

Large Language Model Applications

LLMs and Numbers

+

Detection of Generated Text

Expectations for project and final week
of class

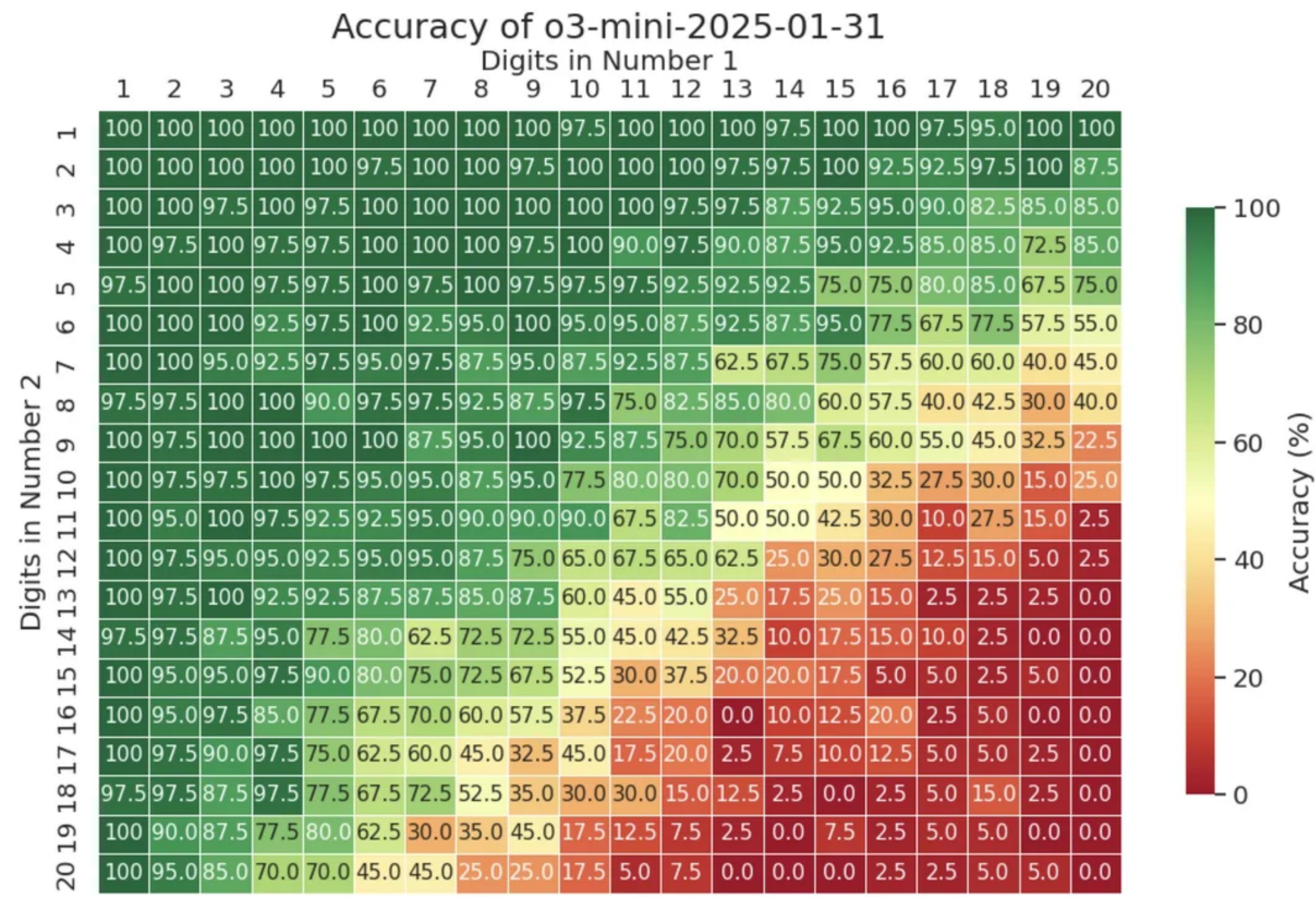
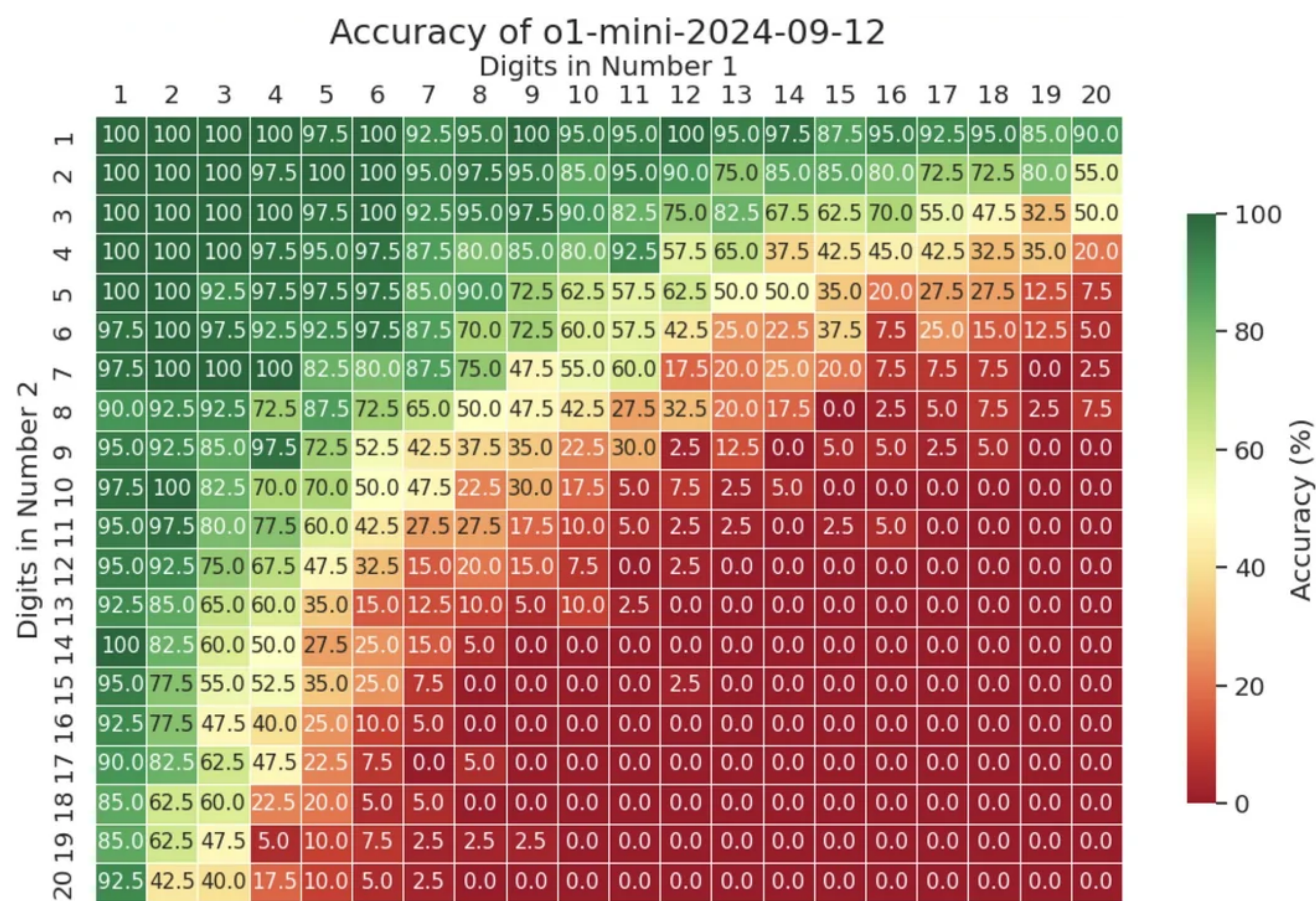
Project last stretch

