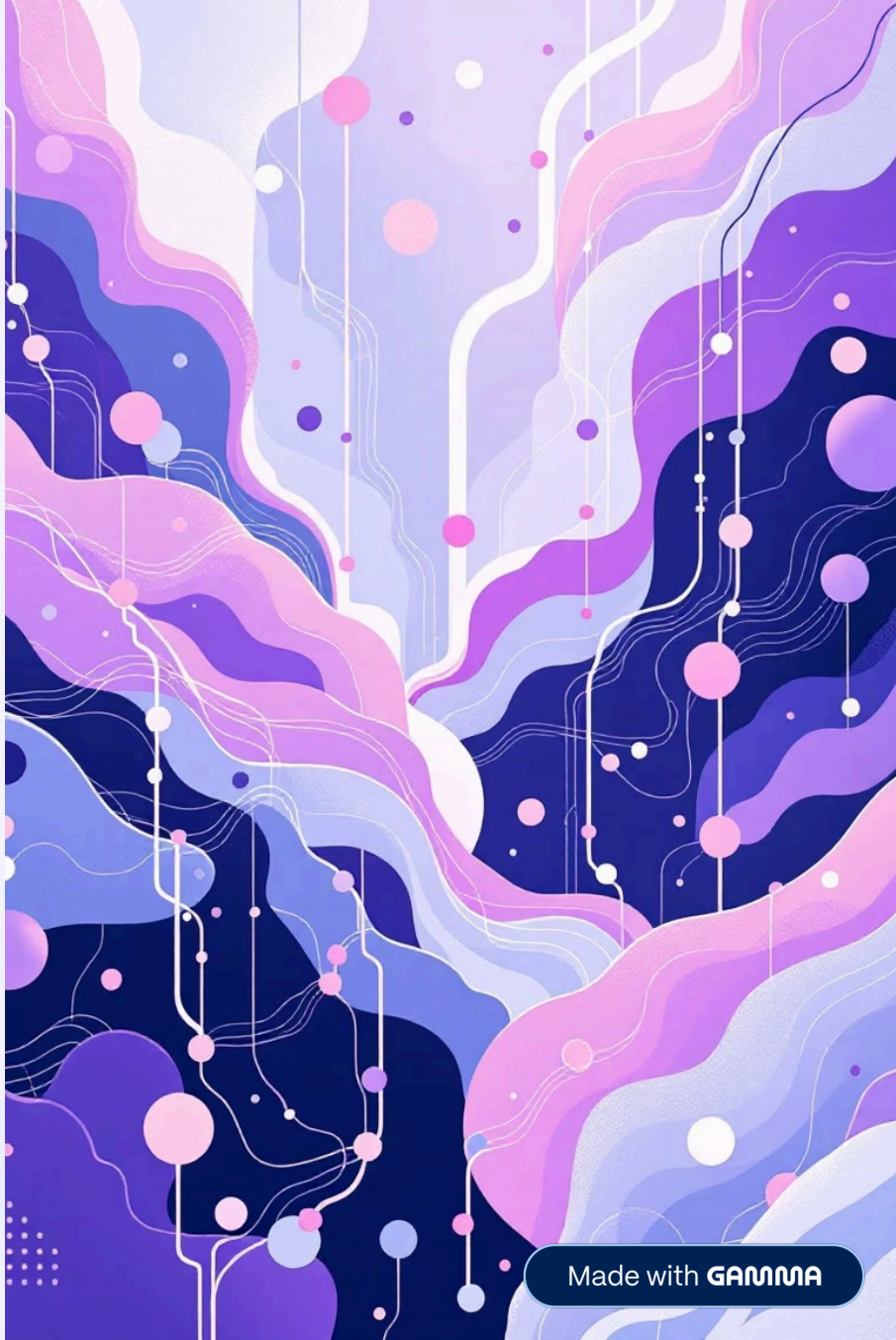
Step 5 – Feed-Forward Network (FFN)

Adding Non-Linearity and Depth to Each Token's Understanding



Where We Are in the Encoder

Each token has already journeyed through multiple transformation stages in the encoder:

