

Interpreting Character-Level Transformers for Addition and Multiplication Through Integrated Gradients

Abhinav Damera

Northern Burlington County Regional High School

New Jersey, USA

27damera@nburlington.com

Abstract

Transformer models have demonstrated strong performance across a variety of reasoning tasks. However, understanding how these models perform arithmetic reasoning remains an open challenge. This study focuses on investigating the factors that influence arithmetic performance and the interpretability of character-level transformer models. These models were trained on increasingly complex tasks including one-digit arithmetic, two-digit arithmetic, and natural language prompts containing arithmetic expressions embedded within them. Model performance was evaluated by examining the effects of altering training iterations, dataset size, and model configurations. In order to account for training variability, representative two-digit configurations were trained using five independent random seeds and evaluated using a standardized prompt-level procedure. In order to perform the interpretability analysis, Integrated Gradients was utilized to calculate token-level attribution for each input token across zero embedding and mean embedding baselines. The results show that model performance generally improved as training iterations, dataset size, and model configurations increased. Additionally, expanding the character vocabulary allowed the model to process natural language prompts containing arithmetic expressions while generally assigning lower attribution scores to contextual letters and whitespace than for arithmetic tokens. The findings show that the character-level transformer models achieved high accuracy on arithmetic expressions embedded within natural language, while attribution scores were generally higher for arithmetic tokens than for contextual letters and whitespace. This study examines factors associated with the performance of character-level transformer models and provides interpretability evidence of the internal features that contribute to model predictions.

Keywords

Transformer Models, Arithmetic Reasoning, Integrated Gradients, Explainable Artificial Intelligence, Interpretability

ACM Reference Format:

Abhinav Damera. 2026. Interpreting Character-Level Transformers for Addition and Multiplication Through Integrated Gradients. In *Proceedings of International Journal of Secondary Computing and Applications Research (IJSCAR VOL. 3, ISSUE 4)*. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.67149/yhjs2024.5/88bi6k5z>

This paper is published under the Creative Commons Attribution-NonCommercial-NoDerivs 4.0 International (CC-BY-NC-ND 4.0) license. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution.

IJSCAR VOL. 3, ISSUE 4

© 2026 The Author(s). Published by IJSCAR.

1 Introduction

Transformer-based models have become crucial in artificial intelligence in the past decade [2, 10, 15]. These machine learning models have the ability to solve simple tasks such as language generation or even more complex operations involving deep learning and reasoning abilities [2, 10, 16]. Although they are effective, these models often operate as black boxes, making it hard for users to understand what is truly happening behind the scenes. It has become increasingly important to interpret the reasoning of these models because this lack of transparency poses a challenge when attempting to determine which specific parts of an input contribute most to model outputs. Additionally, increased transparency may help determine whether successful predictions are due to meaningful representations or other patterns that may form during model training.

As artificial intelligence becomes increasingly embedded within everyday life, understanding the specific reasoning behind model predictions has become a significant problem. Interpretability methods are useful for addressing this issue, as they allow users to investigate the level of influence each feature and input has on the model's final prediction [8, 11, 13]. These methods can improve transparency and trust in the system by providing insight into the model behavior. Specifically, this is significant when models produce correct or incorrect outputs and do not provide explicit explanations demonstrating how the predictions were made. Token-level attribution shows the influence of each specific input token in an input sequence on model outputs.

Studying interpretability on large-scale language models is difficult due to their architectural complexity, size, and the wide range of information contained within natural language prompts. Arithmetic problems could be used to test interpretability as they provide more controlled environments. Furthermore, utilizing arithmetic can be used to quantitatively and objectively understand model behavior by examining the model's output. This allows us to distinguish the impact of different pieces of each prompt. Beginning with single digit arithmetic and progressively increasing task complexity provides a controlled framework for evaluation and allows us to understand how the model performs against increasingly difficult tasks. Furthermore, embedding arithmetic within natural language prompts can be useful in determining how the model identifies relevant numerical information among the unrelated textual context.

Previous studies have focused on transformer reasoning and interpretability methods for understanding neural network predictions. However, little research has been done on token-level attribution in controlled arithmetic tasks while evaluating how training conditions and model architecture affect numerical predictions. By combining quantitative performance evaluation with

token-level attribution, researchers can understand how models produce successful predictions and what input portions influence those predictions.

This study focuses on small character-level transformer models rather than large-scale language models. Therefore, the results should be interpreted as evidence about transformer behavior in this controlled setting rather than as a direct characterization of arithmetic reasoning in frontier-scale language models. The models are trained on one-digit and two-digit arithmetic operations, allowing individual variables and tokens to be examined systematically, and are evaluated quantitatively. The models are evaluated on the number of training iterations, dataset size, and model capacity. Finally, the model's reasoning behavior is evaluated through Integrated Gradients. The results show how transformer architecture influences performance and provide insight into input features associated with predictions.

2 Related Work

Modern machine learning models have increasingly relied on transformer-based architectures due to their ability to use self-attention mechanisms, allowing each individual token to use information from other tokens in the input sequence, rather than recurrent computations [2, 10, 15]. The attention operation computes weighted combinations of value vectors based on query-key similarity:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

where Q represents the query, K represents the key, and V represents the value. Subsequent autoregressive transformer models further demonstrate that large language models can perform increasingly sophisticated reasoning tasks, including arithmetic through next-token prediction [2, 10, 16].

Recent research has focused on understanding the ability of transformer models to perform arithmetic reasoning. These works suggest that language models are able to learn numerical patterns from training data even if they were not explicitly designed for symbolic computation [9, 12, 14]. However, these studies often lack insight into the mechanisms that lead to success in transformer models when completing operations as they primarily focus on predictive performance.

