u3a Computing Group

Alan Hopwood, 3 September 2026

Agenda

Welcome
Current News, Issues and Questions
Topic Plan
Topic: AI Chatbot Operation
Discussion

Current News, Issues and Questions

Anything to discuss?

Topic Plan

Topic	Votes
Getting more from your smart phone, e.g. attaching keyboard and monitor broadening its use.	5
Managing and Saving Photos	4
Apple/MS Windows/Chromebook/Linux comparison	4
Home Security Systems and Devices	3
use of AI	3
How AI works - Operation	3
How AI works - Training	
Exploring AI further: comparison to human intelligence, biases, risks, AI Industry	
Internet Connectivity and Networks	2
NAS (Network Attached Storage) Drives	2
Basics of Quantum Computing (inc. Bloch Sphere)	2
Email Clients and Servers (cover compatibility)	1
Raspberry Pi	0
What hardware have you made redundant and what did it do?	0

Topic

How AI Works:
LLM Chatbots Explained

Understanding Claude, ChatGPT, and Similar Systems

What We Will Cover

Today

How an Operational Chatbot Works

From the moment you type a message to the reply you see — every processing step inside the system.

Next meeting

How the AI System Is Built

Training from raw text to a useful, helpful, and safe chatbot — the full pipeline.

How an Operational Chatbot Works

The Full Path: Input to Output

Input Processing: What Goes In?

Text
Chat messages, prompts, questions

Images
Photos, diagrams, screenshots

Documents
PDFs, spreadsheets, slides

Chat History
Previous messages in conversation

Content extraction and parsing: The system identifies the user's intent, any attached files, conversation history, and system instructions. It strips noise and builds a clean request object for the next stages.

Tokenisation: Translating Text into “AI Vocabulary”

Example: "Hello, how are you?"

Gets Split and translated Into Tokens:

"Hello"	"com"	"how"	"are"	"you"

Why Tokenisation?

Tokens are the smallest units AI works with. Total vocabulary is about 100,000 tokens, with each token having a unique number. Common words are single tokens. Complex words will be split.

Token Counting

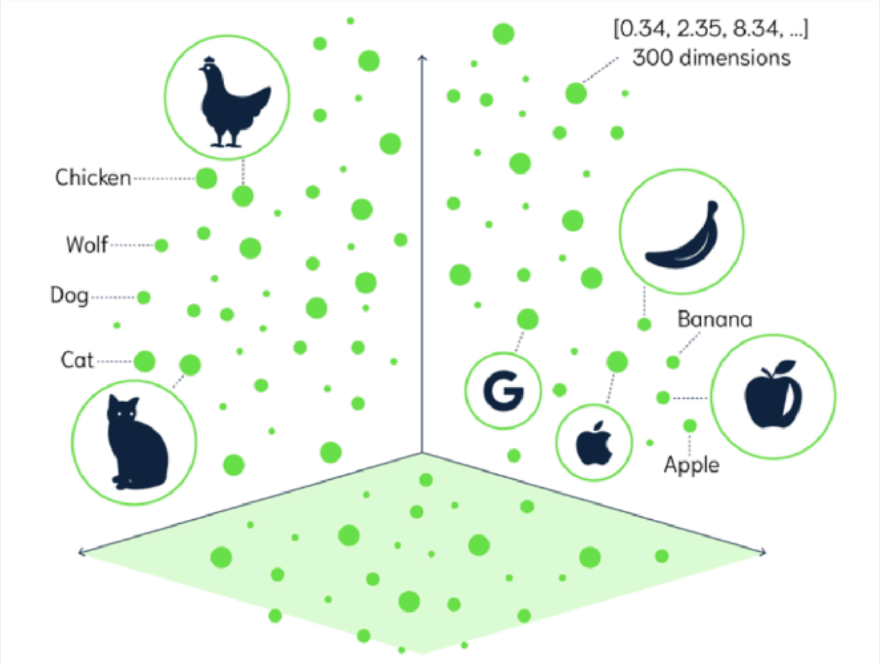
Each AI has a limit on tokens it can process at once (context window), e.g. 128k tokens.

Multimodal tokens — Images and other media are also turned into sequences of tokens so the same transformer can process everything together.

Embedding

Each token is allocated a high-dimensional vector (a list of thousands of numbers) from a look-up table. Similar meanings end up close together in this “space”.

Simplified 3-D meaning space



Why it matters

The model never sees the word "cat". It only sees a vector that “cat” has been translated to. The cat vector exists near other animal vectors. All later reasoning happens in this continuous number space.

Embeddings: From Numbers to Meaning

Each token becomes a high-dimensional vector:

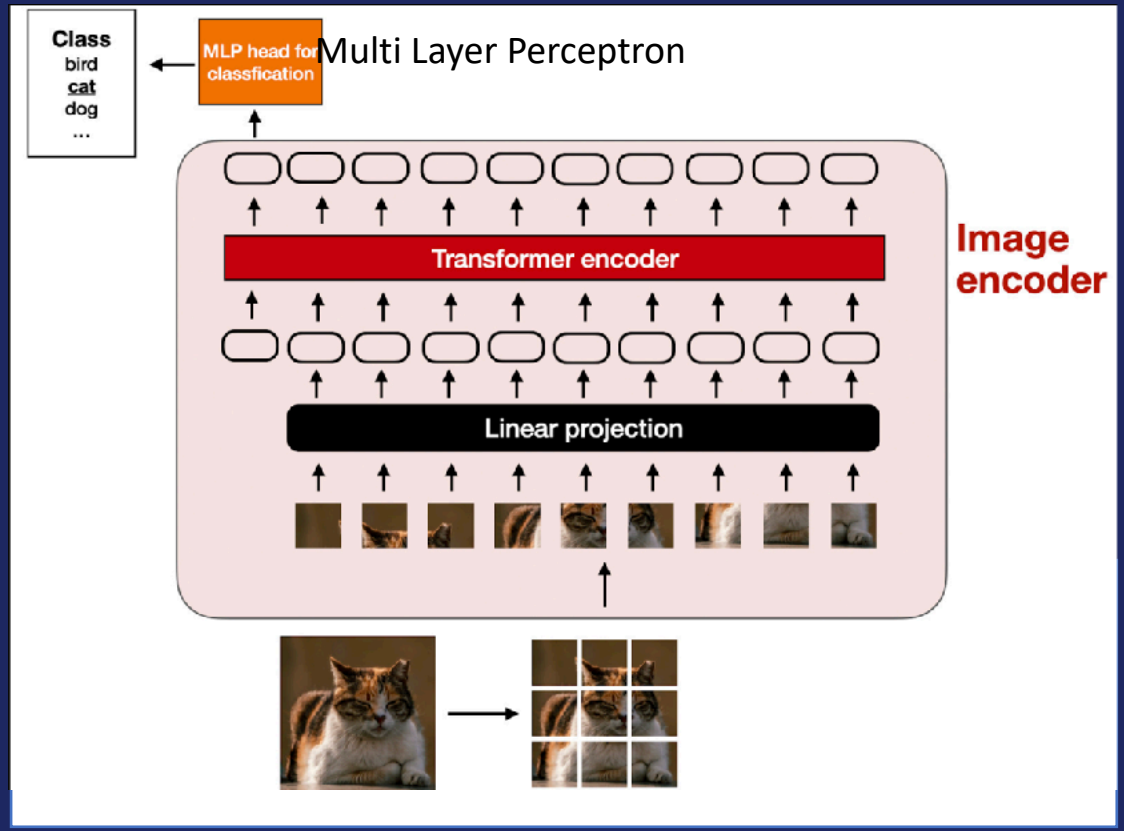
What does this mean?

