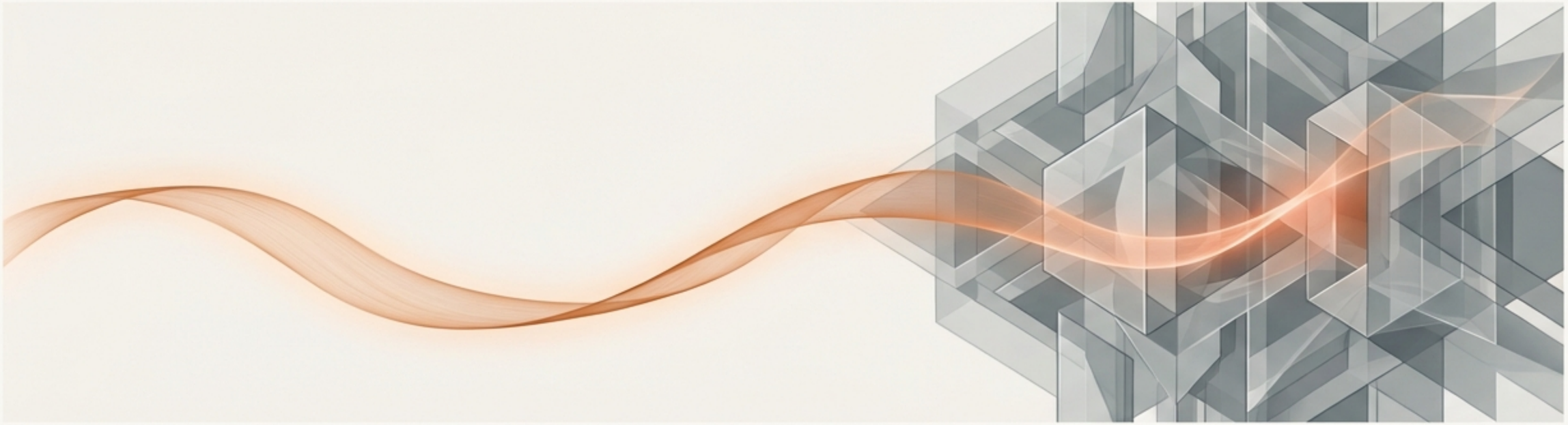
La Odisea de un Prompt

Anatomía de la Inferencia en un Transformer



Seguiremos el viaje de un simple texto de entrada, desde su origen como palabras hasta su transformación final en una predicción. Este es el flujo de datos que permite a un modelo como GPT-2 comprender y generar lenguaje, paso a paso.

Hola mundo. Esta es una prueba de tokenización real.

Etapa 1: La Traducción a Números (Tokenización)

Todo comienza convirtiendo el texto en una secuencia de IDs numéricos, o 'tokens'. El modelo no ve palabras, sino una lista de identificadores únicos extraídos de su vocabulario.

CÓDIGO

```
texto = "Hola mundo. Esta es una prueba de
tokenización real."

tokens = tokenizer(texto, return_tensors="pt")
```

RESULTADO

Hola mundo. Esta es una prueba de tokenización real.

