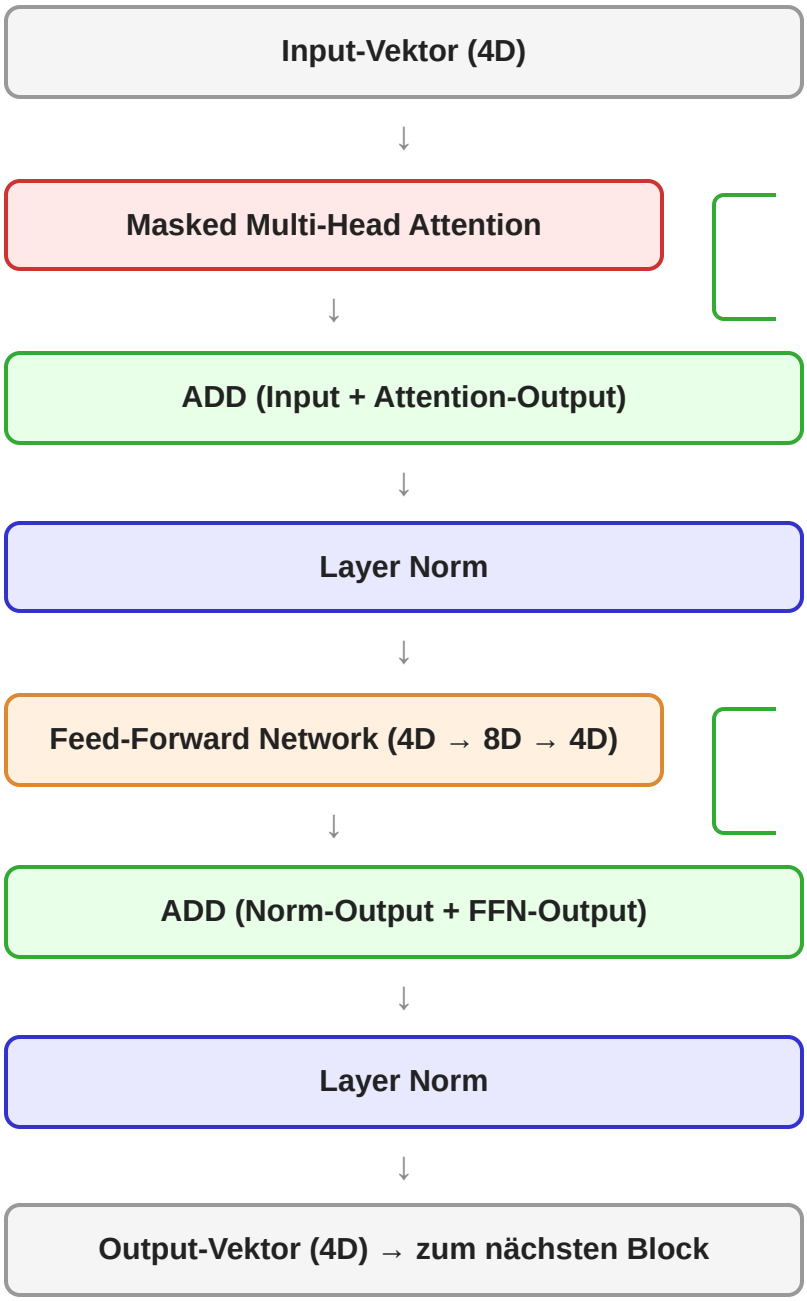
Schritt 5: Add & Norm + Feed-Forward Network

Tokenizer ✓ → Embedding ✓ → Pos. Enc. ✓ → MH-Attention ✓ → Add & Norm + FFN → Output

Der komplette Transformer-Block



> Die grünen Linien rechts sind **Skip Connections** (Abkürzungen). Sie leiten den Input am jeweiligen Schritt vorbei direkt zum ADD.
>
> **Wichtig:** Jedes Token durchläuft diesen Block einzeln. Es gibt keine Interaktion mehr zwischen den Tokens — die passiert nur in der Attention!

Rechnen wir das für **Katze** durch:

Was	Vektor
Original-Input (vor Attention)	[0.8, 1.4, 0.1, 1.2]
Attention-Output	[1.26, 1.15, 0.84, 0.06]

Add: Input + Attention-Output

Erlebnispark: *Du kommst aus den Attention-Räumen zurück. Aber bevor du weitergehst, triffst du* **dein früheres Ich** *— den Original-Vektor von vor der Attention. Ihr verschmelzt: Dein neues Wissen wird zu deinem alten Wissen* **addiert**.

So geht nichts verloren. Die Attention fügt hinzu, ersetzt aber nicht.

