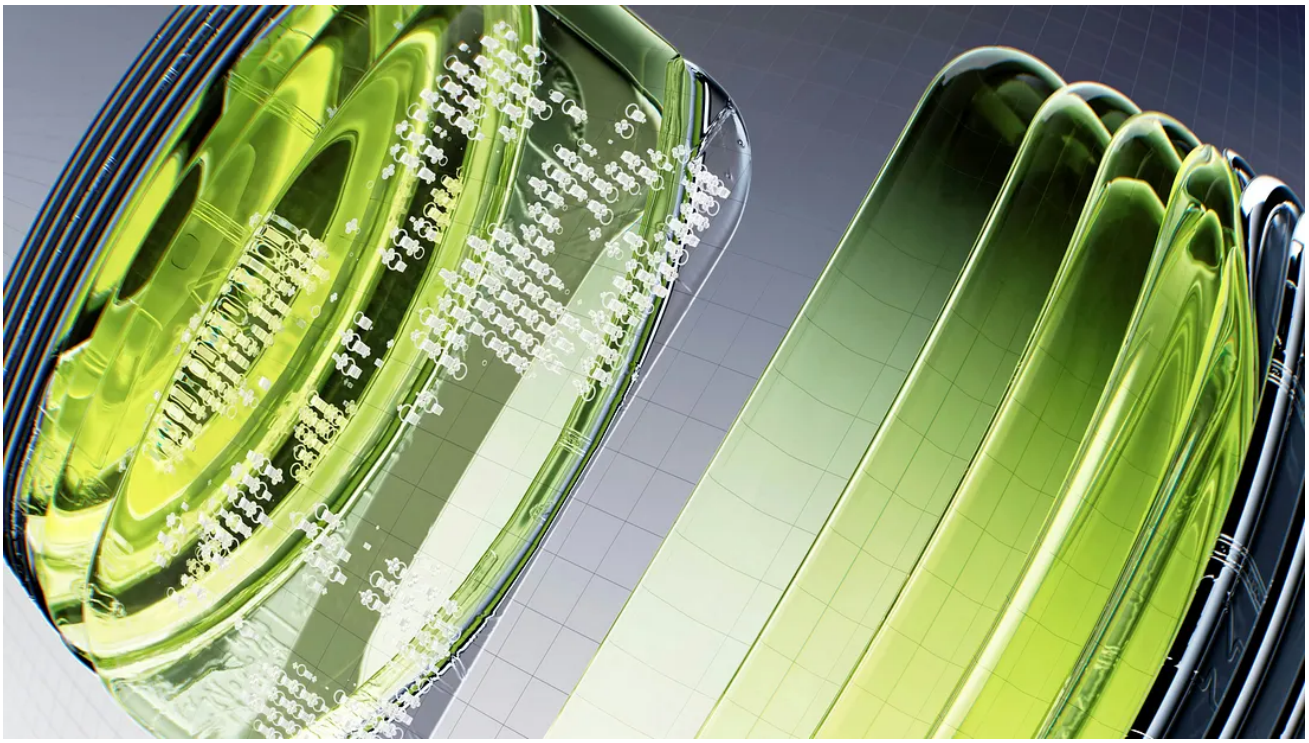

Photo by Google DeepMind on Unsplash

It’s been almost 2 years since I first encountered the “Attention is all you need” paper by Vaswani et al. (2017). I’ve mentioned it confidently in a few

conversations, but I only just came to understand it *enough* this month.

> *We listen, we don’t judge.*

In all fairness to me, I only started taking my NLP journey seriously late last year, and I’ve had to wade through information overload, *hype* and a lot of foundational stuff just to get to a point where papers like this finally make proper sense. I even wrote about Word2Vec in my last article; you can check it out.

Before this latest attempt, I had tried several times to read the paper, but it never fully clicked. This time around, I used the 3-pass technique I saw on Madukoma Blessed’s blog, and I finally got the core message and technical parts.