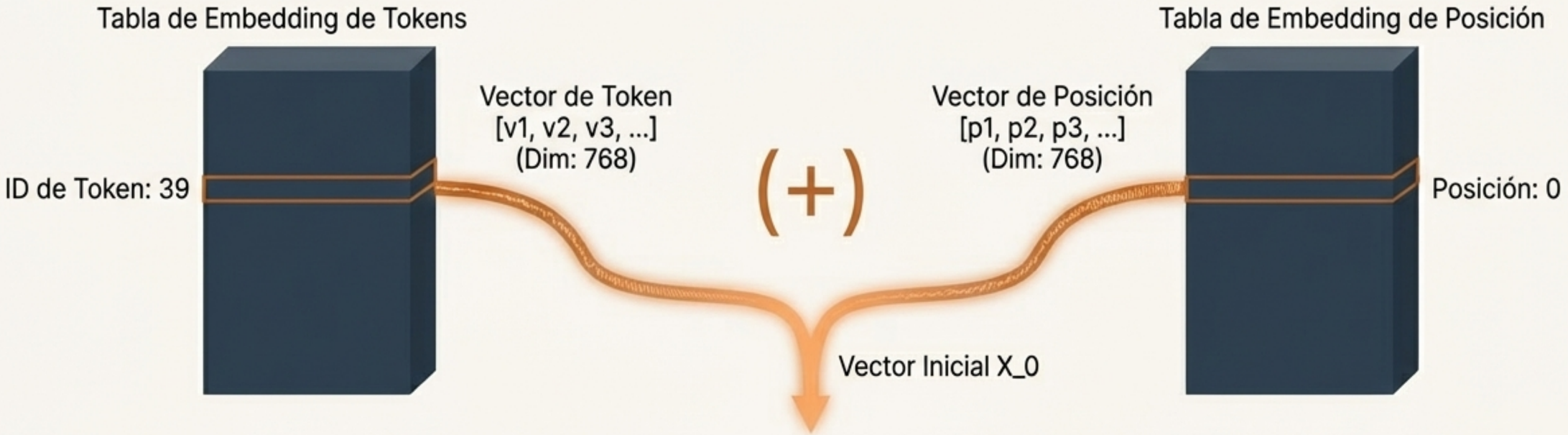

```
# Salida: 20 tokens
tensor([[ 39, 5708, 27943, 78, 198, 22362, 64, 1658, 555,
          556, 8451, 20139, 1634, 78, 39, 5708, 27943, 78,
          198, 22362]])

# Shape: torch.Size([1, 20])
```

Nuestra frase ahora es una secuencia de 20 números, la materia prima para el siguiente paso.

Etapa 2: Asignando Identidad y Contexto Posicional

Cada token ID y cada posición en la secuencia se utilizan para consultar dos 'bibliotecas' de vectores pre-entrenados (embeddings). La suma de ambos crea el vector inicial de cada token.



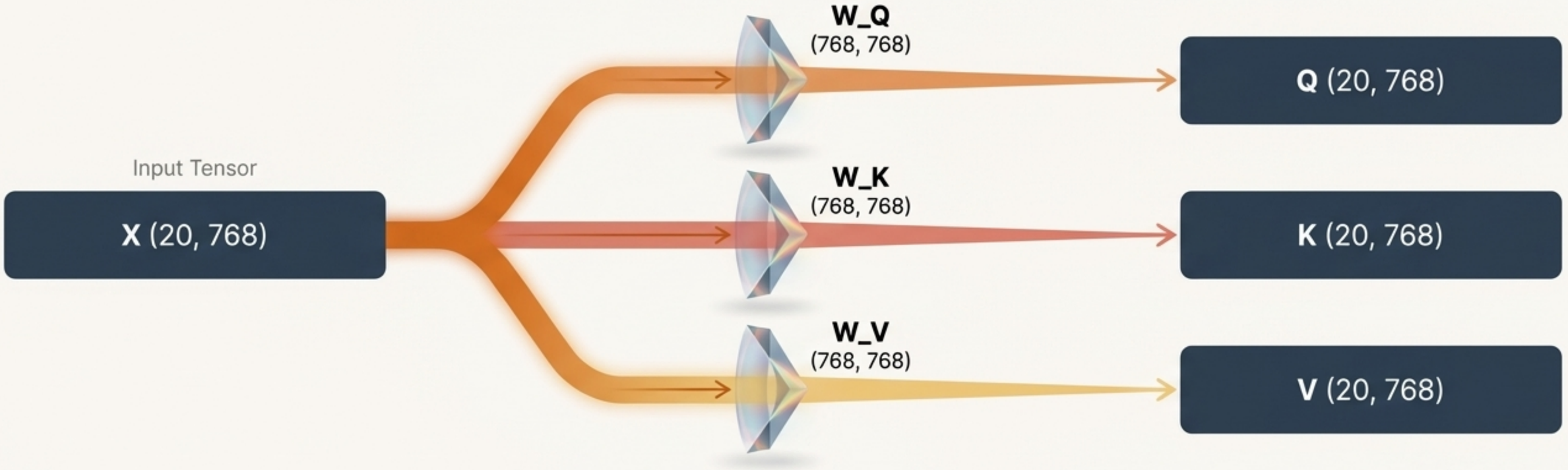
Embedding de Token (significado) + Embedding de Posición (orden) = Embedding Inicial Contextualizado.

