A 456-Parameter Transformer Solves 10-Digit Addition

February 2026

Code: `https://github.com/yinglunz/A-456-Parameter-Transformer-Solves-10-Digit-Addition`

1 A 456-Parameter Model for 10-Digit Addition

We present a **456-parameter** transformer that solves 10-digit integer addition. Given two integers $A, B \in [0, 10^{10})$, the model predicts $C = A + B$ autoregressively, achieving **100%** exact-match accuracy on 100,000 test examples.

Building on `smallest-addition-transformer-codex` [1] (1,644 params) and `gpt-acc-jax` [2] (777 params), we introduce **low-rank factorization** ($W = AB$, rank 3) to reduce the model to 512 parameters. [3] then replaced LayerNorm with **RMSNorm without bias** to reach 491 parameters. We build on this and add two further techniques to reach **456 parameters**:

- **Shared-A tied-KV QKV**: in the low-rank QKV factorization ($Q, K, V = hB_q, hB_k, hB_v$ where $h = xA$), we tie $B_k = B_v$ so that $K = V$, reducing from 3 B-matrices to 2 (saves 21 parameters).
- **Rank-2 attention output**: attention output projection reduced from rank 3 to rank 2 (saves 14 parameters).

1.1 Architecture

The model is a single-layer, single-head, decoder-only transformer with:

- $d_{\text{model}} = 7$, $d_{\text{ff}} = 14$, vocabulary size 14 (digits 0–9, `+`, `=`, `<PAD>`, `<EOS>`)
- Input: zero-padded operands (e.g., `0000000005+0000000007=`), 22 prompt tokens + 11 reversed target digits
- Learned position embeddings, low-rank factorized (rank 3)
- RMSNorm (weight only, no bias) before attention, FFN, and at the final layer
- QKV projection: shared-A with tied K=V B-matrix (`shareA_tieKV`), rank 3
- Attention output projection: low-rank, rank 2
- FFN up/down projections: low-rank, rank 3
- Weight-tied input embedding and output head
- No bias, no dropout

The per-component parameter breakdown is given in Table 1, and the full reduction journey in Table 4.

Table 1: Parameter breakdown of the 456-parameter model.

Component	**Factorization**	**Params**
Token embedding (tied)	14×7	98
Position embedding	$33{\times}3 + 3{\times}7$	120
RMSNorm (pre-attn)	weight only	7
QKV (shareA_tieKV, r=3)	A: 7×3; B_q: 3×7; B_{kv}: 3×7	63
Attention output (r=2)	$7{\times}2 + 2{\times}7$	28
RMSNorm (pre-FFN)	weight only	7
FFN up (r=3)	$7{\times}3 + 3{\times}14$	63
FFN down (r=3)	$14{\times}3 + 3{\times}7$	63
Final RMSNorm	weight only	7
Output head	(tied with token embedding)	0
Total		**456**

1.2 Training

We use a 3-phase curriculum following `gpt-acc-jax` [2]:

1. Phase 1 (steps 0–2,000): operands with 1–3 digits
2. Phase 2 (steps 2,000–7,000): operands with 1–6 digits
3. Phase 3 (steps 7,000–54,000): operands with 1–10 digits (full range)

Optimizer: AdamW, peak LR = 0.02, linear warmup (1,350 steps) + cosine decay, min LR = 0.002, weight decay = 0.01, gradient clipping = 1.0, batch size = 512, total steps = 54,000.

1.3 Grokking

All successful models exhibit *grokking*: prolonged near-zero validation accuracy followed by a sudden phase transition. Figure 1 shows the validation exact-match curves for the 582-parameter, 512-parameter, and 456-parameter models across seeds. For the 582-parameter and 512-parameter models, the default seed 42 groks reliably within 27K steps, though grokking behavior is inherently stochastic and may vary across hardware and software configurations. For the 456-parameter model, which operates at a tighter parameter budget, 2 out of 5 random seeds grokked (seeds 43 and 44) within 54K steps. Seed 43 began grokking at step ∼26,500 and first reached 100% at step 34,000. Seed 44 began grokking at step ∼22,000 and first reached 100% at step 37,000.

1.4 Evaluation

During training, models are validated on a fixed held-out set of 5,000 examples (used for checkpoint selection and the grokking curves in Figure 1). For final evaluation, we use a separate, stricter protocol: 10 independent test sets of 10,000 examples each (100,000 total), generated with random seeds disjoint from both training and validation. Table 2 summarizes the aggregate results across all three model sizes. The per-test-set breakdown for the 456-parameter model is shown in Table 3.

