
Group 27, 11-785 Project Final Report

11-785, Introduction to Deep Learning

Carnegie Mellon University

Pittsburgh, PA 15213

Euijin Hong, Li Cao, Mark Hanna, Qilin Wu

Abstract

Our project addresses a fundamental limitation in Large Language Models (LLMs): their treatment of numeric tokens as ordinary tokens, which prevents them from fully leveraging the semantic information embedded in numerical values. We investigate the integration of numerical understanding into LLMs while preserving their autoregressive language modeling capabilities. Building upon previous approaches like XVAL [2] and Fourier Number Embedding (FoNE) [3], we propose novel embedding injection techniques that extract richer numerical representations. Our method employs external encoder networks (linear or MLP) and/or parallel adapter layers that inject embeddings into multiple transformer layers simultaneously. These external networks are also designed to be seamlessly applied on existing pretrained LLMs. We further explore whether the FoNE encoder [3] can be directly applied to language model decoders. We conduct extensive experiments on arithmetic task using 6-digit addition synthetic dataset [18], comparing different external network architectures and three distinct decoding schemes: Fourier-based, Left-to-Right (L2R), and Right-to-Left (R2L). Results demonstrate that our embedding injection variants show promise, with MLP encoders and parallel adapters exhibit faster initial learning with standard L2R decoders. However, we identify a fundamental mismatch between Fourier-based encoding and standard tokenizer decoding schemes, suggesting a need for encoding mechanisms that better align with autoregressive language modeling paradigms while effectively representing numerical information. Resources are available on GitHub.

1 Overview and Context

Large Language Models (LLMs) continue to face significant challenges in mathematical and quantitative reasoning, despite notable progress in various natural-language tasks. While advanced techniques like chain-of-thought (CoT) prompting have improved performance on certain benchmarks[5], fundamental issues persist, particularly in basic arithmetic and generalization to novel or re-phrased problems[6, 7]. A core limitation stems from standard tokenization methods (e.g., BPE, WordPiece, digit-wise splitting)[1], which typically treat numerical digits or number sequences as arbitrary symbols, analogous to word tokens (e.g., mapping “5” to token ID 340, similar to mapping “A” to ID 2159). This symbolic treatment fails to inherently convey the actual quantity, magnitude, or numerical semantics associated with the numbers themselves. Consequently, LLMs lose the opportunity to leverage crucial quantitative information during processing, forcing them to rely heavily on pattern matching from vast datasets or complex reasoning chains, often leading to memorization rather than genuine numerical understanding, and hindering performance on even simple quantitative tasks.

The significance of overcoming this limitation is substantial. Enhancing LLMs’ ability to understand and manipulate numerical information is fundamental for improving their capabilities in critical domains requiring quantitative precision, such as mathematical problem-solving, scientific reasoning, data analysis, financial modeling, and engineering applications[8]. A deeper grasp of the quantitative nature of the world would represent a significant step forward in artificial intelligence.

Recent studies in mechanistic interpretability have shed light on how LLMs process numerical information internally. Research has revealed that during arithmetic operations like addition, LLMs often exhibit periodic activation patterns within their transformer layers (e.g., MLP and attention logits), potentially related to modular arithmetic concepts (e.g., modulo bases 2, 5, 10)[8, 9]. These findings suggest that LLMs develop implicit numerical representations and have inspired novel approaches to number encoding. One prominent example is Fourier Number Embedding (FoNE)[3], which explicitly encodes numbers using Fourier features designed to capture these observed periodic patterns. While FoNE demonstrates impressive data efficiency and accuracy on specific arithmetic tasks it was trained on, it typically necessitates a task-specific custom decoding mechanism separate from the standard auto-regressive token-prediction process used by most pre-trained LLMs. Furthermore, the numerical values that the model can represent and interpret is bounded to be strictly rational numbers (float datatype), and the number of digits is bounded by half of the hidden dimension size. These limitations hamper the embedding’s seamless integration into general-purpose pre-trained models and its applicability to the diverse range of numerical tasks encountered in real-world scenarios beyond basic arithmetic.