El resultado es un tensor que representa nuestra secuencia completa. Esta es la forma que mantendrá a lo largo de su viaje.

```
Shape final: torch.Size([1, 20, 768])
# [batch_size, n_tokens, embedding_dimension]
```

El Crisol: Preparando los Sentidos para la Atención (Q, K, V)

Para que cada token pueda 'mirar' a los demás, primero proyectamos su embedding en tres espacios diferentes: Query, Key y Value. Esto se logra multiplicando el embedding por matrices de pesos (W_Q, W_K, W_V) aprendidas durante el entrenamiento.



Query (Q): ¿Qué estoy buscando?

Key (K): ¿Qué información relevante tengo para ofrecer?

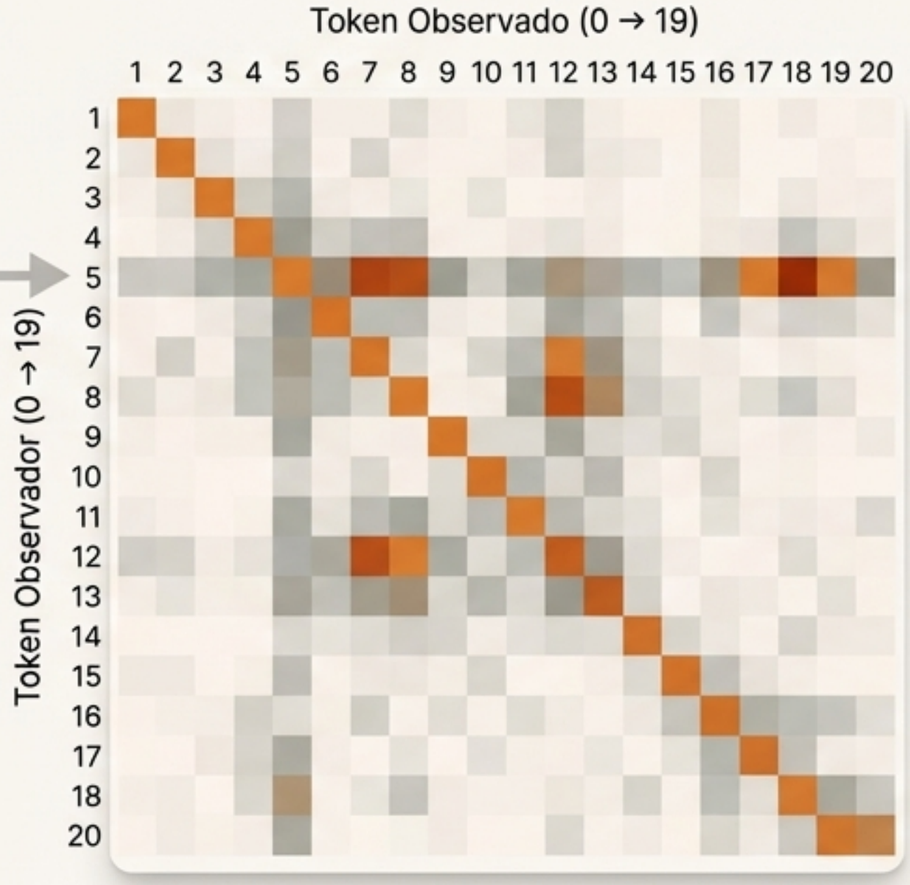
Value (V): La información que finalmente compartiré.

Las matrices de pesos W son fijas después del entrenamiento. Aquí es donde reside gran parte del "conocimiento" del modelo.

Calculando Relevancia: ¿A Quién Debo Prestar Atención?

Usando Query (Q) y Key (K), el modelo calcula una puntuación de atención entre cada par de tokens. Esta puntuación determina qué tan relevante es un token para otro en el contexto actual.