- All students:
 - Submit a 10-14 minute video presentation on your project. **These are due Monday, April 20 at 11:59 PM, no exceptions.**
 - Write a project report. **These are due Thursday, April 30 at 11:59 PM. The usual late penalty applies.**
 - Complete participation activities in class on Tuesday, April 21 and Thursday, April 23. Rather than PollEverywhere, participation will involve you submitting feedback for 2+ other projects one each of the data.
- Applied track: demo session on Tuesday, April 21
 - Present a live demo to your classmates and instructors. We will use GHC 6115 for this class.
- Research track: spotlight talks on Thursday, April 23
 - ~6 projects will be selected to give live talks to the class.

LLMs and numbers

Can language models do multiplication?

Language models have historically been bad at arithmetic.



The “weirder” the number, the more LLMs struggle with math.

Question 1

Judy teaches 5 dance classes every day on the weekdays and 8 classes on Saturday. If each class has 15 students and she charges \$15 per student, how much money does she make in 1 week?

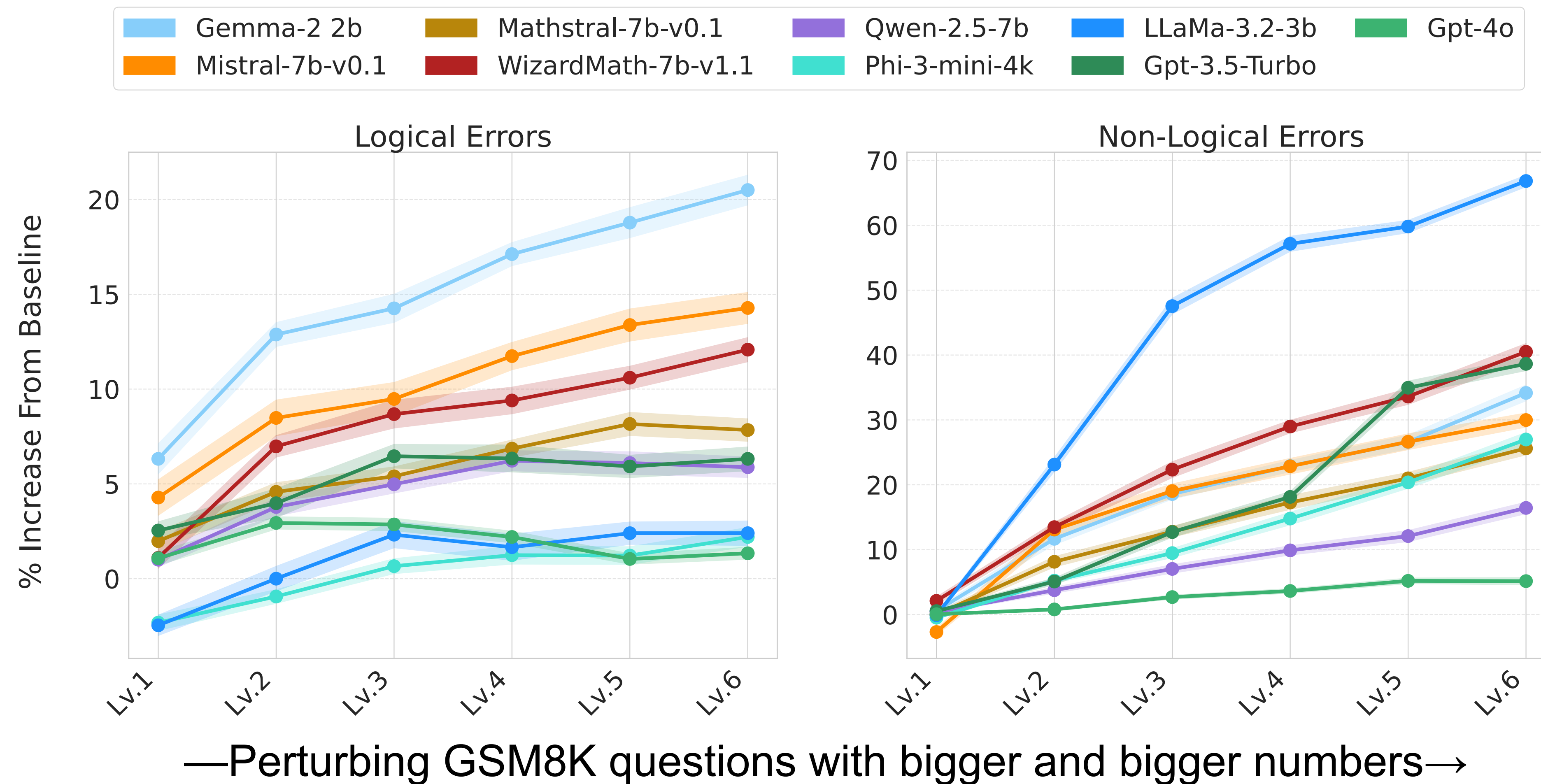
Question 2

Judy teaches 3,124,213 dance classes every day on the weekdays and 7,832,129 classes on Saturday. If each class has 25 students and she charges \$35 per student, how much money does she make in 1 week?



Would you expect a human highschooler who can easily solve Question 1 to also be able to solve Question 2?

The “weirder” the number, the more LLMs struggle with math.



$$456.9 \times 343.2$$

The human approach



How would you approach solving 456.9×343.2 ?