A major challenge as transformer models become more complex is understanding their reasoning for each output they produce. Explainable AI methods such as LIME and SHAP aim to address this issue by identifying features and inputs that influence model predictions [8, 11]. These interpretability techniques focus on utilizing local approximations and feature contribution analysis. Integrated Gradients, a gradient based attribution method, has become widely adopted because it satisfies properties such as Sensitivity and Implementation Invariance [13]. Integrated Gradients creates attribution scores for each input token, making it a useful attribution-based method as it does not rely fully on a single token. Although Integrated Gradients has become widely used, few studies have focused on utilizing it to understand arithmetic reasoning behavior in transformer-based language models.

Recent studies have examined attention distributions, neuron activations, and feature attribution techniques to better understand

transformer reasoning [1, 3, 4]. This study focuses on addressing these gaps by using quantitative evaluation methods on increasingly difficult arithmetic tasks and then using Integrated Gradients to perform an interpretability analysis on the final natural language processing model to demonstrate how token-level attribution relates to arithmetic reasoning.

3 Methodology

During model training and evaluation, datasets were distributed with an equal number of multiplication and addition expressions. Additionally, positive numbers were used throughout all experiments. In order to investigate arithmetic reasoning in transformer models, three datasets were created that progressively became more complex. The first dataset consisted of one-digit multiplication and addition expressions, ranging from 0 to 9, to evaluate whether a character-level transformer could learn basic arithmetic relationships. The second dataset consisted of two-digit multiplication and addition expressions, ranging from 10 to 99. In order to successfully complete two-digit operations, the model had to learn more nuanced relationships between numerical values. These nuances include carrying operations in addition and multi-digit computation in multiplication. The increased numerical complexity allowed for the evaluation of the model's ability to scale beyond simple arithmetic tasks. The last dataset further increased in complexity. The numerical values still ranged from 10 to 99 similar to the second dataset. However, this dataset included natural language embedded with arithmetic operations. The natural language prompts were originally constructed manually and tested one by one. Eventually, three different 1,000-prompt test sets (one for the one-digit model, one for the two-digit model, and one for the natural language model) were generated separately and used to evaluate each model's performance. These evaluation prompts were kept fixed across repeated runs of the same configuration so that models trained with different random seeds were evaluated under identical conditions. This helped solidify our findings by expanding the range of linguistic contexts and reducing dependency on manually selected examples. The natural language characters served as irrelevant linguistic context to test whether the model would still perform arithmetic operations while ignoring unrelated information.

Each dataset was generated using custom Python scripts. Each of them was converted into the binary training format required by nanoGPT. Separate training and validation splits were prepared for each dataset using a 90/10 split. The model was trained on the training portion while it used the validation portion to monitor model performance during training. After training, the model was evaluated using the fixed 1,000-prompt test set to measure its performance.

The evaluation prompts consisted of the same numerical ranges as the corresponding training datasets. Specific operand combinations were not deliberately held out from the training data. Rather, the evaluation prompts were generated separately from the same numerical ranges.

The model used character-level tokenization where each individual character (letters, digits, operators, punctuation) was represented by a unique token. The model produces the probability distribution of the next token given the preceding sequence:

$$P(x_t | x_1, x_2, \dots, x_{t-1}) = \text{softmax}(Wh_t + b) \quad (2)$$

where x_t represents the predicted token, h_t is the hidden representation, while W and b are output parameters. This allowed the model to perform arithmetic operations even with natural language prompts embedded under one unified vocabulary. Transformer configurations were modified in order to test model scaling. Each experiment varied in the number of transformer layers, attention heads, and embedding dimensions. However, the model maintained its overall architecture throughout all experiments. A checkpoint was saved for each experiment to allow consistent comparison across models.

Models were trained independently on each dataset using supervised character prediction. Cross-entropy loss was minimized between the predicted and target token distributions:

$$L = -\frac{1}{N} \sum_{i=1}^N y_i \log(\hat{y}_i) \quad (3)$$

where y_i represents the target token distribution and $\hat{y}_i$ represents the predicted probability distribution. The AdamW optimizer was used during training with learning rate warmup and cosine learning rate decay following the nanoGPT framework [7]. Unless specified by the experiment, models were trained for 20,000 iterations using a batch size of 64, a block size of 32, a learning rate of 1×10^{-3} , and a dropout rate of 0.2. During the model scaling experiment, the model depth, number of attention heads, and embedding dimension were varied while the remaining model parameters were held constant. Training and validation loss were observed during each experiment. Larger model configurations were tested in order to understand whether increasing model capacity improves arithmetic performance. The final and most advanced model, consisting of natural language, used the largest architecture and was used to perform the interpretability analysis.

The performance of each model was evaluated quantitatively using prompt-based inference. The evaluation involved introducing a fixed set of arithmetic expressions to each model based on the model's configurations. Arithmetic problems in the test set did not contain the answers to each problem. The model's predictions were compared against the correct arithmetic answers. The number of correctly solved problems was divided by the total number of prompts in the test set to produce the accuracy. Every prompt contributed to the model's accuracy. The same procedure was utilized for all experiments and random seeds.

The natural language model also focused on solving arithmetic expressions correctly while the prompts involved distractor letters and numbers. Multiple prompt styles were tested with the model, including both two-digit multiplication and addition, so that robustness across different arithmetic tasks and sentence contexts could be tested. The model was evaluated using the same prompt-level procedure.

To account for variability of random initialization and training, representative configurations were trained using five independent random seeds. The accuracy for these configurations is presented as the mean and standard deviation across each of the five runs. The repeated seed evaluation was used for the two-digit model when testing the 5,000 example and 20,000 example dataset as well as the

expanded two-digit model that trained on the same 20,000 example dataset. The natural language model utilized the expanded model architecture and the same 20,000 example dataset. This model and the subsequent Integrated Gradients analysis was conducted using a single trained model rather than multiple random seeds.