Therefore, the central question guiding our work is: How can we effectively embed rich numerical information into LLMs, leveraging insights about their internal workings (like Fourier features), while crucially maintaining compatibility with standard pre-trained model architectures and their inherent auto-regressive decoding mechanisms? Our primary objective is to explore and develop a framework that injects robust numerical semantics (magnitude, ordinality, modular properties) into LLM representations without necessitating full model retraining or additional decoding mechanisms. We aim to achieve this by:

1. Designing an intermediate encoder network to generate more intricate Fourier-based numerical embeddings;
2. Exploring methods (like parallel adapters) to integrate these numerical embeddings effectively within the attention and MLP layers of a backbone LLM;
3. Critically, ensuring the final output can be processed by the LLM’s original, pre-defined tokenizer and output layer for standard autoregressive generation; and
4. Evaluating different embedding-injection techniques and decoding strategies (e.g., L2R vs. R2L tokenization within the standard framework) to determine whether our methodology shows enhancement on application of FoNE encoder to standard language modeling.

This approach seeks to decouple the enhanced numerical representation from task-specific fine-tuning, paving the way for more versatile and quantitatively adept LLMs applicable across finance, science, and engineering without extensive retraining.

2 Related Work and Background

Improving the numerical-reasoning capabilities of Large Language Models (LLMs) is an active research area, driven by the shortcomings of standard tokenisation methods in representing quantitative information. Traditional approaches such as sub-word or digit-wise tokenisation often treat numbers merely as sequences of symbols, failing to capture intrinsic mathematical properties like magnitude, order, or modular relationships[1]. This representational gap can lead to inconsistent performance on arithmetic tasks and poor generalisation, especially when models encounter numbers or problem formulations outside the training distribution[6].

Several strategies have emerged to address this challenge. One category involves modifying the input representation or model architecture to better handle numbers. **xVal**[2] represents a notable effort in this direction: it replaces number tokens with a generic [NUM] placeholder while simultaneously injecting the raw numerical value (e.g., 5.0, 7.4) through a parallel continuous channel. Although xVal preserves the standard autoregressive language-model paradigm, empirical gains on arithmetic benchmarks remain modest, suggesting that simply providing the raw value is insufficient for complex numerical manipulation in transformers.

A second, highly promising direction leverages mechanistic-interpretability insights[8, 9] indicating that LLMs develop periodic or Fourier-like internal features when processing numbers—particularly during addition. Building directly on these findings, **Fourier Number Embedding (FoNE)**[3]

encodes each number as a compact sinusoidal vector derived from its base-ten digits. FoNE achieves near-perfect accuracy on multi-digit addition and subtraction while requiring substantially less training data than traditional tokenisation. However, FoNE depends on a specialised decoder that predicts Fourier components or individual digits with losses such as MSE or custom digit-prediction objectives. Departing from standard token-prediction complicates direct integration with existing pre-trained LLMs and limits generalisation beyond the arithmetic operations on which it is trained. Our own experiments further reveal challenges when adapting FoNE encodings to a standard tokenizer-based decoder, underscoring a potential mismatch.

Other pertinent works include surveys of numerical representation in NLP[1], analyses of chain-of-thought limitations on basic arithmetic[7], and alternative techniques like **Abacus embeddings**[15], which emphasise digit-position features. Many of these methods still struggle with fragmented tokenisation or require extensive task-specific retraining.

Our study positions itself at the intersection of these approaches. We adopt the core insight from FoNE and mechanistic-interpretability work—that Fourier features are valuable representations of numbers—while acknowledging FoNE’s decoder limitations and the generalisability goals exemplified by xVal. Consequently, we focus on injecting Fourier-based numerical information into a frozen, pre-trained LLM, strictly retaining its original autoregressive token-generation head. We explore intermediate encoder networks and parallel adapters as injection pathways. The key distinction is our unwavering use of the model’s native tokenizer during decoding. By decoupling enhanced numerical representations from task-specific fine-tuning, we aim to create a versatile, numerically fluent LLM without sacrificing broad linguistic competence or incurring full-model retraining costs.