Esta fila muestra cómo el token 5 distribuye su atención entre todos los demás tokens.



$$\text{Puntuaciones de Atención} = \text{softmax}\left(\frac{\mathbf{Q} \mathbin{@} \mathbf{K}_{\text{transpuesta}}}{\sqrt{d_k}}\right)$$

Donde d_k es la dimensión de los vectores Key.

El resultado es una matriz de `(20, 20)`. Cada fila representa la distribución de atención de un token a través de toda la secuencia. La suma de los valores de cada fila es 1.

La Atención en Acción: "Hoy martes llueve"

Veamos un ejemplo simplificado para calcular el nuevo embedding de 'hoy'. Su representación final será una mezcla de los demás tokens, ponderada por las puntuaciones de atención.

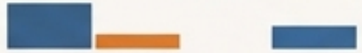
Step 1: Escenario

Queremos recalcular el vector de **hoy**.
Supongamos que las puntuaciones de atención de 'hoy' hacia los demás son:
- Hacia 'hoy': **0.2**
- Hacia 'martes': **0.7**
- Hacia 'llueve': **0.1**

Step 2: Vectores V

V["hoy"]  [0.3, 0.2, 0.1, 0.5]

V["martes"]  [-0.4, 0.6, 0.3, -0.2]

V["llueve"]  [0.7, -0.1, 0.0, 0.2]

Step 3: Multiplicación Ponderada

0.2 * V["hoy"] ×  [0.06, 0.04, 0.02, 0.10]

0.7 * V["martes"] ×  [-0.28, 0.42, 0.21, -0.14]

0.1 * V["llueve"] ×  [0.07, -0.01, 0.00, 0.02]

Step 4: Suma Final

Resultado (Embedding Score): [-0.15, 0.45, 0.23, -0.02]

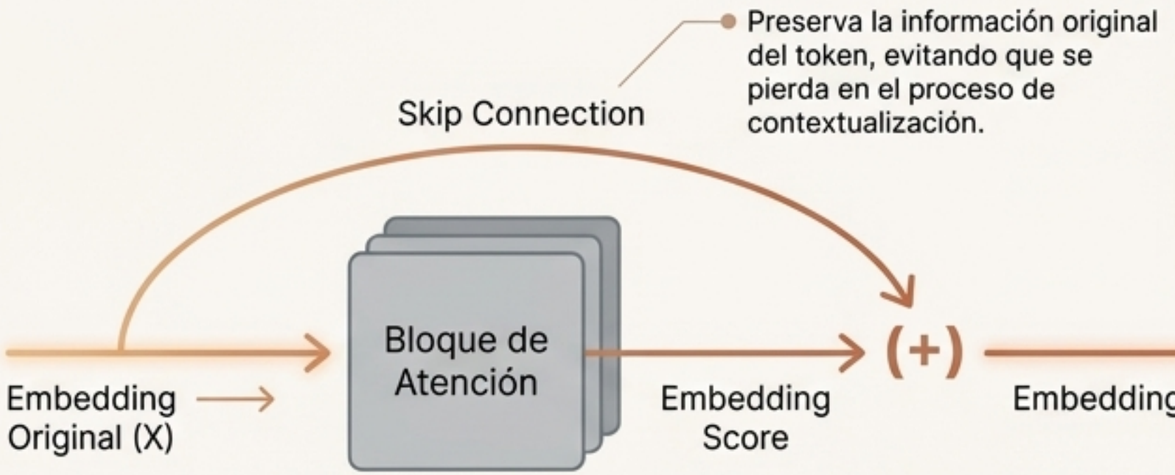
Conclusión Clave

El nuevo vector de 'hoy' ahora está fuertemente influenciado por 'martes' (peso 0.7).
Ha absorbido el contexto. Esto no es magia, es una suma ponderada.