If you’ve also been putting off reading “scary papers,” maybe this method might help you. Walk with me.

The strategy

The 3-pass method is pretty straightforward.

- First, you skim and note the overall idea;
- Then, you dig a bit deeper for key ideas;
- Finally, you go all in on the technical details.

It helps to ease into tough papers without burning out. And that’s exactly what I did. As you read further, I’ll detail my learnings in bullet points under each pass.

Some information...

To help you understand the notes in the rest of the article, you need to be familiar with a few things, like RNNs, CNNs and Sequence transduction models (like Seq2Seq). You don’t have to cram the math. Just get the general idea.

- **RNNs (Recurrent Neural Networks):**

They process data *step by step*, passing information from one step to the next. Think of them like reading a sentence one word at a time, remembering what came before. They're great at handling sequences, but they can be slow and struggle with long sequences.

- **CNNs (Convolutional Neural Networks):**
  Originally built for images, but sometimes used for text too. They focus on learning *local patterns. F*or example, detecting small chunks or phrases in text. They're faster than RNNs in some cases, but still have limits with longer dependencies.

- **Sequence transduction models (like Seq2Seq models):**
  These models take in a sequence and produce another sequence. Classic examples? Translation and summarisation. You give it a sentence, and it outputs a new sentence in another language or a shorter version.

If these still feel confusing, it might be helpful to pause and brush up a little more on NLP basics first. And that's perfectly okay. Been there, done that. 2 years *hello*?!

> *Maybe still a little there to be honest.*

Pass 1: Skimming and "Wait, that's why?!"

- The whole point of the paper is this: What if we stopped using complicated stuff like *RNNs* and *CNNs* to build sequence transduction models and just used *attention* instead?

- To prove this idea, they introduced a new architecture called the *Transformer*, which was built solely with the attention mechanism, without any recurrence or convolutions.

- It worked and proved to be more efficient than the State-of-The-Art (SOTA) models at the time, using *BLEU scores* as the evaluation metric.

- Hence, attention is all you *really* need. How did I not get that since?!

> *PS. Weirdly enough, I finally learnt the meaning of* ***SOTA****, and I'm feeling like a cool "kid". Time to not-so-casually throw it around in conversations.*

Pass 2: Key Ideas & Takeaways

The Old Way (Before This Paper)

- Sequence models (like for translation) were usually built using *encoders* and *decoders* made up of RNNs (like LSTMs or GRUs), sometimes in combination with CNNs or the *attention* mechanism.
- While they worked, they had a major bottleneck: They processed data step by step, meaning one word at a time.
- This was not-so-great for modern hardware (GPUs, TPUs) designed for parallel processing. Longer sequences meant more memory, more time, and endless waiting.
- Even with *tricks*, you were stuck in a step-by-step loop.

The New Way (The Transformer)

This improved architecture meant:

- Full parallelisation during training: You can process everything at once, instead of waiting for previous steps.
- Faster training and better performance.

Their Results?

- The transformer beat older models on English-to-German and English-to-French translation tasks.
- They used multiple attention heads (8 in the base model), which helped capture different patterns in the data.
- Dropout helped prevent overfitting.
- They also tested a version of the model on an English parsing task, showing it could generalise beyond translation.

Side Note

Although training is now fully parallel, generating text (output) still happens one word at a time (sequentially). They mentioned this as a future challenge. The

authors also acknowledged that it's text-based only (no multimodal *yet,* or at least, *as at the time*).