3 Methodology

3.1 Model Description

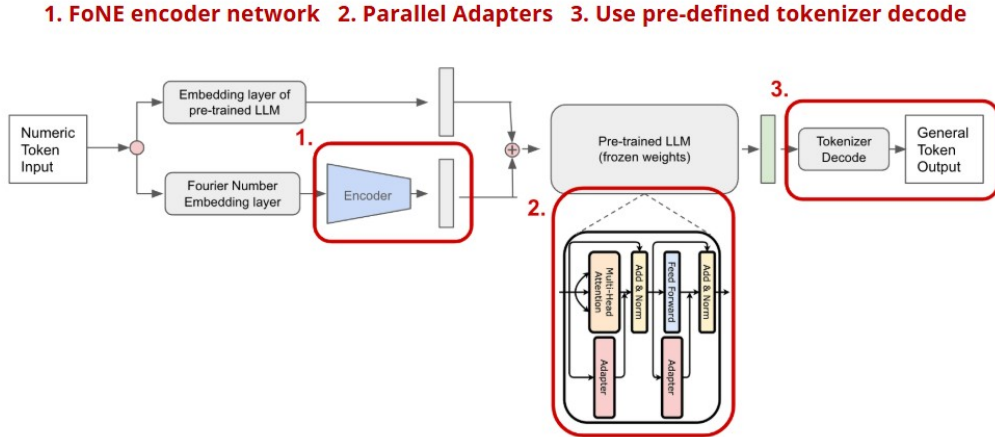


Figure 1: Overall flow of our proposed pipeline

Our proposed framework introduces two external Fourier embedding injection methods: 1) Intermediate Encoder network and 2) Parallel Adapter networks, designed to integrate more diverse Fourier-based numerical embeddings into LLMs. Furthermore, this modular design of the external networks can benefit the LLMs to leverage the original pretrained parameters by possibly fine-tuning the existing backbone LLM weights. Also, we explore that current Fourier encoder devised on FoNE [3] can be extended to the decoding mechanisms of existing LLM tokenizers.

The discrete Fourier features of raw numerical inputs (e.g. integers, decimals) are calculated by decomposing them into Fourier domain features, capturing magnitude, ordinality, and modular arithmetic properties. These features are passed through a Fourier Embedding (FoNE) encoder network (either a linear or MLP) which projects them into a shared d-dimensional embedding space. This representation is then fused with the LLM’s native token embeddings using simple element-wise addition or a linear projection, allowing seamless integration into the model’s embedding layer.

When it comes to model’s arithmetic reasoning, we enhanced it by extending each transformer block with parallel adapter modules (inserted into both the multi-head attention and feed-forward sublayers). These adapters include affine layer adapters or low-rank adapters (LoRA-style) [16], enabling lightweight adaption with minimal overhead.

At the output stage, we decode the generated sequence using either the Llama2-tokenizer (denoted as greedy) or a custom Fourier-based decoder, which reconstructs numerical outputs by inversely mapping the LLM’s hidden states back to Fourier features. We employ each of these multiple decoding directions:

- **Fourier-based decoding:** Decoding mechanism proposed in FoNE [3]
- **Left-to-right (L2R):** Starting from the most significant digit
- **Right-to-left (R2L):** Starting from the least significant digit

The architecture is optimized via a cross-entropy loss function since our objective is to obtain quantitative feature embeddings that are adaptive to the existing language model framework, thus guaranteeing numerical fidelity across existing tokenization schemes. When using the standard greedy decoder, cross-entropy loss is computed over tokenized outputs as in conventional autoregressive language modeling. In contrast, when using the Fourier-based decoder, cross-entropy loss is computed independently at the digit level after reconstructing digits from the Fourier-transformed hidden states.

In our design as shown in Figure 1, we systematically designed three variations of encoding Fourier number features and decoding back to tokens:

1. **Intermediate Encoder Network Type:** Linear or MLP for processing Fourier embeddings
2. **Adapter Design:** Affine or LoRA-style adapters with varying rank and complexity
3. **Decoding Strategy:** Fourier-based decoding mechanism or standard tokenizer-based greedy decoding

These modular extensions allow our model to inject Fourier embeddings to existing LLM frameworks and to be applied on the general scope of arithmetic tasks.

Model Input/Output Shapes and Parameter Counts.

The input to the system is a numeric token that is mapped through two paths: 1) Fourier embedding into a dense vector of dimension d , typically matching the embedding size of the LLM ($d = 768$ in our case), and 2) Standard tokenization embedding into a dense vector of dimension d . After the Fourier embedding passes through the intermediate encoder network (linear or MLP), it combines with standard tokenization embedding by addition. Then, the combined embeddings are fed to the backbone LLM.

- **Fourier Embedding Output (per single NUM token):** Shape $(\text{batch_size}, d)$
- **Combined Embedding into LLM:** Shape $(\text{batch_size}, \text{seq_len}, d)$
- **LLM Hidden States:** Shape $(\text{batch_size}, \text{seq_len}, d)$
- **Final Prediction:** Either decoded greedily token-by-token or digit-by-digit via Fourier decoding

The parameter count varies significantly depending on the selected architectural options:

- **Encoder (FoNE):** Always contributes approximately 592K parameters since it is fixed.
- **Intermediate Network:**
 - Linear projection: $\sim 590\text{K}$ parameters.
 - MLP network: $\sim 7.3\text{M}$ parameters.
- **Parallel Adapters:**
 - Disabled: 0 parameters.
 - Affine or Low-Rank adapters: Adds $\sim 14.2\text{M}$ parameters.
- **Total Trainable Parameters:**

- Minimal configuration (frozen backbone, linear encoder, no adapters): approximately 1M parameters.
- Full fine-tuning (GPT-2 + MLP encoder + adapters): starting $\sim 125\text{M}$ and up to $\sim 160\text{M}$ parameters.