- Each token (with embedding) is, for example, 4,096 numbers (a “vector")
- Numbers encode semantic meaning and context
- Similar words have similar vectors
- *The Token’s **position** in sequence is also encoded*
- Neural network uses these vectors to understand relationships
- *Note that a word has a specific embedding sequence, even if the word has multiple meanings (e.g. bark) - that issue is resolved by the “Self Attention mechanism” (covered later)*

Image - Tokenisation & Embeddings

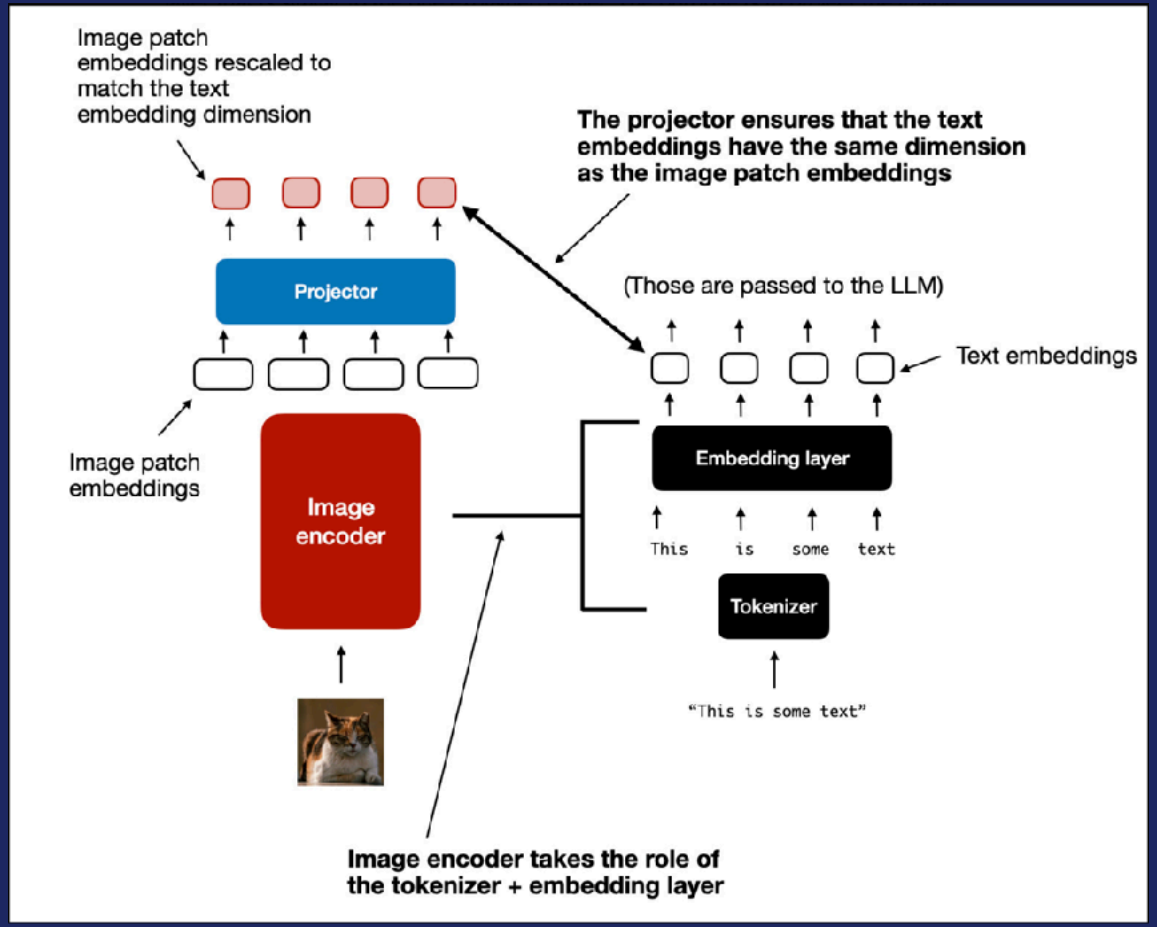
Steps to achieve:

- Resize/normalise
- split into tiles (16x16 pixels)
- Give each tile a unique ID
- Process tiles through image encoder (typically based on transformer architecture) to produce embeddings
- Embeddings capture visual features: redness, vertical edges, texture smoothness etc.



Images - Tokenisation & Embeddings

- Embeddings go part way to identifying objects such as “cat”
- Embeddings are compressed to image tokens
- token/embedding stream is combined with text input for LLM processing



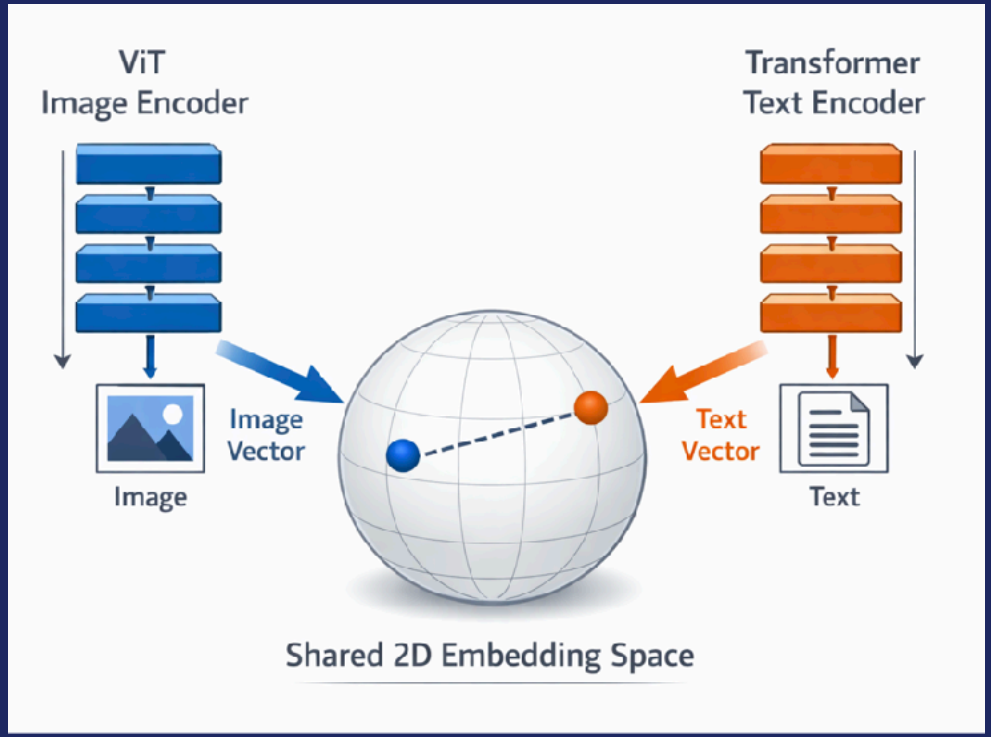
Images - Tokenisation & Embeddings

Key Differences from Text

Text	Images
Characters → tokens (discrete)	Patches → embeddings (continuous)
Vocabulary is fixed (~100k)	Vision encoder is specialized model
1 token per common word	~50-200 tokens for typical image
Direct token ID mapping	Embedding → token conversion step