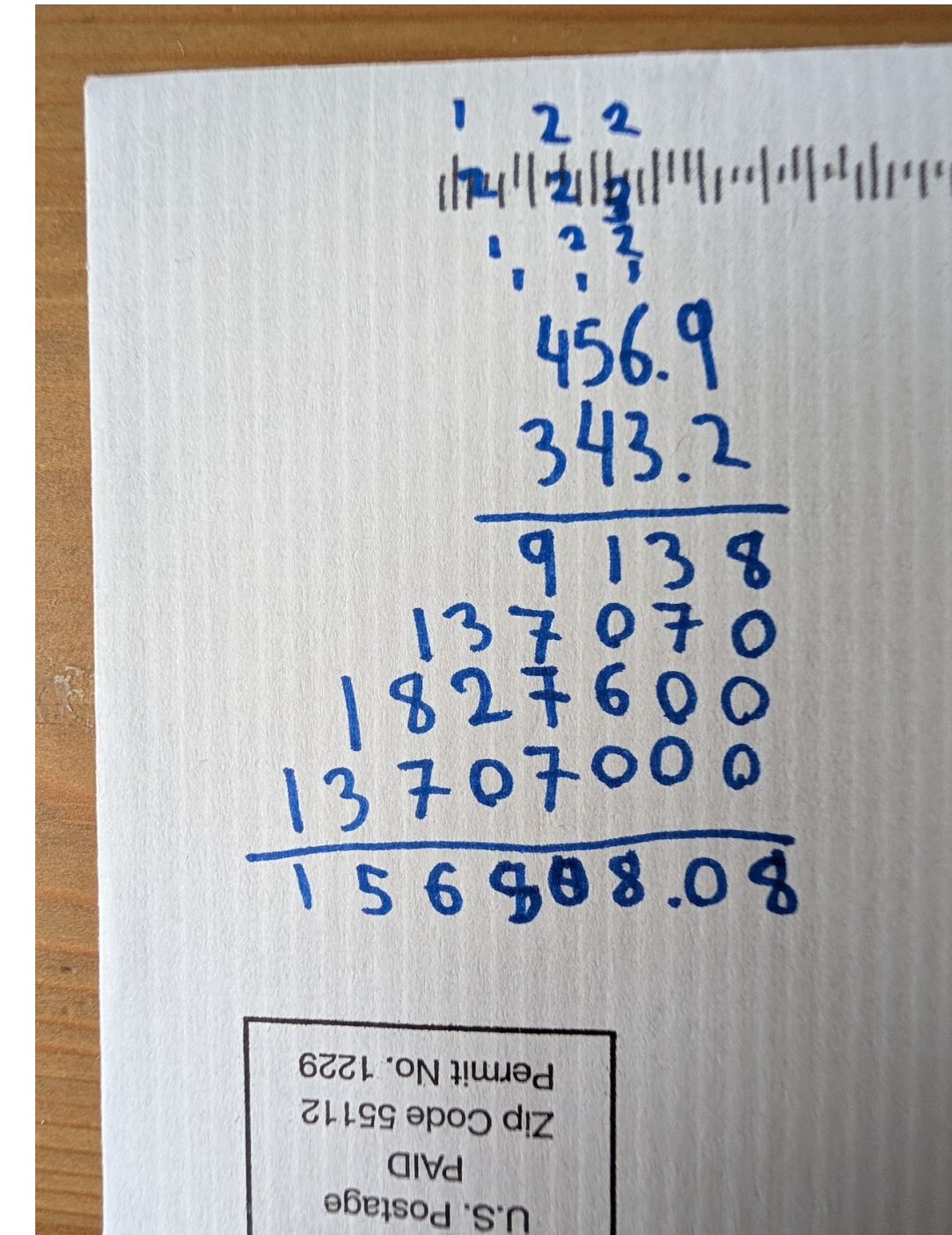
- A. Do it in your head (i.e. mental arithmetic)
- B. Input it into a calculator (or calculator app)
- C. Solve using pencil & paper
- D. Ask your favorite LLM

$$456.9 \times 343.2$$

The human approach

How would you approach solving 456.9×343.2 ?

- A. Do it in your head (i.e. mental arithmetic)
- B. Input it into a calculator (or calculator app)
- C. Solve using pencil & paper
- D. Ask your favorite LLM



```
MacBook-Pro-1389:~ daphne$ python3
Python 3.9.6 (default, Dec 2 2025, 07:27:58)
[Clang 17.0.0 (clang-1700.6.3.2)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 456.9 * 343.2
156808.08
>>> |
```

Let's use LLMs to solve 456.9×343.2

LLaMA 3.3 70B

To calculate the product of 456.9 and 343.2, I'll multiply the numbers.

$$456.9 \times 343.2 = 156,831.28$$

So the result is 156,831.28.