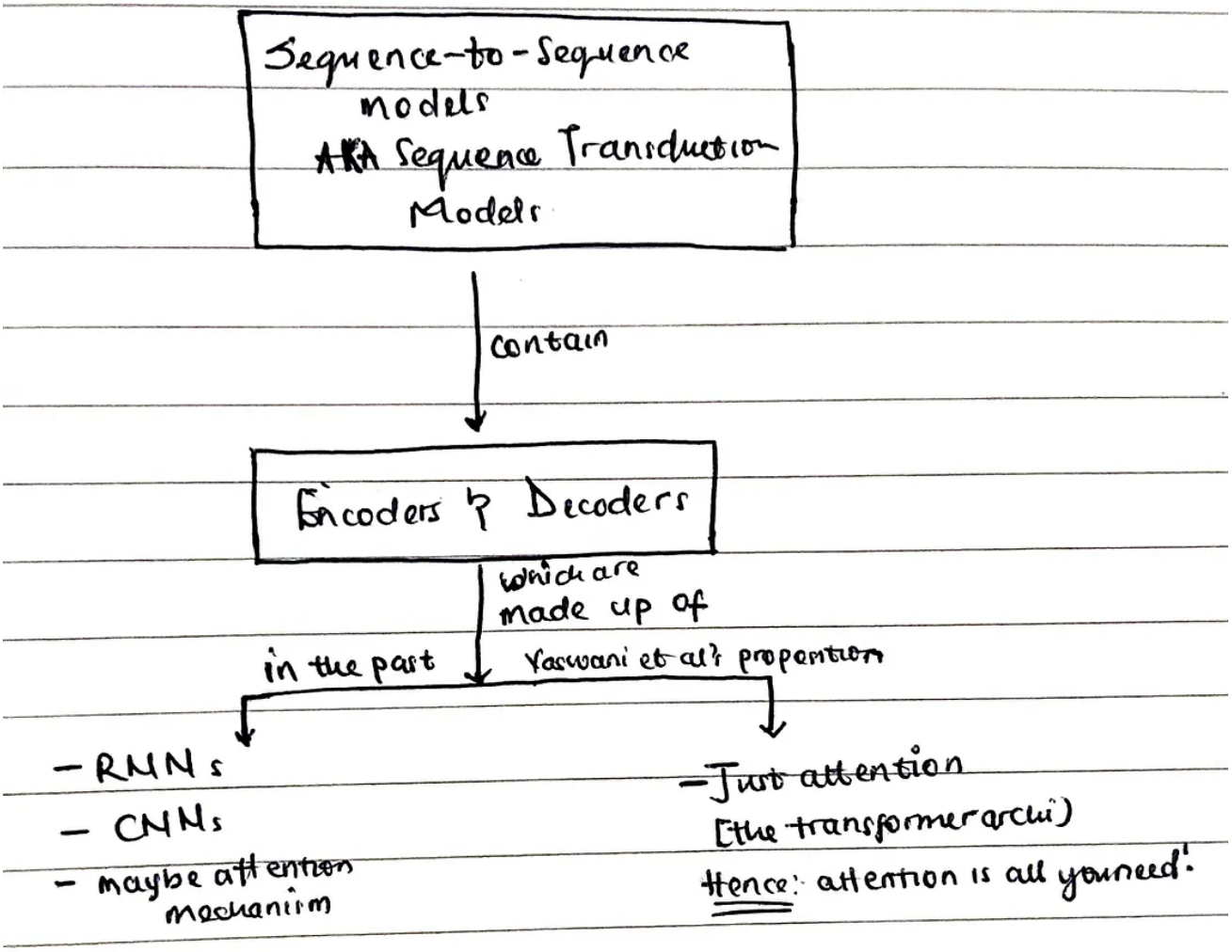

Figure 1: This was my understanding after pass 2

Pass 3: Breaking Down the Model Architecture

I focused on Sections 3 and 4 of the paper, which describe the components of the architecture in detail. The Transformer, like older seq2seq models, has an **encoder** and a **decoder.** Both are made of layers, and those layers have sublayers. A formula for calculating the output of a sublayer of either the encoder or decoder is:

```
Output = LayerNorm(x + Sublayer(x))
```

where:

- ***x*** is the input to the sublayer.
- ***Sublayer(x)*** is the actual computation (e.g., attention, feed-forward network).
- ***x* + *Sublayer(x)*** is called a **residual connection**, a fancy term for adding the input back to the output of the sublayer. It's crucial for stable training and optimisation in deep networks.
- ***LayerNorm()*** is Layer Normalisation, which is applied for stability after adding the input back.