In order to understand which input tokens were associated with the model's predictions, Integrated Gradients (IG) from the Captum library [6] was utilized to complete interpretability experiments:

$$IG_i(x) = (x_i - x'_i) \int_0^1 \frac{\partial F(x' + \alpha(x - x'))}{\partial x_i} d\alpha \quad (4)$$

where x represents the original input embeddings, x' represents the baseline embeddings, $F(x)$ represents the logit corresponding to the model's predicted next token, and i is the input token being analyzed. Integrated Gradients computes attribution scores by accumulating gradients along a continuous path between a baseline input and the observed input. During the interpretability analysis, 100 integration steps were used to compute attribution scores for all 1,000 prompts in the natural language test set. Two different baseline configurations were utilized: a zero embedding baseline and a mean embedding baseline. For each prompt, the target was the model's highest-probability next-token prediction at the final input position. A single attribution value was obtained for each input token by summing attribution values across the embedding dimensions. The resulting scores were aggregated by token category: digits, operators, letters, punctuation, and whitespace. This was done to examine whether the observed attribution patterns were consistent across the evaluation set. Larger positive attribution scores suggest a greater influence on the overall prediction. The resulting attribution scores were used to determine whether the transformer mostly relied on the arithmetic tokens (digits and operators) or whether the unrelated natural language influenced predictions more heavily. This analysis provided an additional insight on the input features associated with the model's predictions beyond traditional prediction accuracy metrics.

4 Experimental Evaluation

The experimental evaluation was structured such that each experiment progressively increased in complexity. We first establish a baseline for the arithmetic performance before testing the effect of training iterations, numerical complexity, dataset size, model capacity, and additional contextual information. In the end, the interpretability analysis is performed where we analyze the token-level attribution of the model by utilizing Integrated Gradients. In all, these experiments not only test the effect of increasing complexity on the model's output, but also identify whether relevant information influences predictions and to what extent the linguistic context impacts these predictions.

4.1 Experiment 1: Baseline Arithmetic Evaluation

The first experiments were created to test whether transformer-based models could learn to perform arithmetic operations by learning arithmetic relationships through supervised training. A synthetic dataset, containing around 20,000 examples including one-digit arithmetic multiplication and addition expressions, was

created using a custom Python script. Additionally, these examples contained the actual answers to the problems for model training while the prompts did not contain correct outputs during evaluation. There was an equal amount of addition and multiplication examples. The model was set up with 1 layer, 1 attention head, an embedding dimension of 120, dropout rate of 0.2, batch size of 64, and a block size of 32. The training hardware which was used was a CPU.

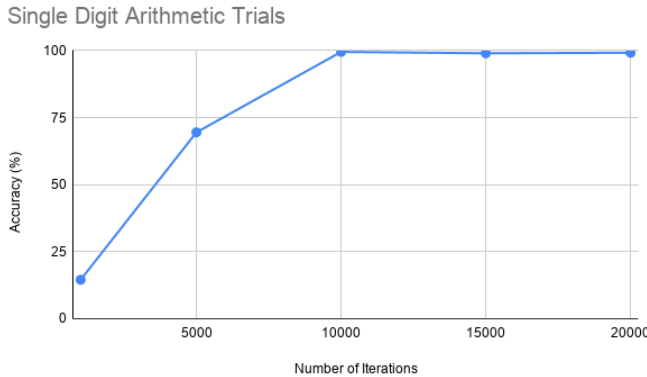


Figure 1: Accuracy improvement of the single-digit baseline arithmetic model with increasing training iterations.

Table 1: Single-digit baseline arithmetic model performance across training iterations.

Trial	Training Iterations	Accuracy (%)
1	1,000	14.51
2	5,000	69.50
3	10,000	99.50
4	15,000	99.00
5	20,000	99.20

The effect of increasing training iterations on model accuracy is shown in Figure 1 and Table 1. The results in Table 1 are from a single training run and therefore do not quantify variability across random seeds. The model was trained using supervised next-character prediction. The objective of the model was to predict the next character in an arithmetic expression. The model's performance was evaluated using arithmetic prompts to verify that the network generated the right numerical outputs. Addition and multiplication problems were evaluated together. These initial experiments established the proof of concept that the model was capable of learning simple arithmetic operations. It allowed us to see whether changes to the model had to be made to allow for it to be able to process arithmetic prompts before moving onto the more rigorous evaluations.

The initial model was adjusted to test the effect of altering the number of training iterations to see if that led to improved accuracy. Model checkpoints were evaluated throughout the training process to determine how the additional optimization impacted

performance. Due to the usage of a CPU over a GPU, training took longer than expected. We noticed as we increased the number of training iterations, the training time substantially increased. Specifically, when the number of iterations was set to 1,000, the training time was less than 15 minutes. However, when the number of iterations was set to 20,000, the training time increased to over 2.5 hours. Accuracy generally improved as the number of training iterations increased, rising from 14.51% at 1,000 iterations to 99.20% at 20,000 iterations. The 1,000 prompt test set contained 199 unique arithmetic expressions. All of the unique prompts appeared in the training portion which means that the evaluation set did not assess generalization to unseen one-digit operand-operator combinations. Therefore, these results should be interpreted as performance on problems represented in the training data. Furthermore, the results are from a single run.