Let's use LLMs to solve 456.9×343.2

glm-5

Attempt 1

We are asked: "What is 456.9×343.2 ?" So we need to compute the product of two numbers: 456.9 and 343.2.
We can compute this exactly. Both numbers have one decimal place. So the product will have two decimal places. We can compute 4569×3432 and then divide by 10000.
Alternatively, we can do:
 $456.9 \times 343.2 = (4569/10) \times (3432/10) = (4569 \times 3432)/100$.
So let's compute 4569×3432 .
We can break it down:
 $4569 \times 3432 = 4569 \times (3000 + 400 + 30 + 2) = 45693000 + 4569400 + 456930 + 45692$.
Compute step by step:
 $4569 \times 3000 = 4569 \times 3 \times 1000 = 13707 \times 1000 = 13,707,000$.
 $4569 \times 400 = 4569 \times 4 \times 100 = 18276 \times 100 = 1,827,600$.
 $4569 \times 30 = 4569 \times 3 \times 10 = 13707 \times 10 = 137,070$.
 $4569 \times 2 = 9,138$.
Now sum them:
 $13,707,000$
 $1,827,600 = 15,534,600$
 $137,070 = 15,671,670$
 $9,138 = 15,680,808$.
So $4569 \times 3432 = 15,680,808$.
Then the product of the original numbers is $15,680,808 / 100 = 156,808.08$? Wait careful: $456.9 = 4569/10$, $343.2 = 3432/10$, so product = $(45693432)/(1010) = 15,680,808 / 100 = 156,808.08$.
But we should check if that is correct. Alternatively, we can compute directly:
 $456.9 \times 343.2 = 456.9 \times 343 + 456.9 \times 0.2 =$ maybe simpler.
Let's verify with a calculator mentally? I trust the multiplication.
Double-check: 4569×3432 .
Another method:
 $4569 \times 3432 = (4500+69) \times (3400+32)$ but maybe not.
We can do the multiplication in columns:

Attempt 2

Copy code
$$\begin{array}{r} 4569 \\ \times 3432 \\ \hline \end{array}$$

First, $4569 \times 2 = 9138$.
Then, $4569 \times 30 = 137070$ (shift one place).
Then, $4569 \times 400 = 1827600$ (shift two places).
Then, $4569 \times 3000 = 13707000$ (shift three places).
Add them:
 0009138
 0137070 (align: $9138 + 137070$? Actually, we align by place):
Write numbers with proper alignment:

Attempt 3

Copy code
$$\begin{array}{r} 4569 \\ \times 3432 \\ \hline 9138 \\ 137070 \\ 1827600 \\ +13707000 \\ \hline \end{array}$$

Sum: start from right:
Units: 8
Tens: $3+7=10$, write 0, carry 1.
Hundreds: $1+9+7+6=23$ plus carry 1 = 24, write 4, carry 2.
Thousands: $2+1+3+7+2=15$ plus carry 2 = 17, write 7, carry 1.
Ten thousands: $1+1+8+7+0=17$ plus carry 1 = 18, write 8, carry 1.
Hundred thousands: $1+2+2+3=8$ plus carry 1 = 9, write 9.
Millions: $1+8=9$, write 9.
Ten millions: 1, write 1.
So we get: 15,680,808. Yes.
Then $15,680,808$ divided by 100 = 156,808.08.
So the answer is 156,808.08.
But we should present it clearly.
Thus, the final answer is 156,808.08.

Let’s use LLMs to solve 456.9×343.2

Qwen 3 30B Thinking

Okay, let's see. I need to calculate 456.9 multiplied by 343.2. Hmm, multiplying decimals can be tricky, but I remember that one way to do it is to ignore the decimals first, multiply the numbers as if they were whole numbers, and then adjust for the decimal places at the end. Let me try that.

First, let's figure out how many decimal places there are in each number. 456.9 has one decimal place, and 343.2 also has one decimal place. So together, that's 1 + 1 = 2 decimal places. So after I multiply the whole numbers, I'll need to move the decimal point two places to the left.

Now, let's convert both numbers to whole numbers by removing the decimals. 456.9 becomes 4569 (since we move the decimal one place to the right), and 343.2 becomes 3432 (same thing). Now I need to calculate 4569 x 3432.

This might be a bit complicated, but maybe I can break it down. Let's use the standard multiplication algorithm. Let me write it out step by step.