Integración y Estabilización del Nuevo Contexto

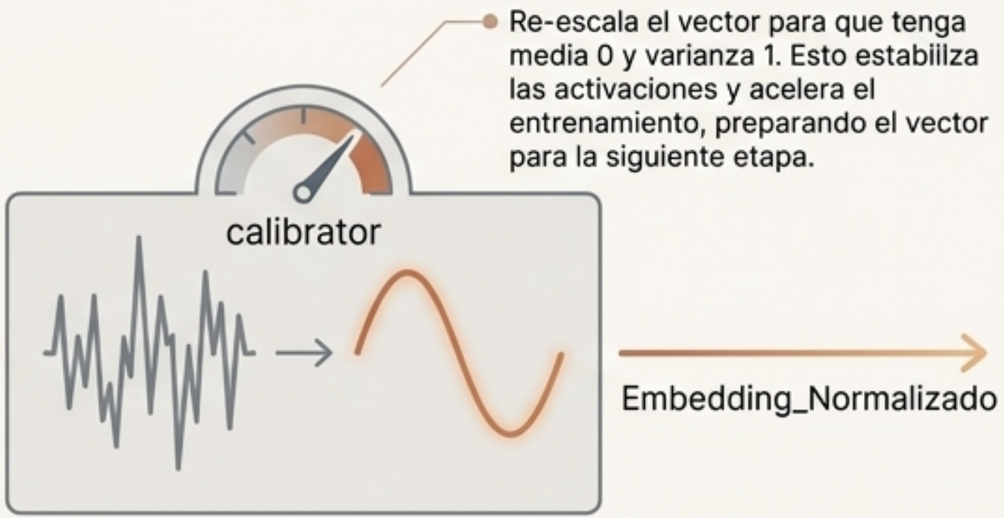
El nuevo embedding contextualizado no reemplaza al original. Se combina y estabiliza a través de dos mecanismos clave para asegurar un aprendizaje estable a través de las capas.

Fase 1: Residual Connection (Skip Connection)



Fórmula: Embedding_Residual = X + Embedding_Score

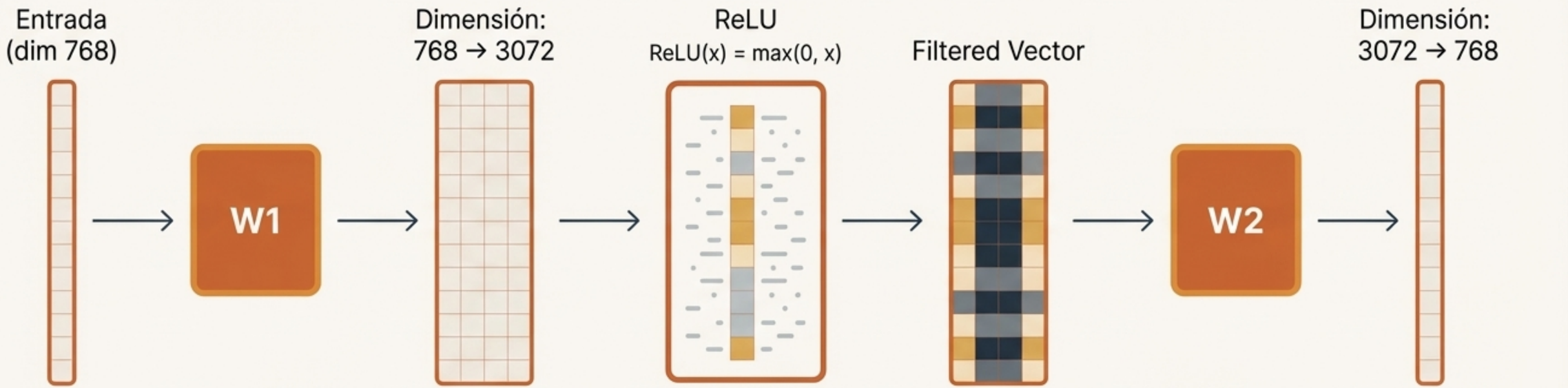
Fase 2: Layer Normalization (LayerNorm)



Fórmula: Embedding_Normalizado = $\frac{x - \text{media}}{\sqrt{\text{varianza} + \varepsilon}}$

Procesamiento Profundo: La Red Feed-Forward (FFN)

Después de la atención, cada token pasa individualmente por una red neuronal simple. Este paso permite al modelo realizar un procesamiento más complejo sobre la información ya contextualizada.