Images - Tokenisation & Embeddings

- Token for text “cat” is from lookup table.
- Token for image “cat” is generated new for each image.
- Embeddings for text and image are close to each other in the “Embedding Space”



LLM Vocabulary Sizes

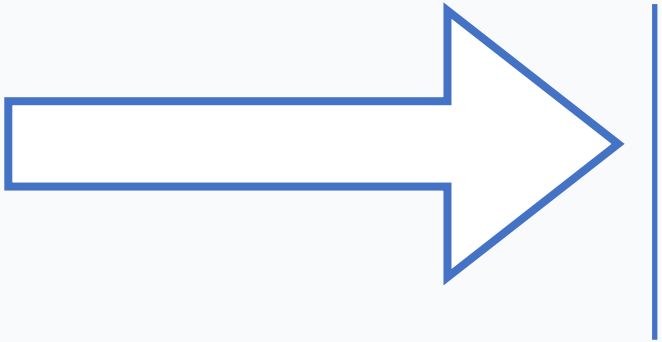
Typical token vocabularies for modern LLMs range from 50,000 to 128,000+ tokens, depending on the model architecture and design choices:

Common Sizes are:

Model Family	Vocabulary Size
GPT-2	~50,000 tokens
GPT-3 / GPT-4	~100,000 tokens
Claude	~100,000 tokens
Llama 2	~32,000 tokens
Mistral	~32,000 tokens
Gemini	~100,000+ tokens

The Full Path: Input to Output

1 Intake	2 Tokenise	3 Embed	4 Context	5 Transform	6 Plan	7 Generate	8 Deliver
Parse input	Split text	Numbers	Assemble	Attention	Orchestrate	Decode	Post-process



Context and Memory Assembly

Before processing, AI assembles *context* from multiple sources:

System Prompt

Instructions (e.g., "be helpful")

Chat History

Previous messages in conversation

User Query

Your current question

Retrieved Knowledge (RAG*)

Search results (if web search enabled)

CONTEXT WINDOW - Everything fed to transformer at once

System Prompts

Are instructions to the LLM

Real Example: Claude's System operates with a system prompt that includes things like:

- "You are Claude, an AI assistant made by Anthropic"
- Guidelines about being helpful, harmless, and honest
- Instructions about not assisting with harmful activities
- Guidance on how to handle different types of requests
- Constraints about not claiming capabilities I don't have

Retrieved Knowledge

Retrieval-Augmented Generation (**RAG**) lets the chatbot look up fresh or private information at answer time instead of relying only on what it memorised during training.

1

Query

Your question is turned into a search vector.

2

Retrieve

Relevant passages are pulled from documents, a database, or the web.

3

Augment

Those passages are inserted into the model's context window.

4

Generate

The model answers using both its knowledge and the retrieved text.

Why it matters

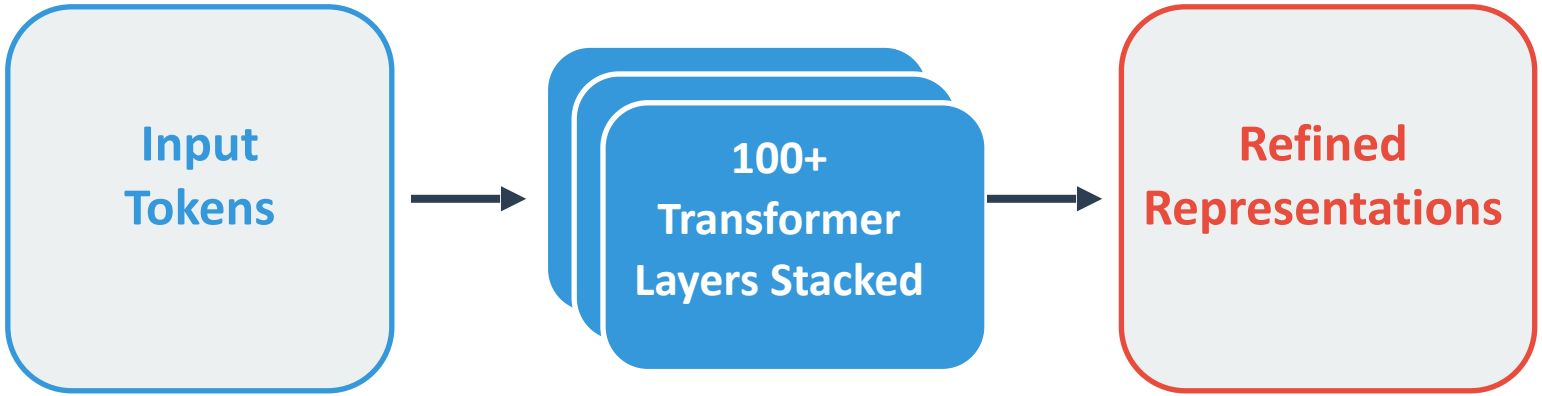
Without RAG a model's knowledge is frozen at its training cut-off. With RAG it can answer questions about today's news, your company's private manuals, or any document you upload — and it can cite the sources it used.

The Full Path: Input to Output

1 Intake	2 Tokenise	3 Embed	4 Context	5 Transform	6 Plan	7 Generate	8 Deliver
Parse input	Split text	Numbers	Assemble	Attention	Orchestrate	Decode	Post-process

The Transformer: The Brain of the AI

The transformer processes all input tokens simultaneously using self-attention.

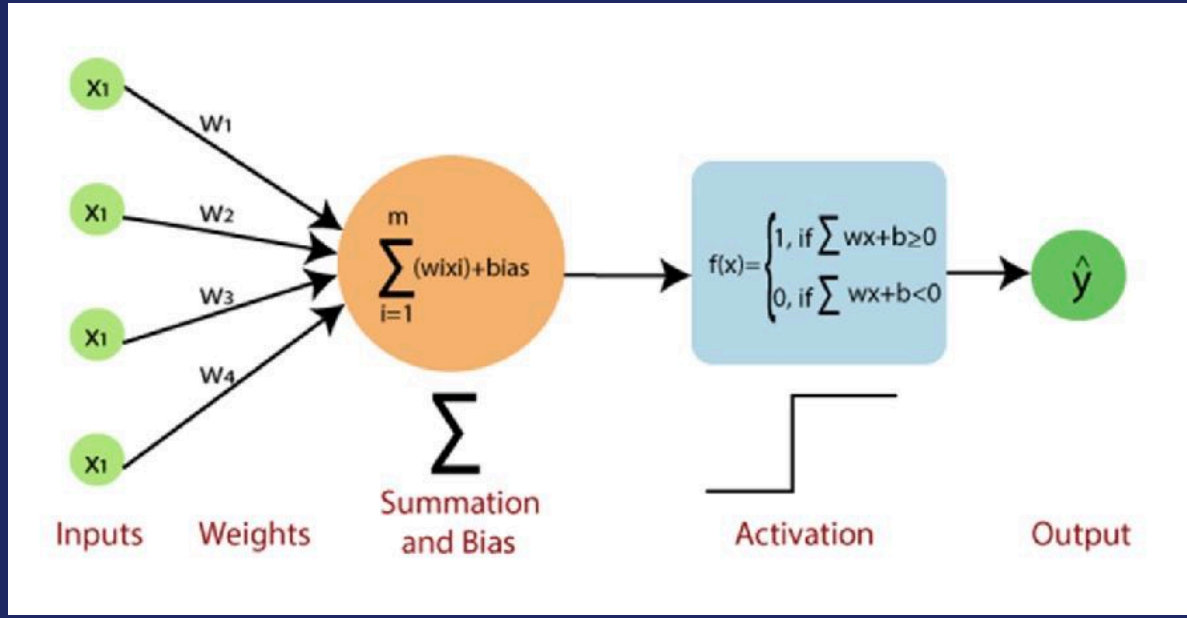


Key Mechanism: Self-Attention

Attention allows each token to "look at" every other token and determine relevance. This is how the AI understands context and word relationships.