First, multiply 4569 by 2 (the units digit of 3432). 4569 x 2 = 9138.

Next, multiply 4569 by 30 (the tens digit of 3432, which is 3 x 10). 4569 x 30 = 4569 x 3 x 10. 4569 x 3 is 13707, so times 10 is 137070.

Then, multiply 4569 by 400 (the hundreds digit of 3432, which is 4 x 100). 4569 x 400 = 4569 x 4 x 100. 4569 x 4 is 18276, so times 100 is 1827600.

Finally, multiply 4569 by 3000 (the thousands digit of 3432, which is 3 x 1000). Wait, hold on, 3432 is 3 thousands, 4 hundreds, 3 tens, and 2 units. So 3432 = 3000 + 400 + 30 + 2. So the thousands digit is 3, so 3 x 1000 = 3000.

So 4569 x 3000 = 4569 x 3 x 1000 = 13707 x 1000 = 13707000.

Wait, let me confirm the digits of 3432: the first digit from the right is units (2), then tens (3), hundreds (4), thousands (3). So yes, 3 (thousands), 4 (hundreds), 3 (tens), 2 (units). So 3432 = 3*1000 + 4*100 + 3*10 + 2*1.

So the multiplication steps are:

4569 x 2 = 9138

4569 x 30 = 137070

4569 x 400 = 1827600

4569 x 3000 = 13707000

Now, add all these together:

First, 9138 + 137070 = 146208

Then, 146208 + 1827600 = 1973808

Then, 1973808 + 13707000 = 15680808

Wait, let's check that addition again.

13707000 (from 3000)

+ 1827600 (from 400)

= 13707000 + 1827600 = 15534600

Then + 137070 (from 30) = 15534600 + 137070 = 15671670

Then + 9138 (from 2) = 15671670 + 9138 = 15680808

Yes, that's correct. So 4569 x 3432 = 15,680,808.

Now, remember that we had two decimal places in the original numbers (one from each), so we need to move the decimal point two places to the left in 15,680,808.

15,680,808 with two decimal places is 156808.08? Wait, no. Wait, 15,680,808 is a whole number. If we have two decimal places, we need to count two digits from the right.

Wait, 15,680,808 is 15680808. Let's write it as 15680808.0

Moving the decimal two places left:156808.08

Wait, let's check: 456.9 is 4569/10, 343.2 is 3432/10, so their product is (4569 x 3432)/(10 x 10) = 15680808 / 100 = 156808.08

Yes, that makes sense. So 456.9 x 343.2 = 156,808.08

Let me verify with another method to be sure. Maybe using approximate values.

456.9 is approximately 457, and 343.2 is approximately 343. So 457 x 343.

Let's compute 457 x 343.

457 x 300 = 137,100

457 x 40 = 18,280

457 x 3 = 1,371

Adding those together: 137,100 + 18,280 = 155,380 + 1,371 = 156,751

Our actual product is 156,808.08, which is a bit higher than 156,751, which makes sense because we rounded up 456.9 to 457 (which is 0.1 more) and 343.2 to 343 (which is 0.2 less). So the overestimation from 457 vs 456.9 might be more than the underestimation from 343 vs 343.2. Let's see:

The difference between 457 and 456.9 is 0.1, so the extra product is 0.1 x 343.2 = 34.32

The difference between 343.2 and 343 is 0.2, so the product is 456.9 x 0.2 = 91.38

So the total approximate product is 156,751 (from 457x343) + 34.32 (from the extra 0.1 in the first number) - 91.38 (from the missing 0.2 in the second number) = 156,751 + 34.32 - 91.38 = 156,751 - 57.06 = 156,693.94

Wait, but that's not matching our previous result. Maybe my approximation is off. Alternatively, maybe using another method.

Alternatively, use the distributive property:

456.9 x 343.2 = 456.9 x (300 + 40 + 3 + 0.2) = 456.9x300 + 456.9x40 + 456.9x3 + 456.9x0.2

Calculate each term:

456.9 x 300 = 456.9 x 3 x 100 = 1370.7 x 100 = 137,070

456.9 x 40 = 456.9 x 4 x 10 = 1827.6 x 10 = 18,276

456.9 x 3 = 1,370.7

456.9 x 0.2 = 91.38

Now add them all together:

137,070 + 18,276 = 155,346

155,346 + 1,370.7 = 156,716.7

156,716.7 + 91.38 = 156,808.08

Yes! That's the same result as before. So that confirms it. So the product is 156,808.08.