The model performance seemed to improve nonlinearly as the number of training iterations increased. Although increasing the number of training iterations did lead to improved performance, the model's performance seemed to rapidly increase from 1,000 to 10,000 iterations and plateaued once it reached around 10,000 training iterations. This suggests that the model had already learned the necessary arithmetic relationships to perform single digit arithmetic operations (see Figure 1 and Table 1). The final model's configurations were used as the baseline for the second experiment. Additionally, 20,000 training iterations were used for subsequent experiments in order to allow each model to have enough training before evaluation.

4.2 Experiment 2: Two-Digit Arithmetic Evaluation

After the baseline model was evaluated, the model was expanded to include two-digit arithmetic multiplication and addition operations in order to determine whether the transformer could perform more complex arithmetic relationships. The two-digit arithmetic model contained most of the same parameters as the single digit model. However, the two-digit model contained 4 layers, 4 attention heads, and an embedding dimension of 128 as preliminary results showed that the model was initially unable to perform arithmetic calculations. Additionally, the two-digit model was trained on 20,000 iterations.

Initially, the two-digit arithmetic dataset only contained 5,000 examples. Similar to the one-digit dataset, only numerical prompts were included. However, the dataset included values from 10 to 99. Single digit numbers were not included in the new dataset. Even though combining one-digit and two-digit examples into one dataset would still have increased complexity, this would introduce different levels of complexity within one dataset making it hard to determine how the model responds to two-digit prompts specifically. By restricting the dataset to only two-digit, we can strictly analyze the model's ability to learn more complex numerical relationships associated with two-digit operations. Specifically, when performing addition operations, the model had to consider carrying between the ones and tens positions. For multiplication, the model had to consider the relationship between multiple digits. The increased numerical complexity was associated with the decrease in performance by the original model configuration.

Each model was trained against five independent random seeds in order to account for the variation caused by random initialization and was evaluated using the standardized prompt-level evaluation procedure. By training on the 5,000 example dataset, the model produced a mean accuracy of 68.74% with a standard deviation of 1.16 percentage points (see Table 2). The relatively small standard deviation indicated that model results were similar across the five random seeds. After the model trained on the original 5,000 example dataset, the performance of the two-digit arithmetic model was noticeably lower compared to the performance of the one-digit arithmetic model. We were initially reluctant to assume that the dip in accuracy was purely due to the increased complexity of the numerical tasks since the initial two-digit dataset was smaller compared to the one-digit dataset. This presented a potential confounding factor that could have affected the model’s accuracy and its ability to create more complex relationships between numerical values. In particular, the dataset of the two-digit model only contained a quarter of the number of examples compared to the one-digit dataset. Therefore, it was not clear whether the performance decreased due to an increase in the complexity of the arithmetic operations or whether it was due to the reduced dataset size.

To isolate the effect of dataset size, the two-digit arithmetic dataset was expanded to include 20,000 examples to match the size of the one-digit arithmetic dataset while containing an equal distribution of two-digit multiplication and addition problems. The model architecture, training procedure, and training iterations were all held constant while the dataset size was the only experimental variable between both two-digit trials. Across the five random seeds, the model produced a mean accuracy of 86.00% and a standard deviation of 2.29 percentage points (see Table 2). The mean accuracy increased by 17.26 percentage points from the initial dataset to the expanded dataset.

The evaluation prompts overlapped with the training data for both dataset sizes, but at different extents. For the 5,000 example dataset, 243 of the 1,000 evaluation prompt occurrences (24.30%) matched arithmetic problems represented in the training portion. On the other hand, 670 of the 1,000 evaluation prompt occurrences (67.00%) matched arithmetic problems represented in the training portion for the 20,000 example dataset. When considering only unique prompts, 233 of 970 problems (24.02%) overlapped with the 5,000 example dataset and 652 of 970 (67.22%) overlapped with the 20,000 example dataset. Therefore, the 17.26 percentage point increase may partly reflect greater exposure to evaluated problems during training. Operand-operator combinations were not deliberately held out. So, these results do not necessarily establish improved generalization to unseen arithmetic problems.

The results suggest that increasing the amount of training data from 5,000 to 20,000 examples substantially improved performance (see Figure 2). The mean accuracy of the two-digit model remained lower than the accuracy of the one-digit model. This demonstrates that the increased numerical complexity remained a challenge under the baseline architecture. These results motivated the subsequent experiment investigating the effect of increasing model capacity to determine whether that would provide an additional improvement beyond increasing the amount of training data.

Effect of Dataset Size on Arithmetic Performance

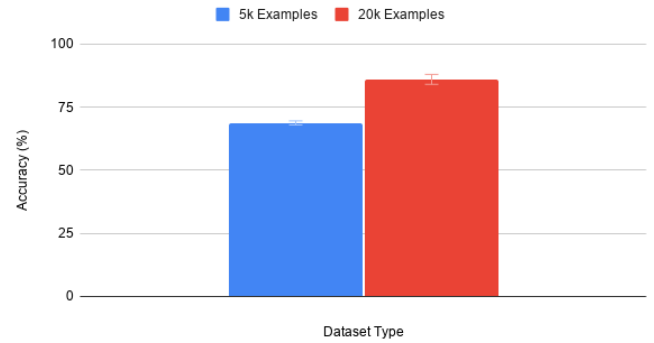


Figure 2: Accuracy comparison between 5,000 and 20,000 example datasets.

Table 2: Two-digit arithmetic accuracy across five random seeds for different dataset sizes.