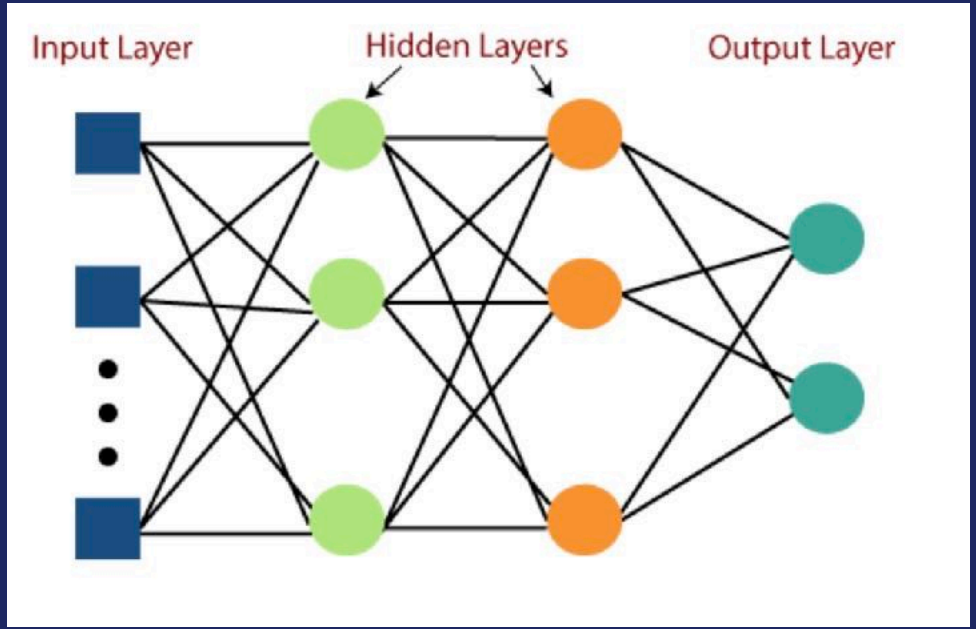
Diving Down: Perceptrons are a Foundation

- Neural networks model the brain's neurons = "perceptron"
  - Each input is given a weight
  - Each neuron operates independently
  - The neuron will "fire" if the sum of the inputs reaches a set threshold - Activation.
- The matrix of weights and biases form the "knowledge" of an LLM
- Modern LLMs use non-linear activation functions - probabilities or gating functions.



Earlier LLMs were essentially Multi-layer Perceptrons

- With multiple layers of perceptron:
  - every neuron receives all outputs from preceding layer
  - weights and Activation function operate for every neuron.
- Multi-layer perceptrons are able to model complex relationships
- These MLPs are used in earlier Recurrent Neural Networks
- They also form the basis of Feed-forward Layers in the Transformer Architecture



Seq2seq encoder-decoder model

- LLMs developed from neural machine translation
- This fixed-length context vector is incapable of remembering long sentences - and would lose meaning.
- Attention mechanisms were developed to help memorise long source sentences.

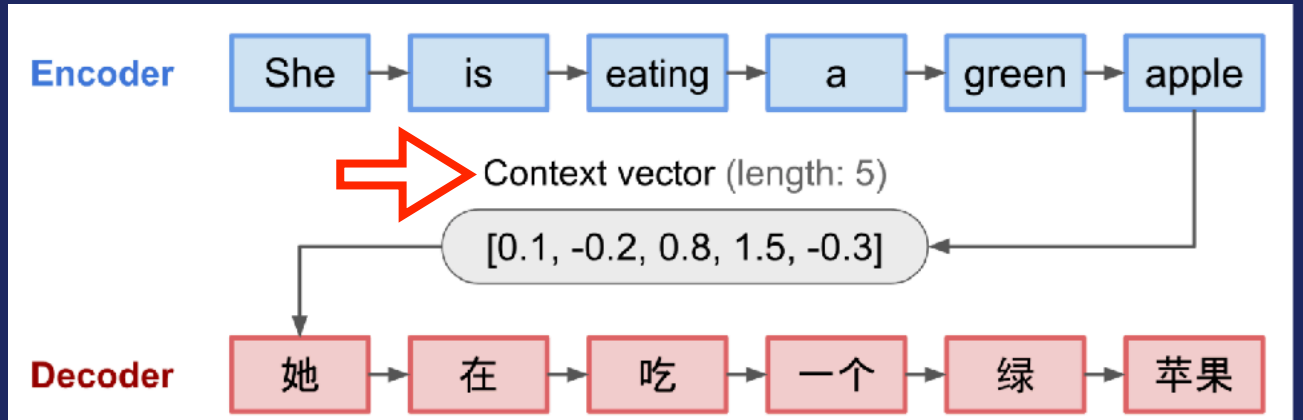


Figure 3: The encoder-decoder model, translating the sentence "she is eating a green apple" to Chinese. The visualization of both encoder and decoder is unrolled in time.

Note the Encoder - Decoder structure which can be seen in LLM architectures

LLMs now use the Transformer Architecture

- In 2017, Google researchers published “Attention is all you need” (Vaswani, et al), a paper that became the basis for modern LLMs
- They proposed the “transformer” model built on ***self-attention*** mechanisms
- Self-attention is a way of calculating the level to which every token should pay attention to every other token
- Words are “understood” in context to obtain the meaning.
- The transformer model also has a key advantage that it does not require feedback loops to achieve this.