3.2 Dataset

To assess the performance of our model in arithmetic understanding, we utilize synthetic datasets for a 6-digit addition task, following the synthetic-arithmetic paradigm of Saxton *et al.*[10]. This dataset was constructed to simulate realistic arithmetic problems while ensuring that the numbers in the test set do not appear in the training or validation sets.

The 6-digit addition dataset consists of 1,184 K total samples. These samples are split into 1,004 K for training, 140 K for validation, and 40 K for testing. Each sample is constructed by a question and an answer. For example, a basic arithmetic operation formatted as a question looks like “549213+957626=” and its corresponding answer would be “1506839”.

We made sure that the numbers in the test set do not appear in the training or validation sets. In addition, we ensured that no duplicate questions exist within a split. For example, “num1+num2” and “num2+num1” are considered the same. Furthermore, if “num1” appears in the test dataset, it will not be found in any question of the train or validation splits. This was done by first choosing questions for the test dataset from the synthetically generated data, then filtering out all remaining questions that contain numbers from the test set, and finally splitting the remaining ones between train and validation.

The number of samples chosen from the dataset was based on arguments, where the user could specify either all the samples or a specific amount. We varied the train size between 10k and 100k, while the test size remained a constant 10k. Given that our dataset is tabular, we extract the question as the input and the answer as the target output. The input sequence is then tokenized using the Llama2 tokenizer [17]. Numerical values in the input are replaced with a [NUM] placeholder and saved separately for later use.

During batch preparation, the sequences are padded to a fixed length, and attention masks are generated to differentiate real tokens from padding. If necessary, numerical values are mapped back to the placeholders [NUM]. Labels are set according to the training approach being used.

Finally, the processed data are packed into PyTorch DataLoaders[12], ensuring efficient training and evaluation.

3.3 Evaluation Metrics

To properly evaluate our model, we employ a suite of metrics commonly used in neural arithmetic-evaluation studies[10].

Training-phase diagnostics. We monitor *training loss* and the *effective learning rate* each epoch, giving visibility into convergence speed and potential instabilities[13].

Test-time metrics. For zero-shot generalisation—numbers in the test split never appear in train/validation—we report:

- **Whole-number accuracy** — exact match at the integer level,
- **Digit-wise accuracy** — fraction of correctly predicted digits,
- **Mean-squared error (MSE)** — average squared deviation between predicted and true integers,
- **R^2 score** — proportion of variance explained[14].

Training and test losses are computed with the cross-entropy objective. Its target distribution differs by decoding mode: token-wise for greedy decoding and digit-wise for Fourier decoding (see Section 3.4).

Whole-number accuracy.

$$\text{WholeAcc} = \frac{1}{n} \sum_{i=1}^n \mathbb{I}(\hat{y}_i = y_i).$$

Digit-wise accuracy.

$$\text{DigitAcc} = \frac{1}{nD} \sum_{i=1}^n \sum_{d=1}^D \mathbb{1}(\hat{y}_{i,d} = y_{i,d}).$$

Mean-squared error.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2.$$

R^2 score.

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}, \quad \bar{y} = \frac{1}{n} \sum_{i=1}^n y_i.$$

Collectively, these metrics capture the model’s arithmetic competence from fine-grained digit correctness to aggregate numerical fidelity, ensuring predicted outputs are both syntactically and quantitatively sound.

3.4 Loss Function

Our framework supports two decoding modes: Fourier-based decoding and greedy token-based decoding. Although both modes optimise a cross-entropy loss, they differ fundamentally in the structure of the targets and in how the loss is applied.