Metric	5,000 Examples	20,000 Examples
Seed 1	69.70	84.00
Seed 2	67.70	83.10
Seed 3	70.00	87.70
Seed 4	68.90	87.10
Seed 5	67.40	88.10
Mean $\pm$ SD	68.74 $\pm$ 1.16	86.00 $\pm$ 2.29

4.3 Experiment 3: Model Scaling

After testing the effect of increasing the arithmetic dataset for the two-digit arithmetic model from 5,000 examples to 20,000 examples, this motivated the investigation of expanding the model architecture. Additionally, the one-digit model still performed substantially better than the two-digit model. Therefore, we decided to focus on altering the model architecture in order to make it larger to examine whether this factor would help the model learn more arithmetic relationships, allowing it to have a higher accuracy boost.

In order to isolate the effect of model capacity, the number of examples in the dataset was held constant at 20,000 examples. Additionally, the number of training iterations was held constant at 20,000 as well. The initial architecture of the model consisted of 4 transformer layers, 4 attention heads, and an embedding dimension of 128, resulting in 0.79 million trainable parameters. The new model was altered from 4 to 6 transformer layers, 4 to 6 attention heads, and an embedding dimension altered from 128 to 192. The largest configuration consisted of 2.66 million trainable parameters. The other model parameters such as dropout rate, batch size, block size, optimizer, and learning rate were held constant (see Table 3).

The largest two-digit arithmetic dataset, containing 20,000 examples, was used to train the model and was evaluated using the same prompt-based inference procedures as the original two-digit model. By keeping the dataset size, training duration, and evaluation methodology constant, this allowed for the evaluation of the model to be purely based on the increases in the model’s capacity

Table 3: Configuration comparison between baseline and expanded two-digit model architectures.

Parameter	Baseline	Expanded
Layers	4	6
Attention Heads	4	6
Embedding Dimension	128	192
Parameters	0.79M	2.66M
Dropout	0.2	0.2
Batch Size	64	64
Block Size	32	32
Optimizer	AdamW	AdamW
Learning Rate	1e-3	1e-3
Training Iterations	20,000	20,000

rather than the influence of external variables. This allowed us to compare the baseline and expanded model architectures in a controlled environment. The experiment went on to show whether larger transformer architectures were beneficial and would lead to improvements in accuracy when tackling tougher arithmetic expressions.

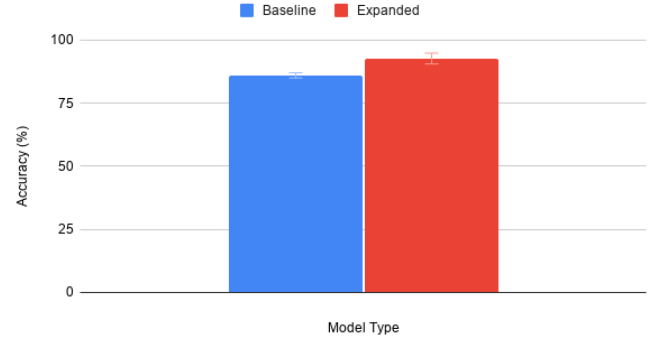
Across the five independent random seeds, the model produced a mean accuracy of 92.66% with a standard deviation of 1.22 percentage points (see Table 4). The model had a 6.66 percentage-point improvement compared to the baseline architecture which achieved an 86.00% mean accuracy (see Figure 3). Both the baseline and expanded models were trained on the same 20,000 example dataset and trained over 20,000 iterations. The evaluation overlap for the expanded model was the same because it used the same 20,000 example dataset and evaluation prompts as the baseline model. Therefore, accuracy improvement indicates better performance under the same data conditions, but does not necessarily establish improved generalization to unseen arithmetic problems.

Table 4: Two-digit arithmetic accuracy across five random seeds for the baseline and expanded model architectures.

Metric	Baseline	Expanded
Seed 1	84.00	93.50
Seed 2	83.10	90.70
Seed 3	87.70	92.90
Seed 4	87.10	93.80
Seed 5	88.10	92.40
Mean $\pm$ SD	86.00 $\pm$ 2.29	92.66 $\pm$ 1.22

The expanded model had an improvement over the baseline architecture. This improvement was smaller compared to the improvement achieved by increasing dataset size from 5,000 to 20,000 examples. However, both changes led to improved performance. Furthermore, the expanded model architecture allowed the model to further approach the accuracy of the one-digit arithmetic model which suggests that increased model capacity was associated with improved performance on two-digit arithmetic operations.

This experiment focuses on investigating overall model capacity rather than isolating the contribution of a single component

Effect of Model Size on Arithmetic Performance**Figure 3: Accuracy comparison between baseline and expanded two-digit models.**

of the model's architecture as the expanded architecture simultaneously increased the number of layers, attention heads, and embedding dimensions. Therefore, the results suggest an association between increasing model architecture and improved arithmetic performance.

4.4 Experiment 4: Natural Language Arithmetic Evaluation

The arithmetic-only experiments provided a foundation for understanding the factors that influenced the model's ability to learn and perform arithmetic operations. In order to add another level of complexity, arithmetic expressions were embedded into natural language prompts which meant that the arithmetic expressions were no longer isolated. These prompts contained surrounding words, punctuation, and non-mathematical characters. This was done to evaluate whether the model would still be able to identify relevant parts of the prompt and perform arithmetic operations despite the surrounding linguistic context.

Initially, natural language prompts were inputted into the arithmetic model to test the model. However, the model was unable to process these prompts purely due to a vocabulary limitation, producing key errors. This was the case because the initial character vocabulary only contained numerical values and operators. Therefore, the model was not able to process the natural language prompts because alphabetic characters and punctuation were not part of the tokenizer vocabulary. Examples of these vocabulary errors are shown in Table 5.

Table 5: Examples of vocabulary errors produced by the arithmetic-only model when processing natural language prompts.