This and following slides on attention based on super post by Lillian: https://lilianweng.github.io/posts/2018-06-24-attention/

Self Attention

- Self Attention relates different positions of a sentence
- In the diagram, the blue shade indicates the activation level of each word for the word in red

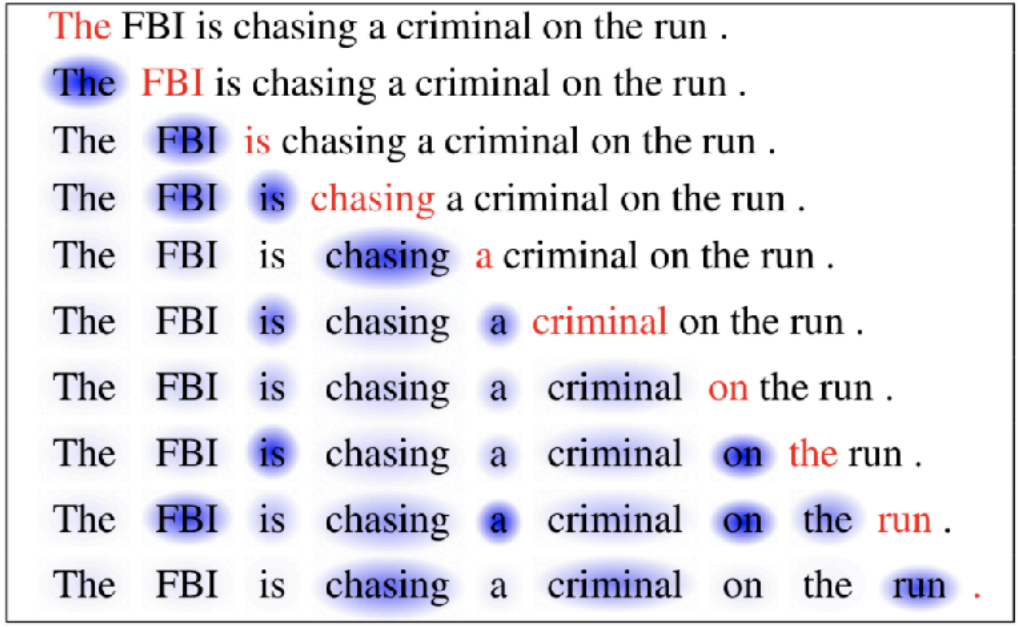


Figure 6: The current word is in red and the size of the blue shade indicates the activation level. (Image source: Cheng et al., 2016)

Attention

An attention mechanism prioritises (or attends to) the most relevant parts of input data, and draws out contextual meaning:

- Consider a language model interpreting the English text:

"on Friday, the judge issued a sentence."

- ***the*** suggests that ***judge*** is acting as a noun—i.e. person presiding over a legal trial—rather than a verb meaning to appraise or form an opinion.
- The context for the word ***judge*** suggests that ***sentence*** probably refers to a legal penalty, rather than a grammatical "sentence."
- The word ***issued*** further implies that sentence is referring to the legal concept, not the grammatical concept.
- Therefore, when interpreting the word ***sentence***, the model should pay close attention to ***judge*** and ***issued***. It should also pay some attention to the word ***the***. It can more or less ignore the other words.

The Transformer Architecture

The heart of modern chatbots. A stack of identical layers that repeatedly refine the meaning of every token by looking at every other token.

Self-Attention

Each token computes how relevant every other token is. "The cat sat on the ___" — attention links "cat" strongly to the blank.

Feed-Forward

A small neural network applied independently to each token. Adds non-linear capacity to transform representations.

Residual + Norm

Skip connections and layer normalisation keep training stable across dozens or hundreds of layers.

Transformer Architecture

- This is (essentially) the full transformer model
- Already covered:
  - Embeddings
  - Positional Encoding
  - Feed-forward
- Next, look at the (Multi-head) attention block

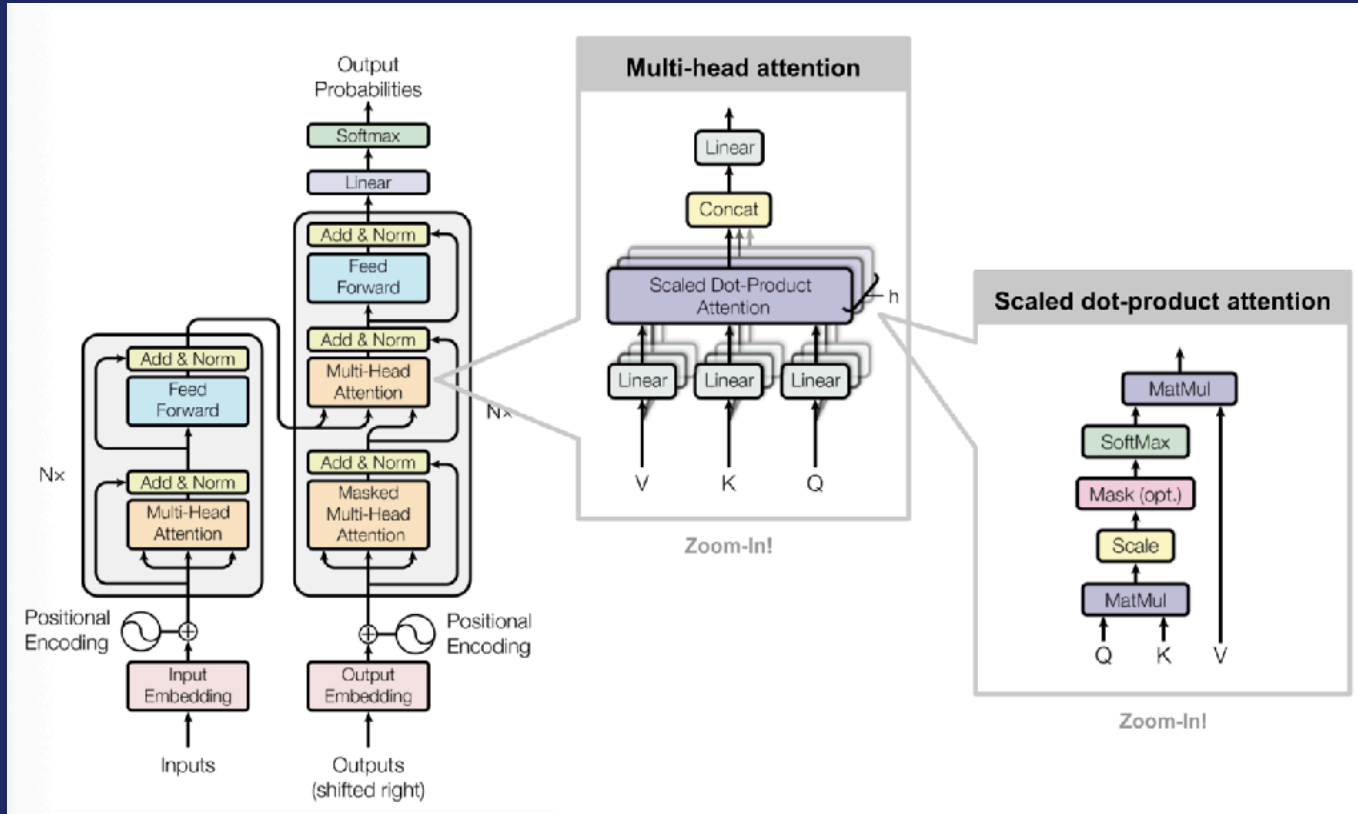


Figure 17: The full model architecture of the transformer. (Image source: Fig 1 & 2 in Vaswani, et al., 2017.)

Self Attention - How it works

- **Self-attention** determines the relative importance of different parts of the input sequence, then influences the model to attend to important parts and disregard unimportant parts.
- Core Components:
  - Tokenisation, embeddings & Positional encoding
  - **Query, key, value vectors** – Produced from embeddings;
    - The **query** vector represents the information a given token is seeking.
    - The **key** vectors represent the information that each token contains. Alignment between query and key is used to compute attention weights.
    - The **value** (or value vector) applies the attention-weighted information from the key vectors. Contributions from keys that are strongly aligned with a query are weighted more heavily; contributions from keys that are not relevant to a query will be weighted closer to zero.