I think that's correct. Let me just check with a calculator method. If I take 456.9 * 343.2:

First, 456.9 * 300 = 137,070

456.9 * 40 = 18,276

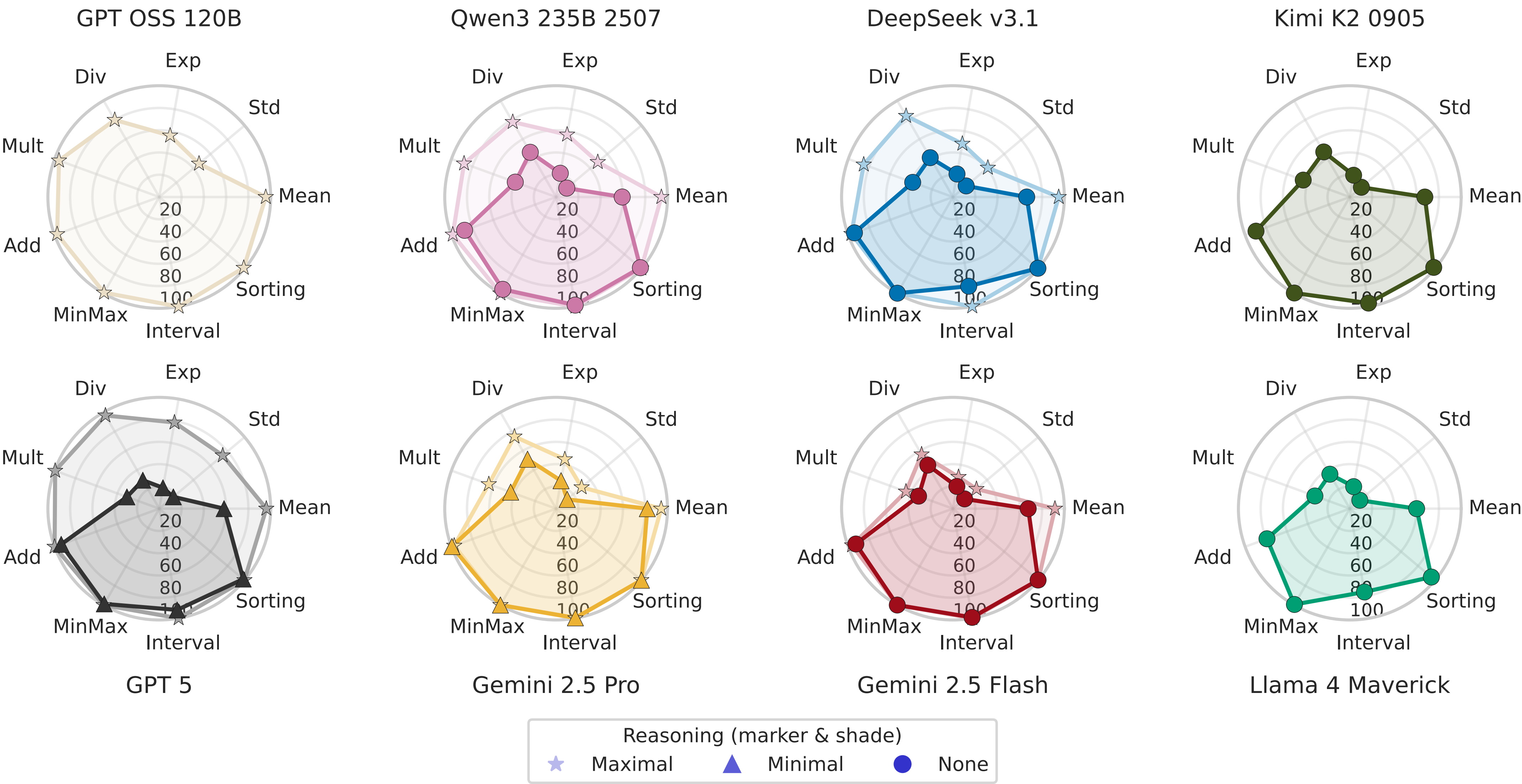
456.9 * 3 = 1,370.7

456.9 * 0.2 = 91.38

Adding all those up: 137070 + 18276 = 155346; 155346 + 1370.7 = 156716.7; 156716.7 + 91.38 = 156808.08. Perfect, that's consistent.