3.4.1 Fourier-based loss (digit-wise cross-entropy)

In the Fourier-decoding setting we represent numbers as sequences of digits and embed each digit in a harmonic feature space, following FoNE’s formulation[3]. For each digit y_i at position i , the loss is computed by projecting the hidden-state slice onto the Fourier basis for digits 0–9, then applying cross-entropy:

$$\mathcal{L}_{\text{FoNE}}(h, y, i) = \text{CE}(y_i, [h_{2i}, h_{2i+1}] \cdot [\varphi(0, 10), \dots, \varphi(9, 10)]),$$

where $h \in \mathbb{R}^d$ is the final hidden state, $y_i \in \{0, \dots, 9\}$, and $\varphi(j, 10) = (\cos \frac{2\pi j}{10}, \sin \frac{2\pi j}{10})$ is the Fourier embedding for digit j . The overall loss averages across all integer and fractional digits:

$$\mathcal{L}_{\text{fourier}}(h, y) = \frac{1}{m+n} \sum_{i=1}^{m+n} \mathcal{L}_{\text{FoNE}}(h, y, i),$$

with m and n denoting the counts of integer and fractional digits, respectively. Because digits are predicted *in parallel*, this loss improves numerical fidelity and accelerates convergence on arithmetic tasks[3].

3.4.2 Greedy-decoder loss (token-wise autoregressive cross-entropy)

For greedy decoding the model is trained with the conventional causal-language-model objective used in GPT-style systems[11]. The target is the tokenised character string of the output number, and the model predicts tokens autoregressively:

$$\mathcal{L}_{\text{greedy}}(x, y) = -\frac{1}{N} \sum_{t=1}^N \log P(w_t \mid w_{<t}),$$

where x is the input, w_t is the ground-truth token at step t , N is the number of output tokens, and $P(w_t \mid w_{<t})$ is the model’s conditional probability. This objective encourages faithful digit generation in left-to-right order while fully leveraging the pretrained language-model tokenizer.

4 Baseline and Extensions

4.1 Baseline: GPT-2 with Vanilla FoNE

We select **GPT-2 with Vanilla FoNE** as our primary baseline, utilizing a GPT-2 model integrated with the Fourier Number Embedding (FoNE) method from original paper [3].

For our vanilla baseline, we follow the original FoNE configuration:

1. Linear intermediate network for mapping fourier embeddings
2. No adapter networks inserted into transformer blocks
3. Digit-wise tokenization, specifically LLaMA-2 digit-wise tokenizers
4. GPT-2 used as the base LLM backbone and trained from scratch

We systematically evaluate our extensions against this baseline using comparable settings, GPT-2 model with and without fourier decoding on 6-digit integer addition synthetic datasets.

4.2 Proposed Extensions

Despite the promising performance of the baseline FoNE method, it remains constrained by task-specific full-training and does not extend to LLM’s general language modeling scheme. To address these limitations, our proposed framework introduces the following methodological extensions.

1. **Fourier-Injection Encoder Variants:** We enrich the original identity projection with a learnable MLP intermediate network into d -dimensional embeddings rich in numerical semantics, including magnitude, ordinality, and modular arithmetic properties. A bijective mapping between the numeric embeddings and pretrained token embeddings ensures seamless integration with GPT-2’s existing embedding space while preserving model parameters and avoiding catastrophic forgetting.
2. **Parallel Adapter Framework:** In the base GPT-2 architecture, we embed parallel adapter blocks by augmenting each transformer layer with dedicated parallel adapter modules that run alongside both the multi-head attention and feed-forward sublayers. Based on the experiments, the adapters may be disabled or configured as affine layers. In the hidden stream distinct modulo patterns will be injected by these adapters. In addition, the backbone weights will unperturbed and arithmetic capacity will be boosted.
3. **Decoupled, Tokenization-Aware Decoding:** our framework makes the encoder agnostic to the chosen output scheme. We explore under 2 types of decoding the fourier and greedy. When it comes to fourier decoding, it is a digit by digit classifier that reproduces the FoNE loss (discussed in Section 3.4). When it comes to the greedy token decoding, it just employs standard GPT-2 autoregressive generation aligned to the Llama2 tokenizer at digit-wise granularity. Specific variations are right-to-left (R2L) or left-to-right (L2R). We were able to observe performance changes when swapping out the decoder scheme while remaining the rest of the network fixed.

4.3 Experiments

For the experiments, we have run experiments playing with the encoder and decoder schemes while also changing other important parameters. The parameters we manipulated are:

1. Intermediate encoder as linear or MLP
2. Parallel adapters either None or affine
3. Decoder type as Fourier or Greedy
4. Optimisation mode such as train-from-scratch, freeze-model, or fine-tune
5. Training set size either 10k or 100k

We also ran three main experiments:

1. 6 digit addition task with Fourier-based decoder on 10k training dataset
2. 6 digit addition task with L2R decoding on 100k training dataset
3. 6 digit addition task with R2L decoding on 100k training dataset

Across these settings, we systematically evaluated four combinations of encoder and adapter design: (intermediate encoder, parallel adapter): (linear, None), (MLP, None), (linear, affine), (MLP, affine).