 # Propósito: Crear una representación más rica con mayor capacidad expresiva.

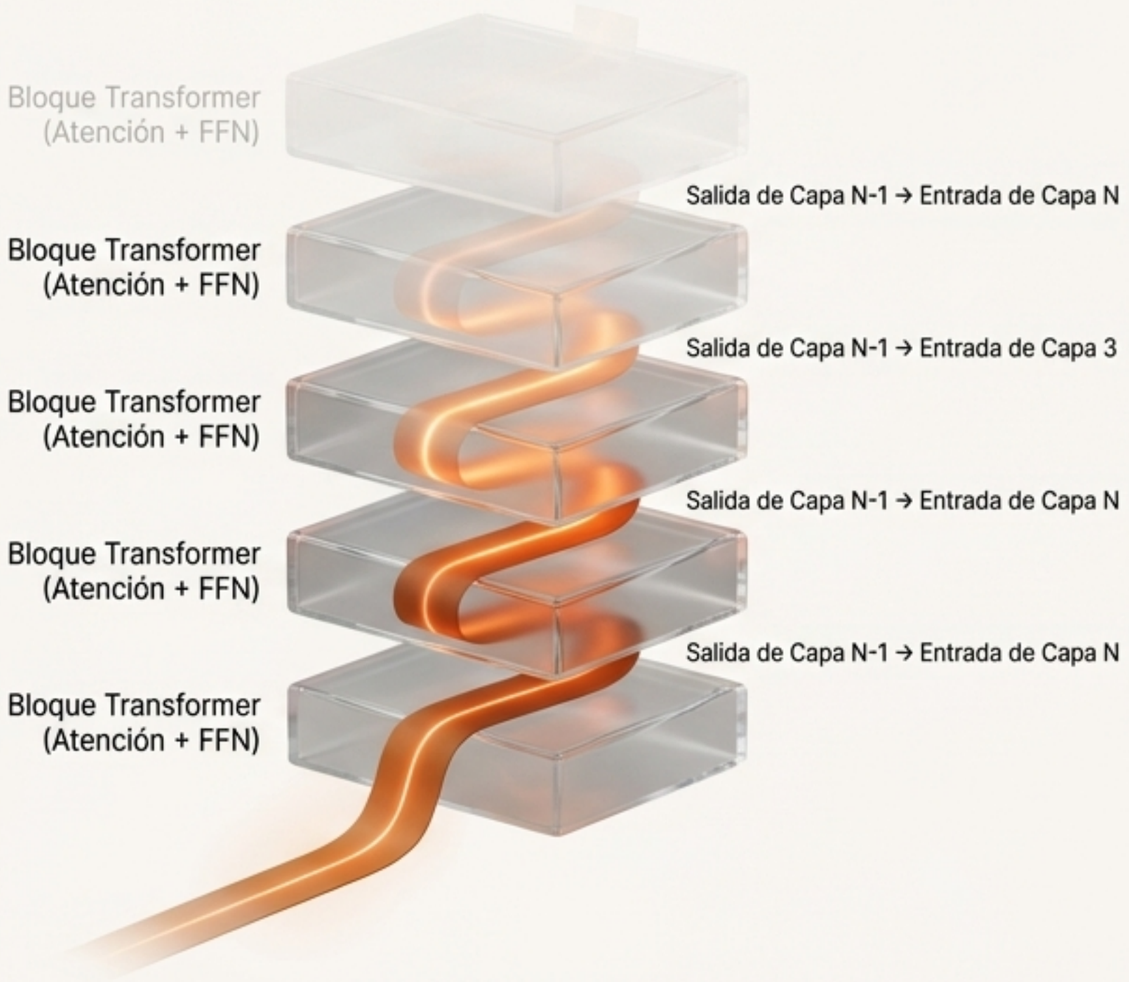
 # Propósito: Descartar activaciones no útiles e introducir no linealidad.