  Attention weights are computed from their dot-product scores.

  - **Multi-head attention** – Multiple attention “heads” learn different relational patterns.

Query, Key and Value - conceptual view

Query: "What am I looking for?”

Input token embedding

↓

Multiply by WQ (Query weights)

↓

Query vector: A representation of "what this token needs to know"

Example: "cat" token's query might encode: "I'm a noun, I'm subject-like, I need to know about verbs/actions"

Query, Key and Value - conceptual view

Key: "What do I offer?"

Input token embedding

↓

Multiply by WK (Key weights)

↓

Key vector: A representation of "what this token can provide"

Example: "sat" token's key might encode: "I'm a verb, I'm an action, I can be modified by nouns"

Query, Key and Value - conceptual view

Value: "What information do I contain?"

Input token embedding

↓

Multiply by WV (Value weights)

↓

Value vector: The actual information/content to be retrieved

Example: "sat" token's value might contain: "past tense action, requires subject and object, indicates state change”

Self Attention - Q, K, V

- Q, K, V are three learned weight matrices that project token embeddings into different “roles"
  - Query asks "what do I need?",
  - Key says "what do I offer?",
  - Value contains "what information I have".

- The dot product of Q and K determines which tokens match.
- The value gives meaning to the match

Multi-head attention

- A single “head” could only focus on one type of relationship
- Multiple heads might relate to subjects, objects, pronouns, references..

    - Head 1: Specialises in subjects
    - Head 2: Specialises in objects
    - Head 3: Specialises in pronouns
    - Head 4: Specialises in distant references
    - ...
    - Head 64: Specialises in something else

But remember these are machine learned and not defined by a designer.

Transformer Layers

- Each Transformer layer (blocks in this diagram) is made up of an attention layer and a feed forward layer.
- Multiple alternating Attention and Feed-Forward layers process the context:

Attention: "Which tokens are relevant?"

↓

Feed-Forward: "What computation should I do with this info?"

↓

Attention: "Now, which tokens matter given what I just computed?"

↓

Feed-Forward: "Process that again..."

↓

(repeat for deep reasoning)

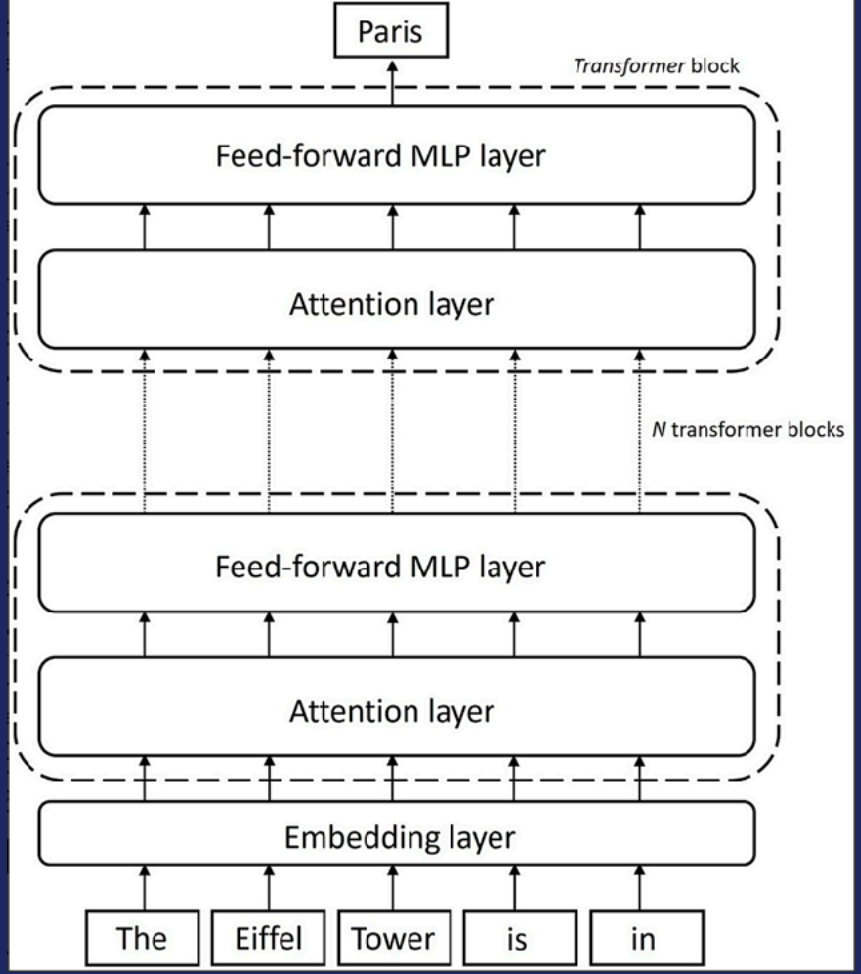
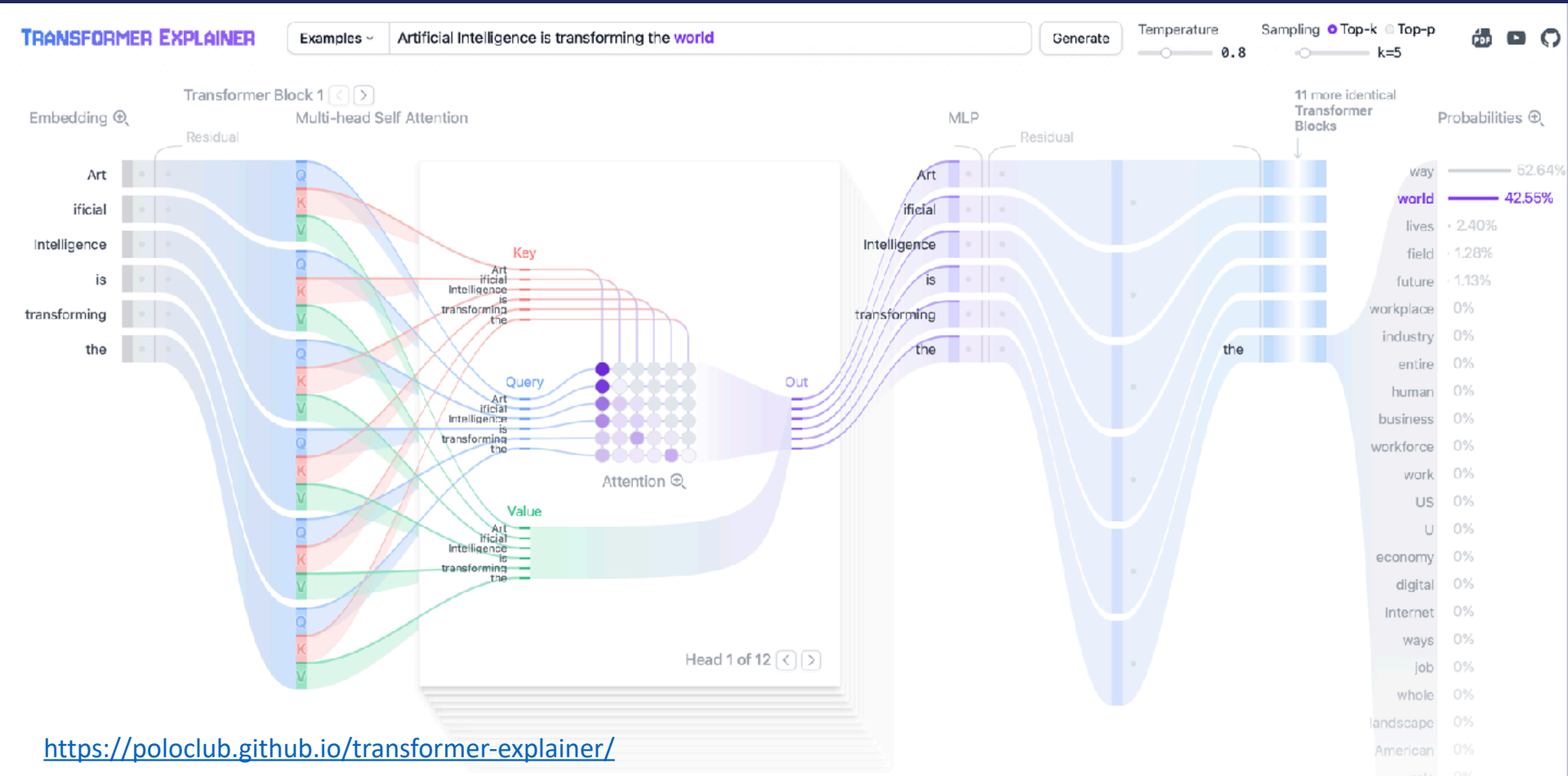