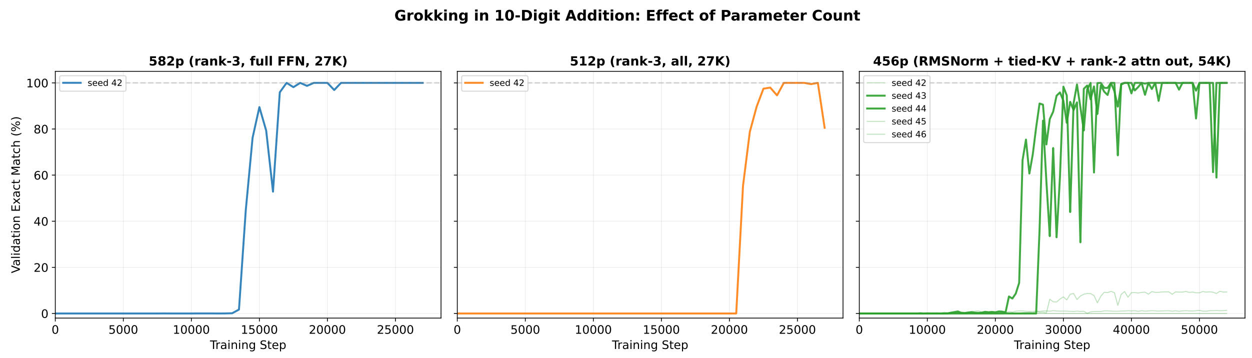

Figure 1: Validation exact-match accuracy during training. The 582p and 512p panels show seed 42 (which grokked); the 456p panel shows all 5 seeds, with grokked seeds (43, 44) drawn in bold. Grokking occurs later for smaller models.

Table 2: Aggregate evaluation results across model sizes (10 test sets × 10,000 examples = 100,000 total per model).

Model	Params	Exact Match	Errors / 100K
582p (seed 42)	582	99.999%	1
512p (seed 42)	512	99.988%	12
456p (seed 43)	456	100%	0
456p (seed 44)	456	99.958%	42

2 A Journey Towards 456 Parameters

The starting point is `gpt-acc-jax` [2]: a 777-parameter transformer (d=7, d_{ff}=14, 1 layer, 1 head, tied embeddings, no bias) that achieves 99.69% accuracy on 10-digit addition. Our reimplementation uses a shorter sequence length (n_{ctx}=33 vs. 35), giving a 763-parameter full-rank baseline.

2.1 763 → 512: Low-Rank Factorization

Integer addition is an inherently low-rank algorithm: it operates column-by-column with a 1-bit carry, requiring only a small number of basis functions. We exploit this by replacing every weight matrix $W \in \mathbb{R}^{m\times n}$ with a factorized product $W = AB$ where $A \in \mathbb{R}^{m\times r}$ and $B \in \mathbb{R}^{r\times n}$, reducing parameters from mn to $r(m + n)$.

We apply rank-3 factorization to all four major components:

- **Position embeddings** ($33\times7 \to 33\times3 + 3\times7$): 231 → 120, saves 111
- **QKV projection** ($7\times21 \to 7\times3 + 3\times21$): 147 → 84, saves 63
- **Attention output** ($7\times7 \to 7\times3 + 3\times7$): 49 → 42, saves 7
- **FFN up/down** ($7\times14 + 14\times7 \to$ rank-3 each): 196 → 126, saves 70

Total savings: 251 parameters. Result: **512 parameters**. The 512-parameter model achieves 100% validation accuracy (seed 42, grokking at step ∼21K). Beyond compression, the low-rank constraint may also serve as a regularizer, restricting the model to a low-dimensional manifold that favors algorithmic solutions over memorization. The parameter breakdown is compared in Table 4.

Table 3: Per-test-set evaluation results for the 456-parameter model (10 test sets $\times$ 10,000 examples = 100,000 total).

	Seed 43		**Seed 44**	
Test Seed	**Exact Match**	**Errors**	**Exact Match**	**Errors**
41	100%	0	100%	0
100	100%	0	99.97%	3
200	100%	0	99.97%	3
300	100%	0	99.92%	8
400	100%	0	99.98%	2
500	100%	0	99.94%	6
999	100%	0	99.93%	7
1234	100%	0	99.95%	5
7777	100%	0	99.95%	5
31415	100%	0	99.97%	3
Aggregate	**100%**	**0**	**99.958%**	**42**

Table 4: Parameter breakdown across the reduction journey. "LR-r" denotes low-rank with rank r; "LN" = LayerNorm; "RMS" = RMSNorm; "sA-tKV" = shareA_tieKV.