 # Propósito: Sintetizar los rasgos relevantes activados en la capa de expansión.

Nota: Al igual que en la atención, esta etapa también tiene su propia Residual Connection y Layer Normalization al final.

El Ciclo de Refinamiento: El Rol de las Múltiples Capas

El proceso completo (Atención + FFN) constituye un "Bloque Transformer". Un modelo como GPT-2 apila estos bloques múltiples veces (e.g., 96 veces). La salida de una capa se convierte en la entrada de la siguiente.



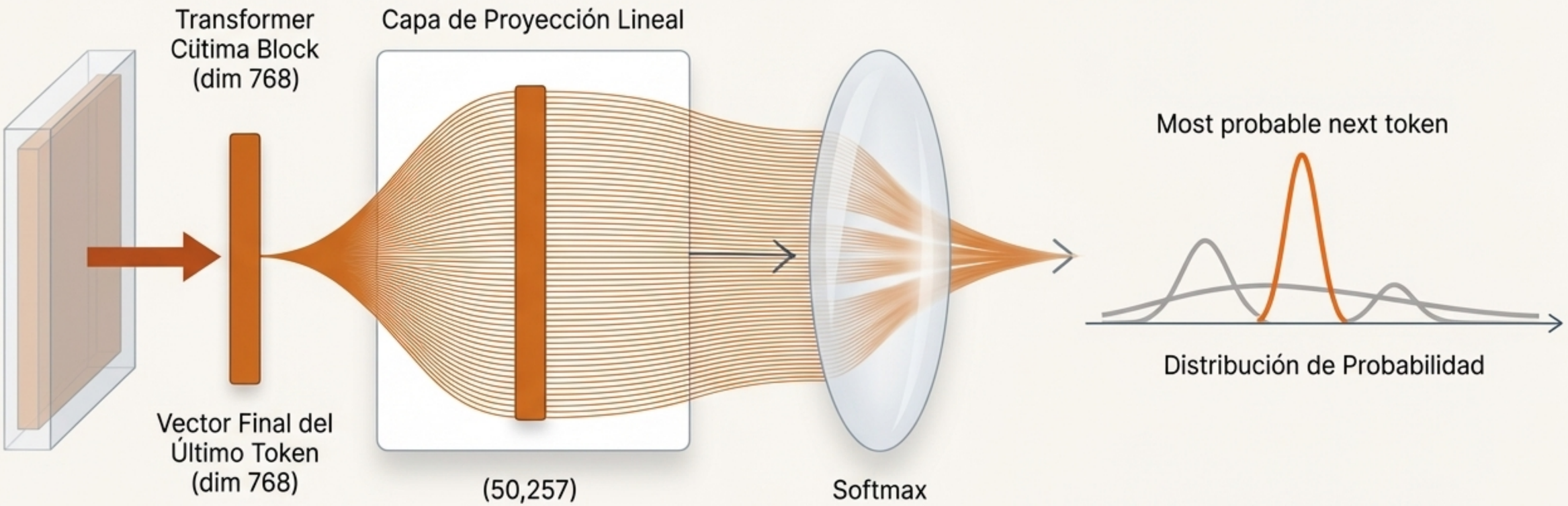
Ejemplo Conceptual: La Evolución del Token "hoy"

Tras Capa N-1:	"hoy" → [contexto completo → alta probabilidad de consecuencia logística como llevar paraguas] (Comprensión de alto nivel)
Tras Capa 3:	"hoy" → [día actual + lluvioso → afecta decisión] (Implicaciones causales)
Tras Capa 1:	"hoy" → [es día actual, está relacionado con martes] (Relaciones básicas)
Entrada (Capa 0):	"hoy" → [vector de significado y posición]

Cada capa refina la representación del token, permitiéndole capturar relaciones cada vez más complejas y abstractas con el resto de la secuencia.

El Desenlace: De la Comprensión a la Predicción

Después de pasar por la última capa, tomamos el vector de salida del último token de nuestra secuencia. Este vector, que contiene todo el contexto acumulado, se utiliza para predecir el siguiente token.