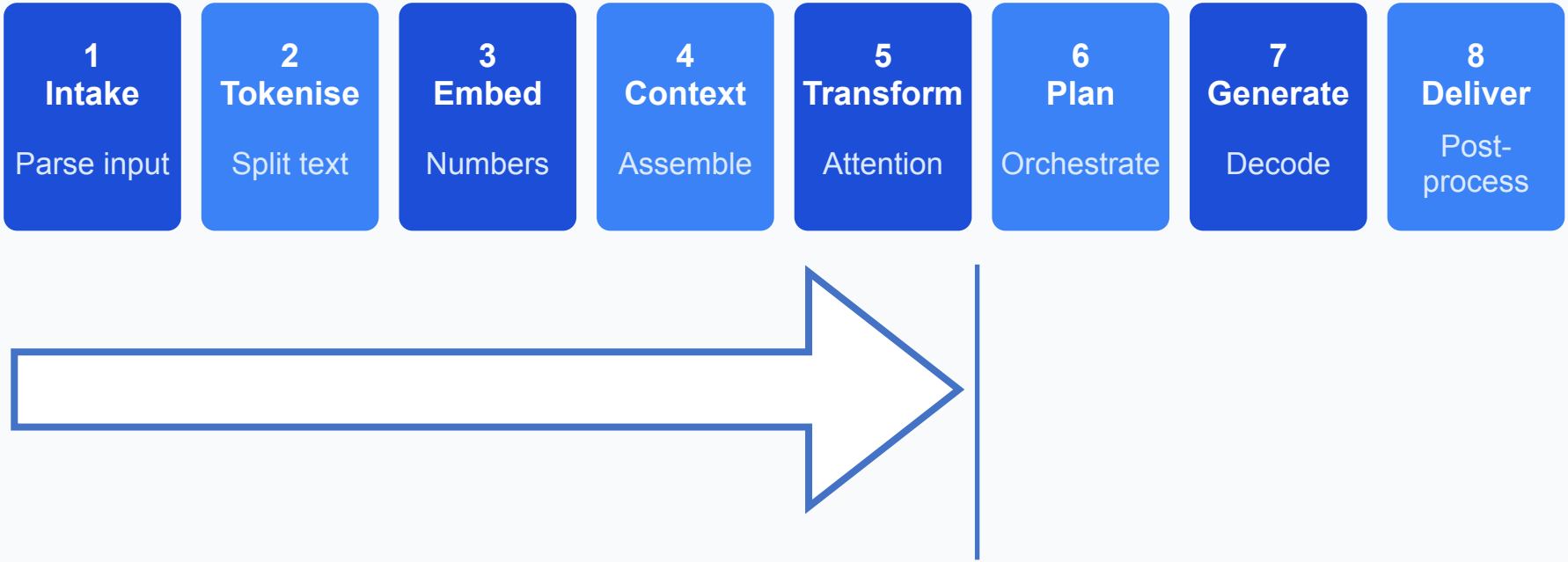
Diagram from:

Transformer Explainer - shows example phrases moving through a transformer block
https://poloclub.github.io/transformer-explainer/



https://poloclub.github.io/transformer-explainer/

The Full Path: Input to Output



Planning and Orchestration

Modern chatbots are not a single forward pass. An outer controller may decide what to do next.

A

Decide

Is a tool needed? Search? Code execution? Or just answer?

B

Call tools

Run web search, calculator, code interpreter, or external APIs.

C

Re-inject

Feed tool results back into the context window.

D

Repeat

Loop until the model produces a final answer (or hits a limit).

Planning and Orchestration can be accomplished within a transformer model, but for more complex tasks, systems use an external orchestration layer

Scaffolded Orchestration (Tool Use)

User: "Calculate 1523 × 847 then explain the answer"

- Orchestration layer:
  1. Parse user request
  2. Identify need for calculation
  3. Call calculator tool
  4. Get result: 1,290,481
  5. Pass result back to transformer
  6. Transformer generates explanation

- The transformer generates which tool to use and what parameters, but a separate system executes it.
- Without the tool use a pure transformer can hallucinate steps and produce incorrect answers

Multi-Step Orchestration (Agent Loop)

User: "Build a web scraper to get stock prices"

Orchestration loop:

Iteration 1:
- Transformer: "I need to write Python code"
- Tool: Execute code editor
- Result: Code written

Iteration 2:
- Transformer: "I need to test this"
- Tool: Execute Python
- Result: Error output

Iteration 3:
- Transformer: "I see the error, fixing it..."
- Tool: Execute code editor again
- Result: Fixed code

Iteration 4:
- Transformer: "Testing again"
- Tool: Execute Python
- Result: Success!

This loop is NOT transformer logic—it's orchestration logic But the transformer drives each decision