Original-Input:	[	0.80	1.40	0.10	1.20]
	+				
Attention-Output:	[	1.26	1.15	0.84	0.06]
Summe:	[	**2.06**	**2.55**	**0.94**	**1.26**]

Warum addieren statt ersetzen?
Ohne Skip Connection müsste die Attention *alles* über das Token erhalten — auch das was sich nicht geändert hat. Mit Skip Connection muss die Attention nur das *Neue* liefern.

Außerdem: Beim Training fließen die Gradienten durch die Skip Connection direkt zurück, ohne durch die Attention hindurch zu müssen. Das macht das Training viel stabiler.

Layer Norm: Werte stabilisieren

Layer Norm normalisiert den Vektor so, dass:
• Der **Mittelwert** aller Werte ≈ 0 ist
• Die **Streuung** (Standardabweichung) ≈ 1 ist

Das verhindert, dass die Werte nach vielen Blöcken explodieren oder verschwinden.

Schritt 1: Mittelwert berechnen

```
Vektor = [2.06, 2.55, 0.94, 1.26]
```

```
Mittelwert = (2.06 + 2.55 + 0.94 + 1.26) / 4 = 6.81 / 4 = 1.70
```

Schritt 2: Varianz berechnen

```
Abweichungen vom Mittelwert:
2.06 - 1.70 = +0.36
2.55 - 1.70 = +0.85
0.94 - 1.70 = -0.76
1.26 - 1.70 = -0.44

Abweichungen² (quadriert):
0.36² = 0.130
0.85² = 0.722
0.76² = 0.578
0.44² = 0.194

Varianz = (0.130 + 0.722 + 0.578 + 0.194) / 4 = 1.624 / 4 = 0.406

Standardabweichung = √0.406 = 0.64
```

Schritt 3: Normalisieren

```
Formel: (Wert - Mittelwert) / Standardabweichung

d₁: (2.06 - 1.70) / 0.64 = 0.36 / 0.64 = +0.56
d₂: (2.55 - 1.70) / 0.64 = 0.85 / 0.64 = +1.33
d₃: (0.94 - 1.70) / 0.64 = -0.76 / 0.64 = -1.19
d₄: (1.26 - 1.70) / 0.64 = -0.44 / 0.64 = -0.69
```

Nach Layer Norm: [0.56, 1.33, -1.19, -0.69]

> **Check:** Mittelwert ≈ 0? → (0.56 + 1.33 - 1.19 - 0.69) / 4 = 0.0025 ≈ 0 ✓
> Die Werte sind jetzt um 0 zentriert und in einem stabilen Bereich.

Feed-Forward Network: Allein nachdenken

Erlebnispark: *Die Attention war Teamarbeit — du hast mit allen gesprochen. Jetzt kommst du in einen* **stillen Raum**. *Hier bist du allein.*

Zuerst wirst du in einen **viel größeren Raum aufgeblasen** *(4D → 8D). Dort hast du Platz zum Nachdenken. Dann wirst du wieder* **komprimiert** *(8D → 4D).*

Man vermutet, dass hier **Faktenwissen** *gespeichert wird: „Paris ist die Hauptstadt von Frankreich", „Katzen sind Säugetiere", ...*

FFN besteht aus 2 Schritten:

1. **Aufblasen:** Vektor × W_1 + $Bias_1$ → größerer Vektor, dann **ReLU**
2. **Komprimieren:** Vektor × W_2 + $Bias_2$ → zurück auf Originalgröße

ReLU ist ganz einfach: Alle negativen Zahlen werden zu 0. Alles andere bleibt.

Bei uns: 4D → 8D → 4D (bei GPT-2: 768D → 3072D → 768D, also 4× größer)

Die W_1-Matrix (4×8) — Aufblasen

W_1 hat 4×8 = 32 Zahlen. Das ist viel für Papier! Wir verwenden eine vereinfachte Matrix mit vielen Nullen, damit die Multiplikation machbar bleibt.

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8
d_1	1	0	-1	0	1	0	0	-1
d_2	0	1	0	-1	0	1	0	0
d_3	0	0	1	0	0	0	1	0
d_4	0	0	0	1	0	0	-1	1

$Bias_1$ = [0, 0, 0, 0, 0, 0, 0, 0] (Null für Einfachheit)

Schritt 1: Aufblasen (4D → 8D)

Input nach LayerNorm: **[0.56, 1.33, -1.19, -0.69]**
Jede Ergebnis-Zahl = Summe aus (Input × Spalte von W_1)

$h_1 = 0.56\times1 + 1.33\times0 + (-1.19)\times0 + (-0.69)\times0 = \mathbf{0.56}$

$h_2 = 0.56\times0 + 1.33\times1 + (-1.19)\times0 + (-0.69)\times0 = \mathbf{1.33}$

$h_3 = 0.56\times(-1) + 1.33\times0 + (-1.19)\times1 + (-0.69)\times0 = -0.56 + (-1.19) = \mathbf{-1.75}$