La capa final proyecta el vector de 768 dimensiones a un espacio del tamaño del vocabulario. La función Softmax convierte estos valores (logits) en probabilidades, indicando la probabilidad de que cada palabra del vocabulario sea la siguiente.

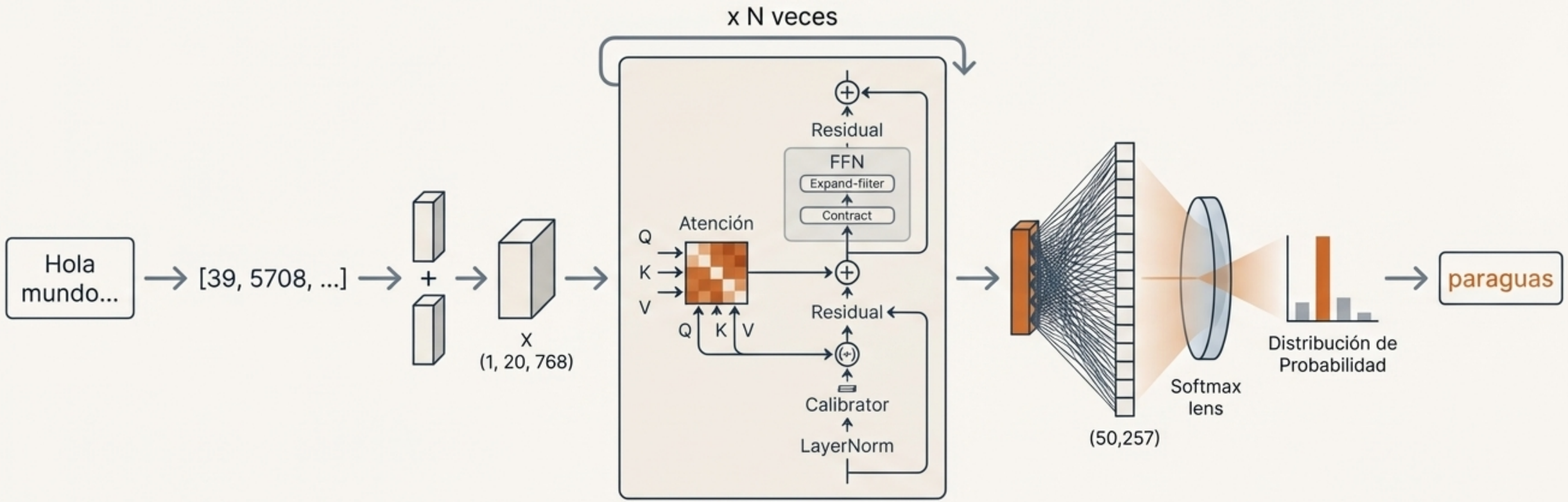
El Veredicto Final: Eligiendo el Siguiente Token

El modelo ahora tiene una probabilidad asignada a cada **posible token siguiente**. Simplemente elige el token con la probabilidad más alta (o muestrea de la distribución) para generar la continuación.

Frase de entrada: "Hoy martes llueve, asi que..."

El token 'paraguas' es seleccionado. Todo el complejo viaje de contextualización a través de las capas culmina en essa esta predicción, que se basa en los patrones matemáticos aprendidos durante el entrenamiento.

La Odisea Completa: Un Resumen Visual del Viaje



La Arquitectura del Contexto

Lo que NO es

- El modelo no 'piensa' ni 'entiende' en el sentido humano.
- No accede a una base de datos en tiempo de inferencia.
- No razona con lógica, sino que opera sobre probabilidades.

Lo que SÍ es

- Un sistema sofisticado de reconocimiento de patrones matemáticos a una escala masiva.
- La aplicación de relaciones (semánticas, sintácticas, causales) codificadas en pesos numéricos durante el entrenamiento.
- Una demostración de cómo operaciones vectoriales complejas, repetidas en capas, pueden simular la comprensión del contexto lingüístico.

*La 'magia' no es conciencia, sino una arquitectura computacional absolutamente genial que asocia 'lluvia' y 'paraguas' con una alta similitud numérica. Es la **matemática del lenguaje**.*