Prompt	Output	Cause
I like pizza. 14*15=	KeyError	Unseen Character "I"
What is 88+23=	KeyError	Unseen Character "W"
Today is Sunday. 45+27=	KeyError	Unseen Character "T"

The vocabulary failure demonstrated that the arithmetic-only model was unable to directly evaluate natural language prompts without first modifying the tokenization scheme. Importantly, this did not suggest that the model was completely incapable of providing correct outputs with natural language surrounding arithmetic prompts as the model stopped directly after the first character when processing each prompt. In order to address this limitation, a new character-level vocabulary was created to account for both the original vocabulary, containing numerical values and operators, as well as alphabetic characters and punctuation marks required for the natural language prompts. The upgraded unified vocabulary allowed both mathematical and non-mathematical characters to be represented.

Subsequently, this motivated the construction of a new dataset which not only included numbers and operators, but also contained the letters and punctuation marks that were missing from the original arithmetic-only dataset. The same training framework and the same number of iterations from the previous experiments was used. Additionally, the range of numbers used was 10 to 99 which is the same range of numbers used in Experiments 2 and 3. Furthermore, the model contained the same expanded transformer architecture as Experiment 3. The initial natural language prompts were constructed and tested manually in order to examine whether the model could process the prompts under varied textual formats. Eventually, prompts were generated and tested in larger quantities to minimize the margin of error and reduce human bias.

The expanded natural language model was evaluated using a set of 1,000 prompts containing linguistic context as well as both two-digit multiplication and addition expressions. Similar to previous trials, model outputs were only considered correct if the output contained the correct numerical result. In order to successfully answer prompts, the model not only had to make sure that it developed the right numerical relationships for both addition and multiplication expressions, but it also had to focus on the relevant portion containing the arithmetic expression and reduce its focus on non-mathematical context.

The natural language model was evaluated through the standardized prompt-based evaluation procedure. The model produced the correct output for 943 prompts from the 1,000 prompt set, resulting in a 94.30% accuracy. The results suggest that the model retained substantial arithmetic performance even when numerical expressions were embedded within natural language prompts. The model seemed to perform slightly better than the expanded two-digit arithmetic model. However, this result is not directly comparable to the five-seed mean produced by the expanded two-digit architecture as it was produced by a single run.

4.5 Experiment 5: Interpretability Analysis

After noticing that the natural language arithmetic model was producing the right outputs for the majority of the evaluated prompts, this motivated further investigation in examining which input tokens influenced the behavior of the model, as the raw accuracy alone was insufficient to indicate which portions of the prompts were being used to generate outputs. In order to do so, interpretability experiments were conducted using the Captum library for PyTorch [6] for our fifth and most crucial experiment. Specifically, the

model's behavior was analyzed using Integrated Gradients [13] as the primary attribution method because it satisfies the Sensitivity and Implementation Invariance axioms and has the completeness property. Integrated Gradients was used because it estimates how much each input token contributes to the overall output by integrating gradients along a continuous interpolated path between a baseline input and the observed input.

Attribution scores were computed systematically across all 1,000 evaluation prompts which had diverse context prefixes. For every input token in each prompt sequence, attribution was evaluated with respect to the highest probability next-token prediction. Two baseline configurations were investigated: a zero embedding baseline and a mean embedding baseline. 100 integration steps were used for each calculation. The target token was selected as the model's highest probability prediction for the final input position. The embedding-dimension attribution values were summed to produce a single attribution score for each input token. Attribution scores were aggregated by token category to observe whether the attribution scores and patterns of individual prompts were consistent across the test set. The attribution scores were crucial as they provided insight into the contribution of each individual character to the model's output. This helped determine whether arithmetic operators and numerical values had a greater influence on the model's predictions than the surrounding natural language tokens. After evaluating all batches, the distribution of attribution scores revealed a divide between natural language prefixes and arithmetic operations.

After observing the zero embedding baseline, arithmetic operators received the highest mean attribution score ($+5.891 \pm 2.722$) and digits resulted in the second highest mean attribution score ($+1.436 \pm 1.239$). Punctuation seemed to produce an intermediate mean attribution of $+0.975 \pm 1.528$. Whitespace and letters received the lowest attribution scores. Similarly, this pattern was also consistent through the mean embedding baseline as operators ($+4.543 \pm 2.317$) and digits ($+1.645 \pm 1.262$) contributed most to model predictions while the whitespace and letters received substantially lower scores. The results indicate that the attribution patterns were broadly consistent across both baselines (see Table 6).

Table 6: Mean Integrated Gradients attribution $\pm$ standard deviation across 1,000 test prompts using zero embedding and mean embedding baselines.

Token	Zero	Mean
Operator	$+5.891 \pm 2.722$	$+4.543 \pm 2.317$
Digit	$+1.436 \pm 1.239$	$+1.645 \pm 1.262$
Punctuation	$+0.975 \pm 1.528$	$+0.918 \pm 1.824$
Whitespace	$+0.222 \pm 0.653$	$+0.152 \pm 0.493$
Letter	$+0.055 \pm 0.231$	$+0.109 \pm 0.276$

For example, in the prompt "Today is Sunday. 42+94=", the highest attribution scores were given to the arithmetic operators (the plus and equals sign) along with the numerical values (4, 2, 9, and 4) that preceded the prediction. On the other hand, lower attribution scores were given to most characters belonging to the natural language prefix (see Figure 4).

- (1) **Operational Tokens Attribution:** The arithmetic operators received the highest mean attribution scores across the test prompts. Specifically, the equals sign frequently produced the highest attribution score. This suggests the operators were strongly associated with the model's predicted outputs.
- (2) **Operand Digit Attribution:** The second operand's leading digits and the first operand's major positional components attracted the majority of feature weight, suggesting that the model's predictions were more strongly associated with numerical tokens than with most contextual tokens. Digits received the second highest attribution, suggesting they also contributed meaningfully to the model's predictions.
- (3) **Contextual Attribution:** The natural language portions of each prompt produced lower attribution scores than the other tokens or even produced negative scores. However, certain punctuation marks sometimes received higher scores than expected, suggesting that some non-mathematical tokens were not entirely irrelevant to the model's predictions.