$h_4 = 0.56\times0 + 1.33\times(-1) + (-1.19)\times0 + (-0.69)\times1 = -1.33 + (-0.69) = \mathbf{-2.02}$

$h_5 = 0.56\times1 + 1.33\times0 + (-1.19)\times0 + (-0.69)\times0 = \mathbf{0.56}$

$h_6 = 0.56\times0 + 1.33\times1 + (-1.19)\times0 + (-0.69)\times0 = \mathbf{1.33}$

$h_7 = 0.56\times0 + 1.33\times0 + (-1.19)\times1 + (-0.69)\times(-1) = -1.19 + 0.69 = \mathbf{-0.50}$

$h_8 = 0.56\times(-1) + 1.33\times0 + (-1.19)\times0 + (-0.69)\times1 = -0.56 + (-0.69) = \mathbf{-1.25}$

Ergebnis (8D): [0.56, 1.33, -1.75, -2.02, 0.56, 1.33, -0.50, -1.25]

Schritt 2: ReLU — Negative Werte löschen

ReLU (Rectified Linear Unit): Wenn negativ → wird 0. Wenn positiv → bleibt.
So einfach ist das!

h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8
Vorher							
0.56	**1.33**	-1.75	-2.02	**0.56**	**1.33**	-0.50	-1.25
Nach ReLU							
0.56	**1.33**	0	0	**0.56**	**1.33**	0	0

4 von 8 Neuronen sind „aktiv"! Die anderen sind „aus" (0). Das ist normal — ReLU erzeugt **spärliche Aktivierungen**. Verschiedene Inputs aktivieren verschiedene Neuronen. Man vermutet, dass verschiedene Neuronen verschiedenes Wissen speichern.

Schritt 3: Komprimieren (8D → 4D)

Die W_2-Matrix (8×4):

	d_1	d_2	d_3	d_4
h_1	1	0	0	0
h_2	0	1	0	0
h_3	0	0	0	0
h_4	0	0	0	0
h_5	0	0	1	0
h_6	0	0	0	1
h_7	0	0	0	0
h_8	0	0	0	0

$Bias_2 = [0, 0, 0, 0]$

```
Input (nach ReLU) = [0.56, 1.33, 0, 0, 0.56, 1.33, 0, 0]

d₁ = 0.56×1 + 1.33×0 + 0×0 + 0×0 + 0.56×0 + 1.33×0 + 0×0 + 0×0 = 0.56
d₂ = 0.56×0 + 1.33×1 + 0×0 + 0×0 + 0.56×0 + 1.33×0 + 0×0 + 0×0 = 1.33
d₃ = 0.56×0 + 1.33×0 + 0×0 + 0×0 + 0.56×1 + 1.33×0 + 0×0 + 0×0 = 0.56
d₄ = 0.56×0 + 1.33×0 + 0×0 + 0×0 + 0.56×0 + 1.33×1 + 0×0 + 0×0 = 1.33
```

FFN-Output: [0.56, 1.33, 0.56, 1.33]

Zweite Skip Connection: Add & Norm (nochmal)

LayerNorm-Output:	[	0.56	1.33	-1.19	-0.69]
+					
FFN-Output:	[	0.56	1.33	0.56	1.33]
Summe:	[	**1.12**	**2.66**	**-0.63**	**0.64**]

Zweite Layer Norm:

Vektor = [1.12, 2.66, -0.63, 0.64]

Mittelwert = (1.12 + 2.66 - 0.63 + 0.64) / 4 = 3.79 / 4 = **0.95**

Abweichungen: +0.17, +1.71, -1.58, -0.31

Abweichungen²: 0.03, 2.92, 2.50, 0.10

Varianz = 5.55 / 4 = 1.39

Std = √1.39 = **1.18**

d_1: (1.12 - 0.95) / 1.18 = **+0.14**

d_2: (2.66 - 0.95) / 1.18 = **+1.45**

d_3: (-0.63 - 0.95) / 1.18 = **-1.34**

d_4: (0.64 - 0.95) / 1.18 = **-0.26**

Block-Output für „Katze" = [0.14, 1.45, -1.34, -0.26]

Katzes Reise durch den Block — Vorher vs. Nachher

Station	d_1	d_2	d_3	d_4	Was passiert
Input (nach Pos. Encoding)	0.80	1.40	0.10	1.20	Nur eigene Bedeutung + Position
Attention-Output	1.26	1.15	0.84	0.06	Hat Info von „Die" gesammelt
Add	2.06	2.55	0.94	1.26	Original + Attention-Wissen
Layer Norm	0.56	1.33	-1.19	-0.69	Werte stabilisiert um 0
FFN (8D → ReLU → 4D)	0.56	1.33	0.56	1.33	Allein nachgedacht, Fakten angewendet
Add + Layer Norm	**0.14**	**1.45**	**-1.34**	**-0.26**	**Fertig! → nächster Block**

Erlebnispark: *Das war* **Block 1 von 12***.*

Katze kam rein als [0.80, 1.40, 0.10, 1.20] — ein Token das nur seine Embedding-Bedeutung und seine Position kannte.