Component	**763 (full)**	**512 (LR-3)**	**491 (+ RMS)**	**456 (+ sA-tKV)**
Token emb. (tied)	98	98	98	98
Position emb.	231	120	120	120
Norm (pre-attn)	14 (LN)	14 (LN)	7 (RMS)	7 (RMS)
QKV projection	147	84	84	63 (sA-tKV)
Attention output	49	42	42	28 (LR-2)
Norm (pre-FFN)	14 (LN)	14 (LN)	7 (RMS)	7 (RMS)
FFN up	98	63	63	63
FFN down	98	63	63	63
Final norm	14 (LN)	14 (LN)	7 (RMS)	7 (RMS)
Output head	0	0	0	0
Total	**763**	**512**	**491**	**456**

2.2 512 → 491: RMSNorm

Building on the 512-parameter model, [3] replaced LayerNorm with RMSNorm. LayerNorm uses both a weight vector and a bias vector, costing $2 \times d_{\text{model}}$ parameters per instance. RMSNorm removes the bias and mean-centering, using only a weight vector:

$$\text{RMSNorm}(x) = \frac{x}{\sqrt{\text{mean}(x^2) + \epsilon}} \cdot w \tag{1}$$

This costs only d_{model} parameters per instance. With 3 normalization layers (pre-attention, pre-FFN, final), this saves $3 \times 7 = 21$ bias parameters (see Table 4, column 3 vs. 4). Result: $512 - 21 =$ **491 parameters**. We build upon this work to construct the 456-parameter model described next.

2.3 491 → 456: Shared-A Tied-KV QKV + Rank-2 Attention Output

Two further changes reduce the model from 491 to 456 parameters (see Table 4, column 4 vs. 5).

1. QKV with shared A and tied K=V (`shareA_tieKV`). In the 512/491-parameter model, the low-rank QKV projection factors $W_{\text{QKV}} = A \cdot [B_q; B_k; B_v]$ with a shared $A \in \mathbb{R}^{7\times3}$ and three separate B matrices $B_q, B_k, B_v \in \mathbb{R}^{3\times7}$ (84 parameters total). We observe that for autoregressive addition, the key and value representations serve a similar role: both encode positional digit information for the attention mechanism to retrieve. We therefore tie $B_k = B_v$ into a single B_{kv}:

$$h = xA, \quad Q = hB_q, \quad K = V = hB_{kv} \tag{2}$$

This reduces the QKV parameters from $7\times3 + 3\times3\times7 = 84$ to $7\times3 + 2\times3\times7 = 63$, saving **21 parameters**.

2. Rank-2 attention output projection. The attention output projection $W_O \in \mathbb{R}^{7\times7}$ is reduced from rank 3 to rank 2:

$$\text{Params: } 7\times2 + 2\times7 = 28 \quad (\text{vs. 42 at rank 3, saves } \mathbf{14}) \tag{3}$$

We found that, for this simple task, the transformer is mostly robust to compression of the attention output matrix—reducing it to rank 2 causes no degradation. In contrast, reducing other weight matrices (e.g., QKV, FFN) to rank 2 leads to training failure.

Combined savings: $21 + 14 = 35$. Result: $491 - 35 =$ **456 parameters**. The grokking curves for this final model are shown in Figure 1, and evaluation results in Table 3.

3 Reproducibility

3.1 Environment

- PyTorch 2.10.0 with CUDA
- Single GPU (NVIDIA RTX PRO 6000 Blackwell)

3.2 Reproduce the 456-Parameter Model

```
python -m src.train \
  --run-name best_456 \
  --pos-rank 3 --qkv-rank 3 --attn-out-rank 2 --ffn-rank 3 \
  --use-rmsnorm --tie-qkv shareA_tieKV \
  --total-steps 54000 --device cuda --seed 43
```

3.3 Evaluate a Checkpoint

```
python evaluate_checkpoints.py \
  checkpoints/best_456p_s43.pt --device cuda
```

References

[1] D. Papailiopoulos, "Glove box challenge: smallest transformer for 10-digit addition," 2026. `https://github.com/anadim/smallest-addition-transformer-codex`

[2] Y. Havinga, "gpt-acc-jax: Smallest GPT for 10-digit addition," 2026. `https://github.com/yhavinga/gpt-acc-jax`

[3] rezabyt, "digit-addition-491p," 2026. `https://github.com/rezabyt/digit-addition-491p`