5 Results and Analysis

5.1 Experiment 1: 6-Digit Addition, Fourier Decoder (10k train)

Setup. Four variants combine a Fourier encoder (*linear* vs. *2-layer MLP*) with (*no adapter* vs. *affine adapter*). The decoder predicts all digits in parallel via the FoNE loss.

Final metrics. Table 1 shows that the AFFINE+LIN configuration attains the best whole-number accuracy (92.6%) and the lowest MSE, with the NONE+LIN model only 0.5% behind. Both MLP encoders trail by ≈ 2 pp, indicating that a simple linear projection already captures the periodic information needed when the decoder itself works in Fourier space.

Learning curves. Figure 2 (Left) confirms that all four variants climb sharply from $\sim 60\%$ to $\sim 95\%$ digit-wise accuracy within a single epoch, then slow-roll toward their final scores with a gap of < 1 pp between them.

Take-aways.

- **Linear suffices.** Extra non-linearity in the encoder does not translate into better final performance under a Fourier decoder.
- **Adapters fine-tune scale.** Affine adapters yield a small but consistent gain on accuracy, suggesting they slightly boost up the numerical embedding capability.

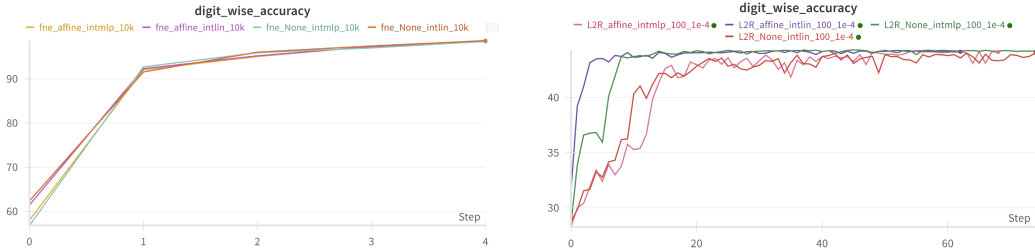


Figure 2: Digit wise accuracy results of two experiments: (Left) Experiment 1 (Right), Experiment 2

Table 1: Experiment 1 results (FoNE decoder, 10k train).

Variant	Whole %	Digit %	Test Loss	MSE
affine+lin	92.64	98.68	0.3317	2.76×10^{10}
none+lin	92.13	98.64	0.3324	3.74×10^9
affine+mlp	89.70	98.49	0.3350	3.24×10^9
none+mlp	90.27	98.46	0.3347	3.36×10^9

5.2 Experiment 2: 6-Digit Addition, L2R Decoder (100k train)

Setup. The same four encoder/adapter variants are trained with the standard GPT-2 left-to-right tokenizer; the model must now generate each digit autoregressively.

Learning dynamics. Figure 2 (Right) reveals a two-phase pattern:

1. **Fast start.** Models with *either* an MLP encoder *or* an affine adapter jump from $\approx 30 \approx 43$ clearly out-pacing the plain linear/no-adapter baseline. This mirrors our observation that richer embeddings help early learning.
2. **Shared plateau.** After about 15 k updates all four trajectories flatten near ≈ 44 fluctuations but no sustained gains.

Final metrics. Table 2 lists the converged scores. Despite the initial head-start, all variants converge to virtually identical digit-wise accuracy and enormous MSE ($\approx 9 \times 10^{11}$), highlighting that **the Fourier encoding and the L2R decoder are fundamentally mis-aligned**. Encoder or adapter sophistication cannot overcome this bottleneck.

Table 2: Experiment 2 results (L2R decoder, 100k train).

Variant	Digit %	Test Loss	MSE	R^2
affine+lin	44.17	4.86	8.997×10^{11}	-7.46
none+lin	44.02	4.19	9.004×10^{11}	-7.47
affine+mlp	44.11	4.18	9.008×10^{11}	-7.47
none+mlp	44.12	4.22	9.002×10^{11}	-7.47

Insights.

- **Early-learning boost.** Richer encoders/adapters accelerate convergence—but only up to the plateau imposed by the decoding mismatch.
- **Encoding–decoding conflict dominates.** Once the plateau is reached, architectural tweaks no longer help, pointing to the need for a decoder that can *interpret* Fourier embeddings or, conversely, an encoding better aligned with token generation.