01

Self-Attention

Tokens learn contextual relationships from surrounding words in the sequence

02

Add & Norm

Residual connections and normalisation stabilise the learning process

03

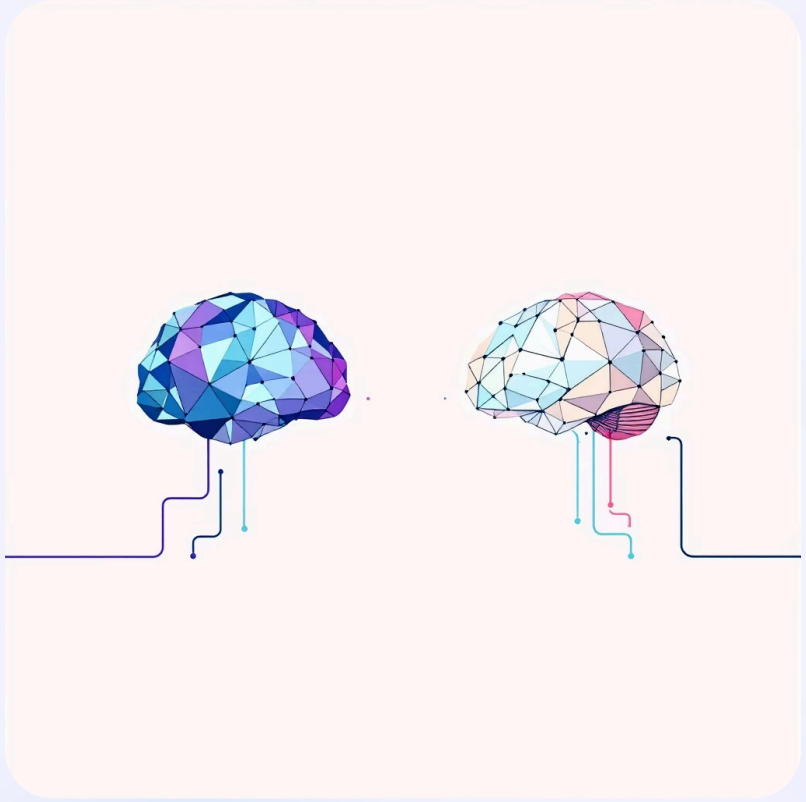
Feed-Forward Network

Deep non-linear transformations help each token learn richer, more abstract semantic meanings

What Is the Feed-Forward Network?

The FFN is a **small neural network** applied **individually** to every token vector in the sequence. Importantly, the same network with identical weights is used across all positions.

This uniform application is why it's called **Position-wise FFN** – each position gets the same treatment, but the operation happens independently for each token.



Why Do We Need FFN?

The Feed-Forward Network serves several critical purposes in the transformer architecture:

Add Depth

Applies deeper non-linear transformations to each token's representation

Richer Understanding

Enables the model to learn complex, abstract patterns beyond linear relationships

Independent Processing

Works on each token separately after it has already acquired context from attention

Complements Attention

Attention mixes tokens together; FFN refines each one individually

Mathematical Formulation

The Feed-Forward Network applies two linear transformations with a ReLU activation function in between:

$$\mathrm{FFN}(x) = \mathrm{ReLU}(xW_1 + b_1)W_2 + b_2$$

Understanding the components:

Symbol	Meaning
x	Input vector from the previous layer (post-attention)
W_1, W_2	Weight matrices – learned parameters that transform dimensions
b_1, b_2	Bias terms – additional learnable parameters for each layer
ReLU()	Activation function that keeps positive values, sets negative values to zero

Step-by-Step Flow

Let's trace how a single token vector transforms through the FFN:

1

Input Token Vector

Start with token embedding **x** (512-dimensional)

2

First Linear Transformation

Multiply by $\mathbf{W_1}$ → expand dimensions (512 → 2048)

3

Apply ReLU Activation

Introduces non-linearity by zeroing negative values

4

Second Linear Transformation

Multiply by $\mathbf{W_2}$ → compress back (2048 → 512)