Now, let’s look at the core components of the Transformer

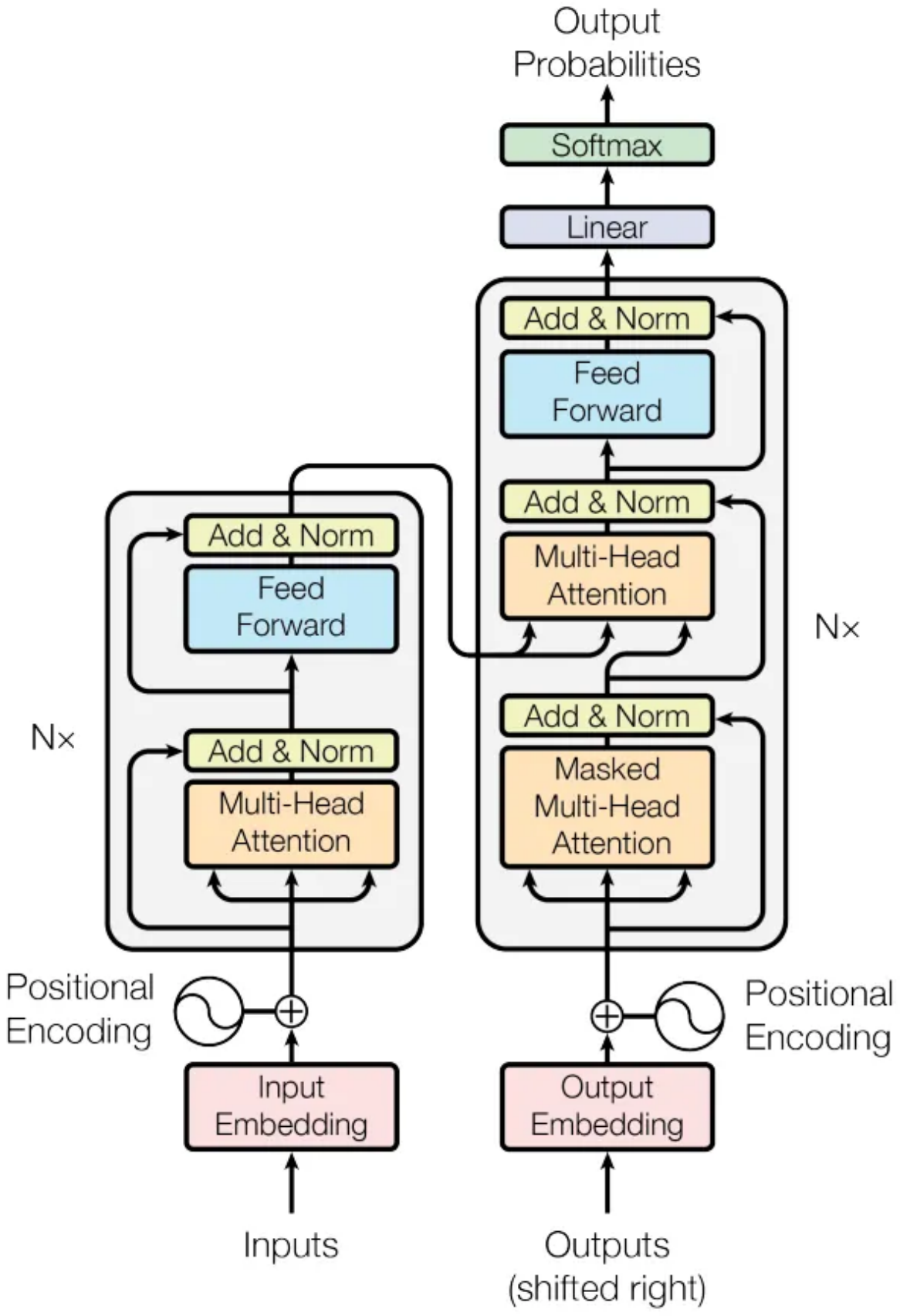

Figure 2: The Transformer architecture with its components — Vaswani et al. (2017). One encoder layer on the left, one decoder layer on the right. The Nx represents the fact that there are multiple (N=6) layers.