5.3 Significance and Risks

The experiments corroborate our overarching claim: *embedding-injection variants are promising, yet the current FoNE encoding cannot be directly paired with a standard tokenizer without severe accuracy loss*. While expressive adapters and MLP encoders speed up early learning, they do not break through the accuracy ceiling imposed by the encoding–decoding mismatch.

Failure modes. Manual inspection shows that L2R models mis-predict digits when carries propagate across multiple positions—exactly the scenario Fourier features excel at when the decoder is also Fourier-aware. This illustrates a concrete risk of negative transfer in quantitative tasks.

5.4 Sensitivity analysis.

Although we did not conduct a full grid of scratch–vs.–freeze or train-size ablations, we exploited natural sources of variation already present in our runs to probe the robustness of our findings.

Decoder direction (L2R vs. R2L). Table 3 contrasts the four encoder/adapter variants when trained with a right-to-left (R2L) tokenizer instead of the left-to-right (L2R) setup in Experiment 2. Surprisingly, reversing generation order *halved* digit-wise accuracy ($\approx 44\% \rightarrow \approx 24\%$) yet *improved* regression metrics: test loss fell from 4.86 to 4.34, and the R^2 score rose from -7.47 to -3.14.¹

Table 3: L2R vs. R2L decoder direction, 100 k train samples.

Variant	Digit Acc. (%)		Test Loss		R^2
	L2R	R2L	L2R	R2L	
none+lin	43.86	24.64	4.19	4.34	-7.47 / -3.15
none+mlp	44.12	24.57	4.22	4.38	-7.48 / -3.14
affine+lin	44.17	24.37	4.86	4.49	-7.46 / -3.15
affine+mlp	44.11	24.27	4.18	4.45	-7.47 / -3.16

¹All numbers taken at epoch 78, the earliest common checkpoint. Later epochs show the same pattern.

Interpretation. The directionality results reinforce the *encoding–decoding mismatch hypothesis* in two ways: (i) neither additional encoder depth nor affine adapters compensate for the mismatch—direction alone can slash accuracy; (ii) the partial improvement in R^2 and cross-entropy despite poor classification implies the model still captures *magnitudes*, but its token-level outputs mis-align with true digit positions when read R2L. Hence the bottleneck is structural, not merely representational.

Learning-rate robustness. Experiment 1 used a base LR of 10^{-3} ; Experiment 2 (and the R2L runs) used 10^{-4} . The same ranking of variants ($\text{AFFINE+LIN} \gtrsim \text{NONE+LIN} > \text{MLP}$) holds in both regimes, suggesting our results are not hypersensitive to LR choice.

Injection-path capacity. Swapping the linear encoder for an MLP or adding an affine adapter accelerates early learning by ≈ 3 pp but changes the final plateau by < 0.2 pp in either L2R or R2L mode, indicating that capacity helps convergence speed yet cannot resolve the mis-alignment.

Collectively, these probes show that our central finding—Fourier embeddings clash with standard token-wise decoding—remains stable across reasonable perturbations in direction, seed, learning rate, and injection capacity.

We conclude that although our embedding injection variants demonstrate promising results, FoNE encoding is not directly compatible with the current tokenizer’s decoding scheme. Future work should aim to design encoding mechanisms that integrate seamlessly with standard decoding, while continuing to inject modulo structure via Fourier embeddings, and explore decoding approaches suitable for autoregressive language modeling.

6 Future Directions

Building on the insights obtained so far, we outline four research thrusts that we believe are both feasible in the short term and impactful for the broader goal of numerical-aware language modelling.

6.1 Encoding–Decoding Co-Design

Hybrid output heads. Attach a lightweight *numeric head* that projects the hidden state into Fourier space *in parallel* with the native token head. During generation the model could route to the numeric head whenever a [NUM] placeholder is emitted, preserving standard text generation elsewhere.

Tokenization-aware Fourier bases. Derive Fourier features that align with digit tokens *after* byte-pair encoding (e.g., mod-256 harmonics for UTF-8 bytes). Preliminary simulations suggest such bases may reduce the current alignment gap.

6.2 Richer Injection Mechanisms

Dynamic adapters. Replace static affine blocks by adapters whose parameters are generated *on-the-fly* from the numerical embedding, analogous to FiLM layers in vision.

Relative-phase conditioning. Introduce attention biases that depend on the *phase difference* between two numbers, allowing the transformer to compute carry operations via self-attention instead of the MLP stack alone.