5

Add Bias Terms

Include learnable biases $\mathbf{b_1}, \mathbf{b_2}$ at each layer

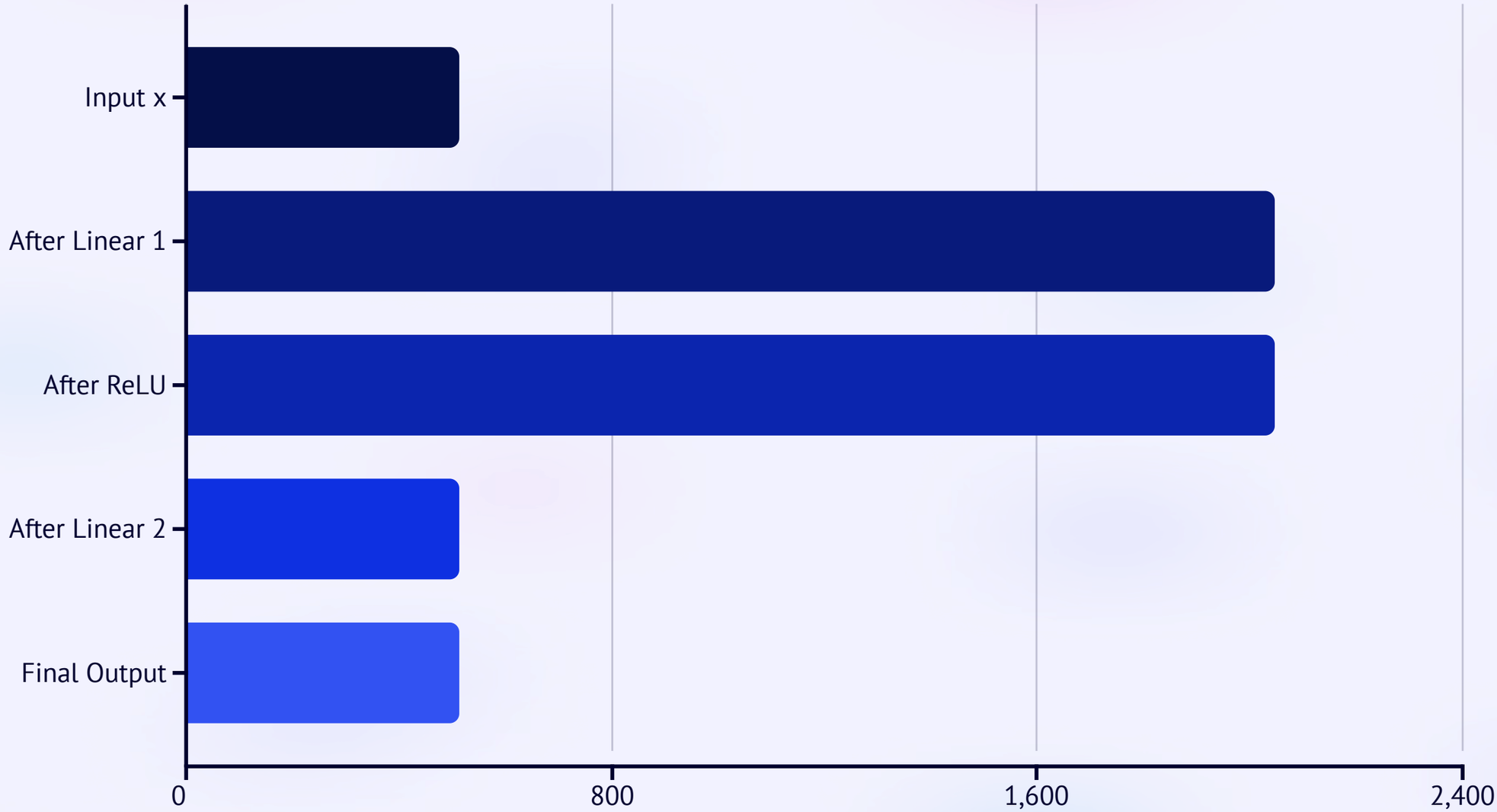
6

Output Refined Vector

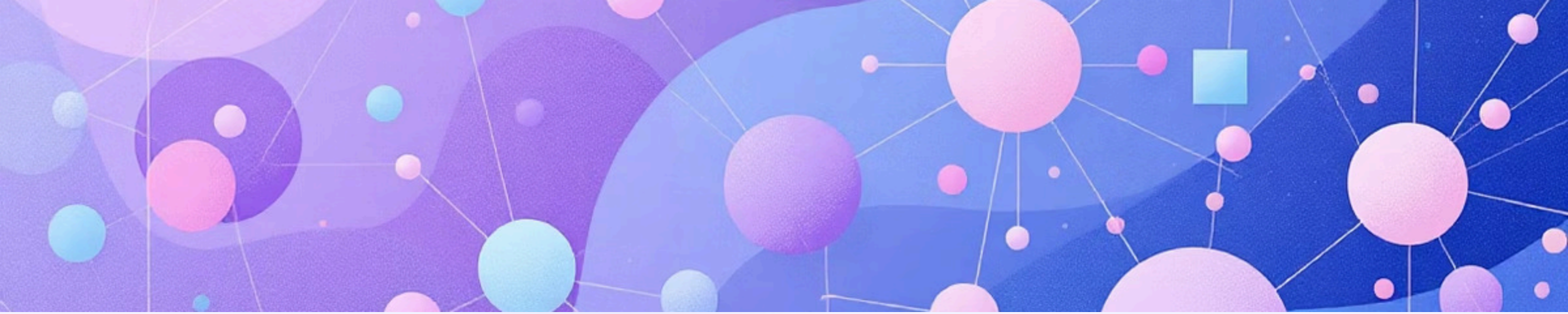
Final enriched representation (512-dimensional)

Example with Shapes

Here's how tensor dimensions change as we pass through each operation in the FFN:



The key insight: each token's embedding is **stretched** to a higher dimension, **activated** non-linearly, then **compressed** back to create a richer, more expressive representation.



Why Apply Separately to Each Word?

Context Already Acquired

Each token already has context from the attention mechanism

Individual Refinement

FFN refines **its own** meaning individually, not other tokens

Shared Weights

The same FFN with identical weights is reused across all positions

Think of it this way: "Every word now knows the context from attention – the FFN lets us polish and refine what each word means individually."

Intuition Analogy

🧩 Attention

How words **talk to each other** and share information across the sequence

⚙️ FFN

How each word **processes what it heard** and deepens its understanding

The FFN gives the model **"brainpower"** to combine features in non-linear ways, enabling it to capture complex relationships that linear transformations alone cannot express.

Quick Recap

1️⃣ Input

Contextualised word vector from attention + normalisation

2️⃣ Transform

Linear + ReLU + Linear creates non-linear transformations

3️⃣ Output

Refined representation maintaining the same dimension

✅ Purpose

Adds depth, expressivity, and non-linearity to the model's representations

🔁 Application

Applied to each token individually in a position-wise manner