1. Attention (Star of the show)

Attention is essentially a way to map a query and a set of key-value pairs to an output. It helps the model “focus” or “pay attention” to a specific part of a sequence. The paper focuses on two key types:

Type	What It Does	Example
Self-Attention	Looks at words in the same sentence.	“bank” in “She went to the bank” looks at nearby words.
Cross-Attention	Decoder looks at encoder output.	Translating “Bonjour” → looks at encoded “Bonjour” to generate “Hello”.

Figure 3: Screenshot from my Notion notes

Why the emphasis on self-attention in the paper?

1. Faster and more parallelisable.
2. Learns long-distance relationships better.
3. Easier to visualise and interpret.

In the paper, they also discussed two techniques to calculate attention:

- **Scaled Dot-Product Attention:**
  This is the core mathematical operation. It takes three matrices: **Q** (queries), **K** (keys), and **V** (values). Each row of the matrices represents a token in the sequence. The formula is as follows:

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

- **Multi-head Attention:**
  Instead of just one *attention* calculation, it runs multiple scaled dot-product attentions in parallel, called *heads*. It doesn’t just split the input; it creates *different learned projections* (like different “views” or “perspectives”) of Q, K,

and V for each “head.”

Each head focuses on different patterns in the input. Finally, the results from all these parallel heads are combined. This gives the model a much richer understanding of the relationships within the data.

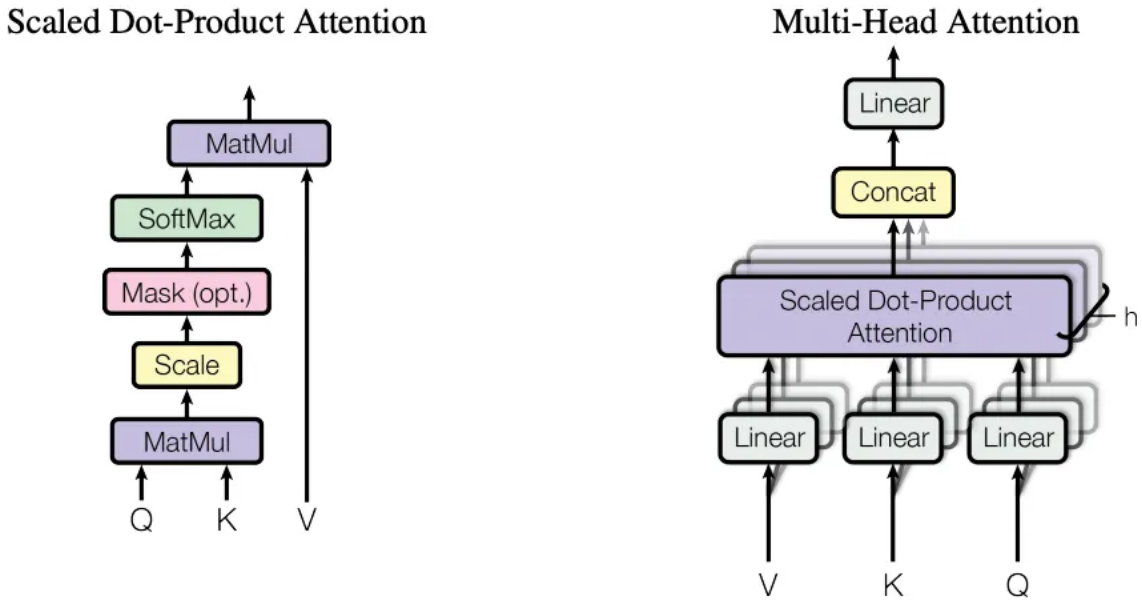

Figure 4: Scaled Dot Product and Multi-head Attention — Vaswani et al. (2017). The “h” in multi-head attention represents the number of heads running in parallel.