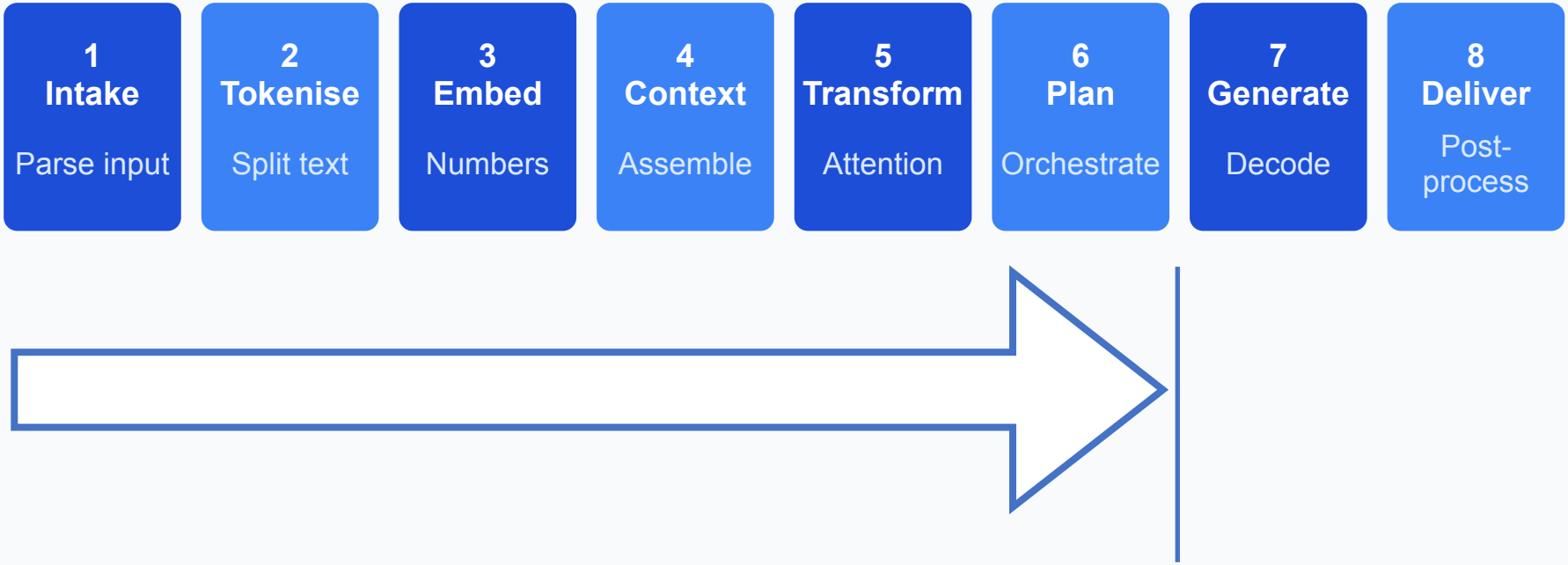
Planning and Orchestration

Planning and orchestration use:

1. **Transformer mechanisms** for reasoning and deciding what to do
2. **Non-neural logic** for orchestrating, validating, and executing
3. **Feedback loops** between them

The transformer is pattern matching at scale—very good at predicting what humans would write. But scaffolding ensures actual execution rather than hallucinated text about execution.

The Full Path: Input to Output



Token Generation: How Responses Are Created

AI generates responses one token at a time, using everything processed so far:

Then repeat steps 2-4 until AI decides to stop

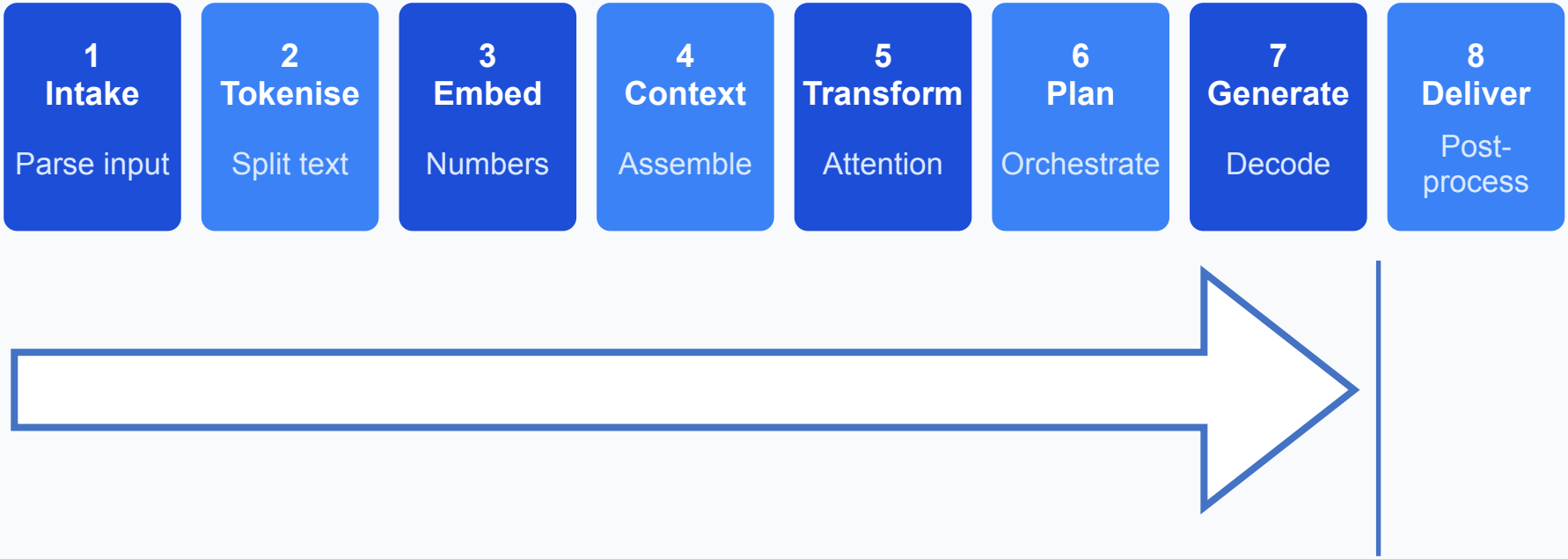
Example: "Hello," → "how" → "can" → "I" → "help" → "you" → "?" (7 tokens generated sequentially)

Token Generation (Decoding)

The model predicts the probability of every possible next token, then samples one. This repeats until an end-of-sequence token appears.

1. Forward pass	Context → transformer layers → logits (raw scores for every token in the vocabulary).
2. Softmax	Convert logits into a probability distribution that sums to 1.
3. Sampling	Greedy (always pick highest), temperature, top-p, or beam search. Controls creativity vs. determinism.
4. Append and loop	Chosen token is added to the context; the process repeats for the next position.

The Full Path: Input to Output



Post-Processing: Preparing the Output

After token generation, several processing steps prepare output for display:

Decode	**Convert token IDs back to text**
Format	**Parse markdown, code blocks, markers**
Safety Check	**Scan for policy violations**
Artifact Gen	**Create files, visualisations**
Display	**Send to your screen**

Multimodal Output: Text, Code, and More

AI can format responses in structured ways:

Plain Text

Conversational response

Code Blocks

Python, JavaScript, SQL

Structured Data

JSON, tables, lists

Artifacts

HTML, React, PDFs, docs

Example dimensions - Kimi K3 model

Architectural & Model Dimensions

- **Total Parameters**: 2.78 trillion
- **Activated Parameters**: 104.2 billion per token
- **Context Length**: Up to 1,000,000 tokens
- **Hidden Dimension**: 7,168
- **Latent MoE Dimension**: 3,584
- **Layers**: 93
- **Attention Heads**: 96
- **Routed Experts**: 896 (16 active per token)

Mixture of Experts activate only portions of network that matter for given token

https://huggingface.co/blog/ResterChed/kimi-k3-model-overview-mxfp4-quantization-open-wei

Resources and References

- https://poloclub.github.io/transformer-explainer/
- https://towardsdatascience.com/transformers-in-action-attention-is-all-you-need-ac10338a023a/
- https://lilianweng.github.io/posts/2018-06-24-attention/
- Wikipedia of course

- What are word Embeddings: https://www.youtube.com/watch?v=hVM8qGRTaOA
-

Thank You!

Questions?