Learning Addition with a Small Transformer

A. Task setup (format, k, model, tokenizer)

The goal is to train a small transformer model to perform addition of two k=3 digits. Given an input like "123 + 456 = ", the model should produce "579" per the stated task goal. We later evaluate whether a trained model can generalize to longer inputs (k=4 and k=5 digits).

The training data format uses the following structure with " = " denoting the end of the input and "\n" denoting the end of the output:

- Input: "<a> + <b> = "
- Output: "<sum>\n"
- Full sequence: "123 + 456 = 579\n"

Character-level tokenization was chosen for this task with a custom vocabulary consisting of the only 14 tokens necessary: chars = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '+', ' ', '=', '\n']

Each digit is treated as a separate token, bypassing tokenizing problems where numbers consisting of multiple digits can be tokenized mid-way (e.g. '1159' into '115' and '9' making decimal level addition difficult). The small custom vocabulary creates a pretty compact embedding layer that should be more quickly trained.

A GPT-2 style decoder-only transformer was used. This choice follows from recent work studying arithmetic in language models, where decoder-only transformers have become the standard architecture for investigating what transformers can and cannot learn about algorithmic tasks [1][2]. The following configuration was used:

```
model_config = GPT2Config(
    vocab_size=14,
    n_positions=25,
    n_layer=6,
    n_head=8,
    n_embd=256,
    embd_pdrop=0.0,
    attn_pdrop=0.0,
    resid_pdrop=0.0
)
```

B. Data design (generation, splits, diagnostics)

A preliminary approach would be to sample our two numbers a and b uniformly from [100, 999] for k=3 digit addition. However, this creates an imbalanced distribution of problem difficulty, using the number of carry operations as a correlate for difficulty level. Under uniform sampling, problems with 1 or 2 carries outnumber those with 0 or 3 carries, leading to a model that may

learn to excel at these cases but not handle outlier carry behavior well.

To address this, I implemented a carry-balanced sampling strategy. For k=3 digit numbers, I ensured equal representation across all carry counts (0, 1, 2, 3). The generation works by:

1) For a given carry count c, randomly select c digit positions where the carries will occur
2) For each digit position, sample digit pairs (a_i, b_i) that produce the required carry behavior (binary carry/no-carry)
3) Enforce leading digits to be non-zero to train on exclusively k-digit numbers
4) Combine digits to form the final numbers a and b

Data is generated on-the-fly during both training and evaluation using the same process that balances carry counts based on the batch size provided.

C. Training details (hyperparameters, runtime, compute)

Hyperparameter	**Value**
Batch Size	128
Training Steps	10,000
Learning Rate	1e-4
Optimizer	AdamW(beta_1=0.9, beta_2=0.95)
Weight Decay	0.01

Training completed in approximately 7 minutes on a T4 GPU in Google Colab. The loss curve showed smooth convergence:

Step 100: Loss = 1.29
Step 1,000: Loss = 0.07, Accuracy = 93%
Step 2,000: Loss = 0.05, Accuracy = 99%
Step 5,000: Loss = 0.01, Accuracy = 100%
Step 10,000: Loss = 0.0001, Accuracy = 100%

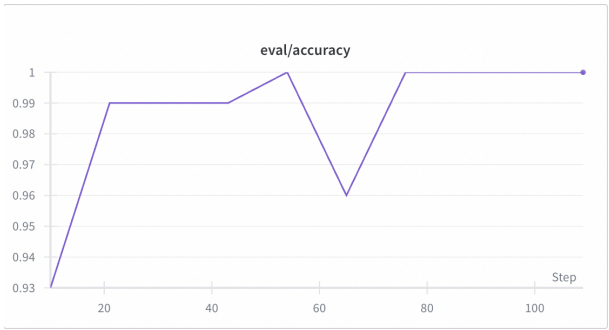
The model converged much faster than the set 60 minutes, making me think that this process for learning 3-digit addition is quite easily learned by this architecture.

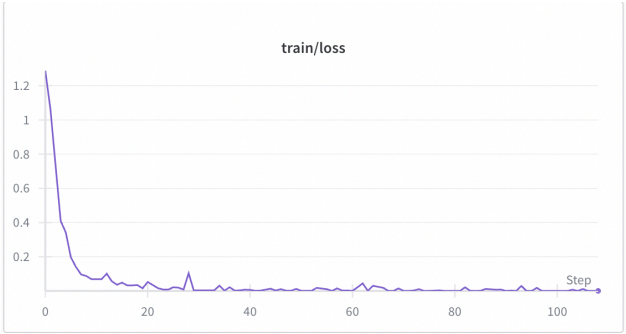
D. Evaluation and results (tests + motivation, tables/plots)

The evaluation answers the question of performance on in-distribution (k=3) sets and tests for length generalization on k=4,5 digit problems to test whether the algorithm just learned some specific 3-digit patterns or if it learned how to actually add two numbers. The below table shows the accuracy across different digit scenarios tested on 1000 evenly carry-distributed samples.

Accuracy is measured as exact-match: the model's generated output must exactly equal the correct sum, character by character, otherwise no credit is given. This is stricter than the cross-entropy loss used during training which is computed per-token, so a model that predicts the correct digit with 90% probability still incurs some loss, and getting 3 out of 4 digits right yields a relatively low loss. This distinction could matter as a model could achieve low loss while still making occasional errors and not having high accuracy. The training logs don't reflect this mismatch though and we see accuracy increasing with loss dropping. Additionally, greedy decoding is used during evaluation since addition has a single correct answer.

Evaluation	**Accuracy**
In-distribution (k=3)	100%
Out-of-distribution (k=4)	0%
Out-of-distribution (k=5)	0%





E. Interpretation and Discussion

The contrast between performance on in-distribution data and out-of-distribution data is quite stark, indicating that the model did not learn the general addition algorithm. Instead, it likely learned some 3-digit position-specific patterns, essentially memorizing a lookup table for 3-digit addition rather than actual arithmetic rules.

A part of what makes this hard is that addition naturally proceeds right-to-left (starting from the least significant digit and propagating carries upward), but transformers generate left-to-right. The model has to somehow plan for carries it hasn't computed yet, which may explain why it tries to fall back on memorization. A main algorithmic change I would like to try is reversing the order of the digits, to essentially capture a digit's "importance" with respect to carry logic and to mimic how we naturally approach addition.

Another simple tweak that should improve this algorithm is to train on mixed-length data with k ranging from 1-5 and then testing on k=6,7. This should expose the model to a larger variety of examples hopefully allowing it to generalize more and not depend on positional patterns as much.

References

[1] Baeumel, T., Gurgurov, D., al Ghussin, Y., van Genabith, J., & Ostermann, S. (2025). Modular Arithmetic: Language Models Solve Math Digit by Digit. arXiv:2508.02513

[2] Lee, N., Sreenivasan, K., Lee, J. D., Lee, K., & Papailiopoulos, D. (2023). Teaching Arithmetic to Small Transformers. arXiv:2307.03381