Attribution Visualization for Example Prompt

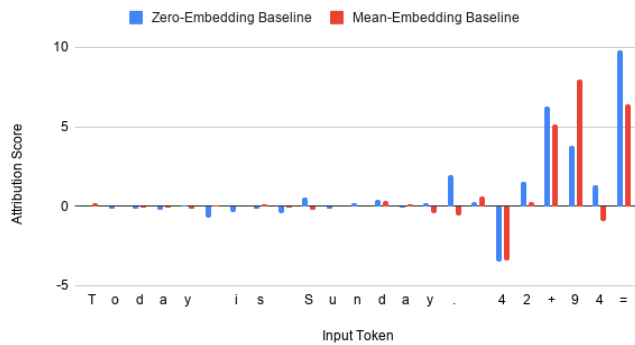


Figure 4: Character-level Integrated Gradients attribution scores for an example arithmetic prompt under zero embedding and mean embedding baselines.

Punctuation was further investigated as it occasionally received relatively high attribution scores. Two experiments were conducted to investigate the role of punctuation in arithmetic prompts. The first experiment focused on removing all punctuation marks. By doing so, we noticed the accuracy decrease from 94.30% on the original prompts to 93.00% on the prompts without punctuation. The 1.30 percentage point decrease in performance suggests that punctuation removal had a modest effect on performance (see Table 7).

Furthermore, the second punctuation experiment focused on shifting the position of the punctuation marks. Specifically, punctuation marks were shifted 5 characters to the left of their initial position. Accuracy decreased to 82.10% for the modified prompts, resulting in a reduction of 12.20 percentage points compared to the 94.30% on the original prompts (see Table 7). The larger decrease in performance suggests that punctuation placement and prompt structure can substantially affect model performance. However, this

altered the local context and naturalness of the prompts. Therefore, the decrease in accuracy may not be solely attributed to the positions of punctuation marks.

Table 7: Effect of punctuation modifications on accuracy.

Condition	Accuracy	Δ (Percentage Points)
Original prompts	94.30%	—
Punctuation removed	93.00%	−1.30
Punctuation shifted left	82.10%	−12.20

Integrated Gradients scores for punctuation were examined under both the zero embedding and mean embedding baselines for the original and modified prompts (see Table 8). For the original prompts, the zero embedding baseline produced mean attribution scores of 0.953 for periods and 1.010 for colons while the mean embedding baseline produced mean attribution scores of 1.089 for periods and 0.651 for colons. For the relocated punctuation results, periods had higher mean attributions than colons under both baselines. Under the zero embedding baseline, periods produced a mean attribution score of 1.280 compared to colons which produced a score of 0.330. Under the mean embedding baseline, periods produced a mean attribution score of 1.930 while colons produced a mean attribution of 0.465. At the aggregate punctuation-category level, the mean score decreased from 0.975 to 0.695 under the zero embedding baseline but increased from 0.918 to 1.028 under the mean embedding baseline. These differences indicate that the observed attribution patterns may depend on the baseline and prompt condition. One possible explanation is that punctuation near an arithmetic expression may provide a structural cue associated with the transition from natural-language context to the expression. However, the results do not fully establish that altering punctuation position solely causes the observed changes in accuracy or attribution since relocating the punctuation marks also changes the surrounding text.

Table 8: Attribution statistics across baselines and prompt conditions.

Baseline	Prompt	Period	Colon	All Punctuation
Zero	Original	0.953 ± 1.701	1.010 ± 1.210	0.975 ± 1.528
Zero	Shifted	1.280 ± 4.369	0.330 ± 0.837	0.695 ± 2.824
Mean	Original	1.089 ± 1.846	0.651 ± 1.760	0.918 ± 1.824
Mean	Shifted	1.930 ± 3.966	0.465 ± 0.905	1.028 ± 2.653

These observations suggest that the model placed greater emphasis on the arithmetic portion of the prompt when predicting the next token. After evaluating the 1,000 prompt test set, the average convergence delta was computed for each baseline. The average signed convergence delta was $+8.88 \times 10^{-4}$ with an average absolute convergence delta of 8.21×10^{-3} for the zero embedding baseline. The mean embedding baseline resulted in an average signed convergence delta of $+2.46 \times 10^{-4}$ and an average absolute convergence delta of 4.11×10^{-3} . The small average absolute convergence deltas indicate that the attribution calculations had relatively small approximation errors for both baselines.

Although these experiments represent an initial analysis, they provide evidence that operators and numerical values received greater average attribution than letters and whitespace. Additionally, the majority of the prompt outputs were correct regardless of the linguistic information. Furthermore, the consistency between the two baselines indicates that the observed attribution patterns were not solely dependent on a single baseline choice.

5 Discussion & Future Work

The experiments demonstrate that the character-level transformer models used in this study were able to learn and perform one-digit and two-digit arithmetic operations under a controlled evaluation setting. Increasing training iterations improved performance for one-digit operations while the decrease in performance of the initial two-digit model suggested that more complex models required additional training data or a larger model capacity. The two-digit model substantially improved when dataset size increased from 5,000 examples to 20,000 examples, rising from a mean accuracy of 68.74% to 86.00%. Expanding the model architecture, with the same 20,000 example dataset, further increased performance to 92.66%. The results suggest that both additional training data as well as increased model capacity are associated with improved performance [5].