2. The Encoder

This has 6 layers, and each layer is made up of 2 sublayers (see figure 2):

- **Multi-head self-attention:** This means each word looks at all the other words in the sentence, including itself, to figure out which ones matter most. This helps the model understand the full context of the input.
- **Feed-forward network (FFN):** A mini neural network applied to each word separately.

The formula `Output = LayerNorm(x + Sublayer(x))` applies to both of these sublayers with the “Sublayer(x)” function either being the multihead self-attention or the FFN.

3. The Decoder

The decoder also has **6 layers**, but with 3 sublayers per layer (also see figure 2):

1. **Masked multi-head self-attention:** Just like the encoder’s, but with a small twist: *masking*. Each word can only look at itself and the ones before it, not the ones ahead. This way, the model doesn’t cheat by seeing future words when generating a sentence.
2. **Cross-attention:** This allows the decoder to look at the *output of the encoder*. This is how the input sequence (e.g., the English sentence) informs the generation of the output sequence (e.g., the German translation).
3. **Feed-forward neural network:** Again, it is an independent network applied to each position in a sequence.

The mathematical representation for one layer of the decoder would be:

```
z1 = LayerNorm(x + MultiHeadMaskedSelfAttention(x))
z2 = LayerNorm(z1 + MultiHeadCrossAttention(z1, EncoderOutput))
z3 = LayerNorm(z2 + FeedForward(z2))
```

With z3 being the output of the layer. As you can see, it’s still built on the general formula.

Other Components (Still Important!)

1. **Feed-forward networks:** Small neural networks that refine the outputs after attention.
2. **Embeddings:** Turn input words into vectors so the model can work with them. The same embedding weights are reused at the end to help predict the output words.
3. **Positional Encoding:** Adds position information to the words/tokens because transformers *process everything in parallel*, and this makes them not take note of order (unlike RNNs, which naturally know/encode order).

Wrapping Up

That’s essentially the summary of the paper. It’s a dense paper, but breaking it down using the *3-pass method* made it much more manageable and, dare I say, enjoyable.

This article is by no means *exhaustive,* as you can still get into the finer details of the paper, but I hope it helps you do that a little easier.

If you’ve been putting off reading this paper or a similar one like I did, here’s your sign to try. Use the *3-pass method,* and feel free to let me know how it goes.

Until next time!

References

1. Attention is all you need https://arxiv.org/abs/1706.03762
2. Research That Sticks: The Strategic Framework for Turning Technical and Academic Papers into Lasting Knowledge https://mblessed.hashnode.dev/research-that-sticks-the-strategic-framework-for-studying-technical-papers

Thank you for being a part of the community

Before you go:

- Be sure to **clap** and **follow** the writer 👏
- Follow us: **X** | **LinkedIn** | **YouTube** | **Newsletter** | **Podcast** | **Twitch**
- **Start your own free AI-powered blog on Differ** 🚀

- Join our content creators community on Discord 🚀
- For more content, visit plainenglish.io + stackademic.com

Transformers NLP Beginner Research Artificial Intelligence

Follow

Published in Artificial Intelligence in Plain English

28K followers · Last published 10 hours ago

New AI, ML and Data Science articles every day. Follow to join our 3.5M+ monthly readers.

Follow

Written by Olubusolami Sogunle

273 followers · 85 following

Early-career human-centred technologist writing about software engineering, AI, Research and other things that interest me. Welcome!

Responses (11)

Write a response

What are your thoughts?