Katze geht raus als [0.14, 1.45, -1.34, -0.26] — ein Token das mit „Die" gesprochen hat, weiß dass es ein Subjekt ist, und allein über sich nachgedacht hat.

Jetzt geht derselbe Vektor in **Block 2***. Dort kommt wieder Attention, wieder FFN. In Block 2 kommuniziert Katze mit Versionen von „Die" und „sitzt" die auch Block 1 durchlaufen haben — sie sind also schon schlauer als am Anfang.*

Block für Block wird das Verständnis tiefer.
Frühe Blöcke: Syntax, Nachbarn, Wortarten.
Mittlere Blöcke: Bedeutung, Beziehungen, Referenzen.
Späte Blöcke: Abstrakte Konzepte, Schlussfolgerungen.

Übung: Berechne den Block für „Die" (Position 0)

„Die" sieht mit Maske **nur sich selbst** (100% Attention auf sich). Das vereinfacht die Rechnung enorm!

$Input_{Die}$ = [0.9, 1.1, 0.0, 1.1]

✏️ Selbst rechnen

Attention-Output: (100% Gewicht auf sich selbst)
Head 1: V_{1Die} = [1.1, 1.1] → Output = [___, ___]
Head 2: V_{2Die} = [0.9, 0.0] → Output = [___, ___]
Zusammengeklebt: [___, ___, ___, ___]

Add: [0.9, 1.1, 0.0, 1.1] + [___, ___, ___, ___] = [___, ___, ___, ___]

Layer Norm:
Mittelwert = ___ Std = ___
Normalisiert: [___, ___, ___, ___]

FFN: (verwende W_1 und W_2 von Seite 3-4)
Aufgeblasen (8D): [___, ___, ___, ___, ___, ___, ___, ___]
Nach ReLU: [___, ___, ___, ___, ___, ___, ___, ___]
Komprimiert (4D): [___, ___, ___, ___]

Add + Layer Norm:
$Block\text{-}Output_{Die}$ = [___, ___, ___, ___]

Der komplette Transformer — Alles zusammen

#	Schicht	Was passiert	Papier-Modell
1	**Tokenizer**	Text → Token-IDs	Kärtchen ausschneiden
2	**Embedding**	IDs → Vektoren	In Tabelle nachschlagen
3	**Pos. Encoding**	+ Positions-Signal	Sinus-Werte addieren
× 12 Blöcke	**Multi-Head Attention**	Tokens sprechen miteinander (mit Maske)	Q·K → Softmax → gewichtete Summe V
	Add & Norm	Original addieren, normalisieren	Vektoren addieren, Mittelwert/Std
	FFN	Jedes Token denkt allein nach	Aufblasen → ReLU → Komprimieren
	Add & Norm	FFN-Output addieren, normalisieren	Nochmal addieren + normalisieren
	↑ Das wiederholt sich 12× (GPT-2) oder 96× (GPT-4) ↑		
4	**Output-Schicht**	Letzter Vektor → Wahrscheinlichkeit für nächstes Token	Noch ein Softmax (kommt als Nächstes!)

Was du aus Papier gebaut hast:

Einen kompletten Transformer-Block. Ein GPT-2 Small besteht aus 12 dieser Blöcke hintereinander — der Output des einen ist der Input des nächsten.

Parameterzählung unseres Papier-Modells:

- Embedding-Tabelle: 6 × 4 = 24 Parameter
- W_Q, W_K, W_V (pro Head): 3 × 4×2 = 24 pro Head, × 2 Heads = 48
- W_1 (FFN): 4 × 8 = 32
- W_2 (FFN): 8 × 4 = 32
- **Gesamt: ~136 Parameter**

GPT-2 Small: 124 Millionen Parameter.
GPT-4: geschätzt ~1.800 Milliarden Parameter.

Tokenizer ✓ → Embedding ✓ → Pos. Enc. ✓ → MH-Attention ✓ → Add & Norm ✓ → FFN ✓ → **Output-Schicht**

Erlebnispark — Fast am Ziel: *Es fehlt nur noch der* ***letzte Schritt****: Wie wird aus dem Vektor des letzten Tokens eine* ***Vorhersage für das nächste Wort****?*

Dafür wird der Vektor von „Matte" (dem letzten sichtbaren Token) in eine riesige Softmax-Schicht geschickt, die für jedes mögliche Token im Vokabular eine Wahrscheinlichkeit berechnet. Das Token mit der höchsten Wahrscheinlichkeit wird gewählt.

Und dann beginnt der ganze Prozess von vorne — mit dem neuen Token angehängt.