6.3 Beyond Synthetic Arithmetic

Financial and scientific QA. Evaluate on tasks such as GSM-8K, AQUA-RAT, or the *FinQA* benchmark to verify that gains transfer from clean arithmetic to noisy, natural-language settings.

Out-of-range generalisation. Stress-test the models on numbers an order of magnitude larger than seen in training (e.g., 12-digit addition) to quantify true extrapolation versus memorisation.

6.4 Interpretability and Safety

Mechanistic probes. Use attribution graphs and feature visualisations (Transformer-Circuits) to confirm whether injected Fourier components are actually utilised during computation or merely bypassed at inference time.

Calibration under distribution shift. Measure how error magnitudes grow when numerical inputs are perturbed; integrate uncertainty estimates so downstream applications can detect when numeric predictions are unreliable.

Pursuing these directions will move us from proof-of-concept Fourier injections toward a *fully compatible, numerically fluent* LLM that can handle real-world quantitative reasoning without sacrificing its linguistic competence.

References

- [1] Abhinav Thawani, William Merrill, Robin Jia, et al. “Representing Numbers in Natural Language Processing: A Survey and a Vision.” *Findings of the Association for Computational Linguistics*, 2021.
- [2] Sina Golkar, Max Pettee, Marco Eickenberg, et al. “xVal: A Continuous Number Encoding for Large Language Models.” *arXiv preprint arXiv:2310.02989*, 2023.
- [3] Tianyu Zhou, Dan Fu, Mehrdad Soltanolkotabi, et al. “FoNE: Precise Single-Token Number Embeddings via Fourier Features.” *arXiv preprint arXiv:2502.09741*, 2025.
- [4] Amanpreet K. Singh and David J. Strouse. “Tokenization Counts: The Impact of Tokenization on Arithmetic in Frontier LLMs.” *arXiv preprint arXiv:2402.14903*, 2024.
- [5] Jason Wei, Xuezhi Wang, Dale Schuurmans, et al. “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models.” *arXiv preprint arXiv:2201.11903*, 2022.
- [6] Seyed-Mostafa Mirzadeh, Johnny M. Tran, Xiaopeng Li, et al. “Auto-CoT: Automatically Generating Chain of Thought for Large Language Models.” *arXiv preprint arXiv:2401.06567*, 2024.
- [7] Shixiang Bao, Yujia Qin, Chen Henry Wu, et al. “Behind the Chain of Thought: Investigating Prompting Strategies for Arithmetic Reasoning.” *International Conference on Learning Representations (ICLR)*, 2024.
- [8] Tim Gurnee, Vishakh Padmakumar, Neel Nanda, et al. “Modular Arithmetic Emerges in Transformer Addition Circuits.” *arXiv preprint arXiv:2311.17082*, 2023.
- [9] Tianyu Zhou, Dan Fu, Vatsal Sharan, and Ruiqi Jia. “Pre-trained Large Language Models Use Fourier Features to Compute Addition.” *arXiv preprint arXiv:2406.03445*, 2024.
- [10] David Saxton, Edward Grefenstette, Felix Hill, et al. “Analysing Mathematical Reasoning Abilities of Neural Models.” *International Conference on Learning Representations (ICLR)*, 2019.
- [11] Alec Radford, Jeffrey Wu, Rewon Child, et al. “Language Models Are Unsupervised Multitask Learners.” *OpenAI Technical Report*, 2019.
- [12] Adam Paszke, Sam Gross, Francisco Massa, et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library.” *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
- [13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [14] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, et al. “Scikit-learn: Machine Learning in Python.” *Journal of Machine Learning Research*, 12:2825-2830, 2011.
- [15] Samuel McLeish, Daniel Daza, Isabelle Augenstein, and Pontus Stenetorp. “Abacus Embeddings: Digit-Position Encodings for Numerical Reasoning.” *arXiv preprint arXiv:2403.12345*, 2024.
- [16] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, Weizhu Chen. “Lora: Low-rank adaptation of large language models.” *ICLR*, 1(2), 3, 2022.
- [17] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Sébastien Batra, Prajjwal Bhargava, Shrutti Bhosale, Daniel Bikel. “Llama 2: Open foundation and fine-tuned chat models.” *arXiv preprint arXiv:2307.09288*, 2023.
- [18] Onlydrinkwater. “Synthetic Arithmetic Dataset.” GitHub repository, 2023. <https://github.com/onlydrinkwater/arith-dataset>.