The repeated seed evaluation demonstrated the importance of accounting for training variability. Both datasets and the expanded model were tested against five independent random seeds. The 5,000 example dataset model produced a standard deviation of 1.16 percentage points. The 20,000 example dataset model produced a standard deviation of 2.29 percentage points. The expanded model produced a standard deviation of 1.22 percentage points. These trials helped provide a more complete representation of model performance rather than solely relying on a single run.

Testing the natural language prompts with the arithmetic-only model showed that the model was originally unable to process these prompts, suggesting that adjusting the vocabulary design was necessary to test arithmetic operations beyond purely numerical inputs. After the vocabulary was expanded, the model successfully produced correct outputs for the majority of the prompts even with surrounding natural language, indicating that the initial failure was due to the limited character vocabulary rather than dataset size or model architecture.

Integrated Gradients provided further insight into the features associated with model outputs. Specifically, operators and digits had the highest mean attribution scores across both the zero embedding and mean embedding baselines. On the other hand, whitespace and letters produced substantially lower scores. The intermediate attribution scores produced by punctuation marks suggest that some contextual tokens did have more influence than expected on model predictions. The punctuation experiments also showed that changes to punctuation placement were associated with a substantial decrease in accuracy, although these changes also altered the surrounding text. This suggests that prompt structure may affect model performance, but further experiments are needed to isolate the effect of punctuation position itself. The attribution values identified the associations between inputs and predictions. However, these results should be interpreted as evidence of feature

contribution rather than direct evidence that the model possesses human-like reasoning abilities.

This study was focused on one-digit and two-digit arithmetic operations and the model's reasoning behavior was evaluated using Integrated Gradients on the final and most advanced model containing natural language. The controlled setting allowed for the analysis of token-level attribution and individual factors. However, it does not establish how the behaviors observed throughout the experiments would relate to larger language models or other mathematical reasoning tasks. The experiments do not compare the models against a non-transformer baseline. Additionally, model architecture parameters were increased simultaneously. Therefore, the effect of increasing overall model capacity was examined. Future work could isolate individual contributions of transformer layers, attention heads, or embedding dimensions. Additionally, future studies should focus on more complex arithmetic tasks, simpler computational baselines, larger benchmark datasets, other interpretability techniques, and examine whether larger or independently trained models exhibit similar attribution patterns while running trials on a GPU to increase computational speed.

6 Conclusion

This study has unpacked how character-level transformer models can learn arithmetic relationships in a controlled setting. Each examined factor demonstrated changes in arithmetic performance. However, the effect differed based on the factor being analyzed. Increasing dataset size substantially improved performance from a mean accuracy of 68.74% to 86.00% across five random seeds. Expanding the two-digit model's architecture increased performance to 92.66%, further reducing the gap in performance between the two-digit model and the simpler one-digit model. Integrated Gradients showed that the model assigned higher average attribution to arithmetic operators and numerical values than whitespace and letters, while the model achieved high accuracy on arithmetic expressions embedded in natural language. This overall pattern was consistent across both the zero embedding and mean embedding baselines. These findings demonstrate that interpretability methods are useful as they add an additional perspective, complementing accuracy-based evaluations, by identifying the influence of different input tokens on the model's predictions. More broadly, these experiments demonstrate the value of investigating transformer behavior through both accuracy and interpretability techniques rather than solely relying on model accuracy to provide evidence of reasoning.

7 Acknowledgements

The author sincerely thanks Dr. Sumit Kumar Jha of the University of Florida for his guidance and mentorship throughout this research. His support and feedback were especially valuable regarding the interpretability analysis during the study using Integrated Gradients.

References

- [1] Abnar, S., Zuidema, W.: Quantifying attention flow in transformers. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. pp. 4190–4197 (2020)

- [2] Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., et al.: Language models are few-shot learners. In: *Advances in Neural Information Processing Systems*. vol. 33 (2020)
- [3] Elhage, N., Nanda, N., Olsson, C., et al.: A mathematical framework for transformer circuits (2021), <https://transformer-circuits.pub/2021/framework/index.html>
- [4] Jain, S., Wallace, B.C.: Attention is not explanation. *arXiv preprint arXiv:1902.10186* (2019)
- [5] Kaplan, J., McCandlish, S., Henighan, T., Brown, T.B., Chess, B., Child, R., et al.: Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020)
- [6] Kokhlikyan, N., Miglani, V., Martin, M., Wang, E., Alsallakh, B., Reynolds, A., et al.: Captum: A unified and generic model interpretability library for pytorch. *arXiv preprint arXiv:2009.07896* (2020)
- [7] Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: *International Conference on Learning Representations* (2019)
- [8] Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: *Advances in Neural Information Processing Systems*. vol. 30 (2017)
- [9] Power, A., Burda, Y., Edwards, H., Babuschkin, I., Misra, V.: Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177* (2022)
- [10] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners. *OpenAI Technical Report* (2019)
- [11] Ribeiro, M.T., Singh, S., Guestrin, C.: "why should i trust you?": Explaining the predictions of any classifier. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. pp. 1135–1144 (2016)
- [12] Saxton, D., Grefenstette, E., Hill, F., Kohli, P.: Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557* (2019)
- [13] Sundararajan, M., Taly, A., Yan, Q.: Axiomatic attribution for deep networks. In: *International Conference on Machine Learning*. pp. 3319–3328 (2017)
- [14] Trask, A., Hill, F., Reed, S., Rae, J., Dyer, C., Blunsom, P.: Neural arithmetic logic units. *arXiv preprint arXiv:1808.00508* (2018)
- [15] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: *Advances in Neural Information Processing Systems*. vol. 30 (2017)
- [16] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., et al.: Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903* (2022)