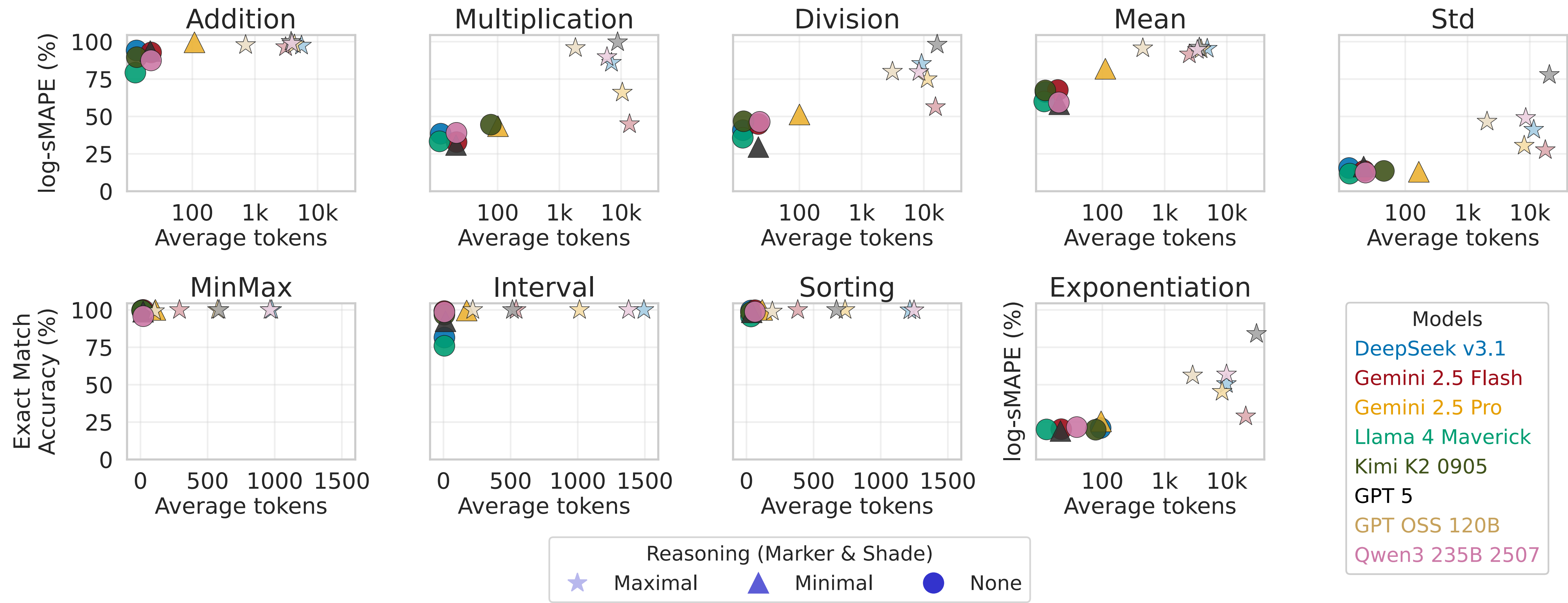
So the answer should be 156,808.08.
</think>

Does reasoning solve arithmetic for LLMs?



Kreitner et al. “Efficient numeracy in language models through single-token number embeddings.” 2025.

Does reasoning solve arithmetic for LLMs?



Let's use LLMs to solve 456.9×343.2

Gemini and GPT-5

What is 456.9×343.2 ?

Hide code

Analysis

Python

```
print(456.9 * 343.2)
```

Code output

156808.08

The product of 456.9 and 343.2 is 156,808.08.

What is 456.9×343.2 ?

ChatGPT Instruments

456.9×343.2

156808

Give feedback

So, $456.9 \times 343.2 = 156,808$.

★ Why might we care about language models understanding numbers, when we can just have them call tools instead?

Why might we care about language models understanding numbers, when we can just have them call tools instead?

GSM8K

Kaityn has 20 red hats and 24 blue hats. Her friend Zola has $\frac{4}{5}$ times as many red hats as she has and twice the number of blue hats. If they combine all the hats together and share them equally between themselves, calculate the number of hats each gets.

MATH Benchmark

When $1 + 7 + 7^2 + \cdots + 7^{2004}$ is divided by 1000, a remainder of N is obtained. Determine the value of N .

Math Olympiad

Find all positive integers n for which there exist non-constant integer-coefficient polynomials $P(x)$ and $Q(x)$ such that $P(x)Q(x) = x^n - 729$.

There's currently big emphasis on LLMs' math problem-solving skills, but numerical understanding is valuable beyond just math problems.

Why might we care about language models understanding numbers, when we can just have them call tools instead?

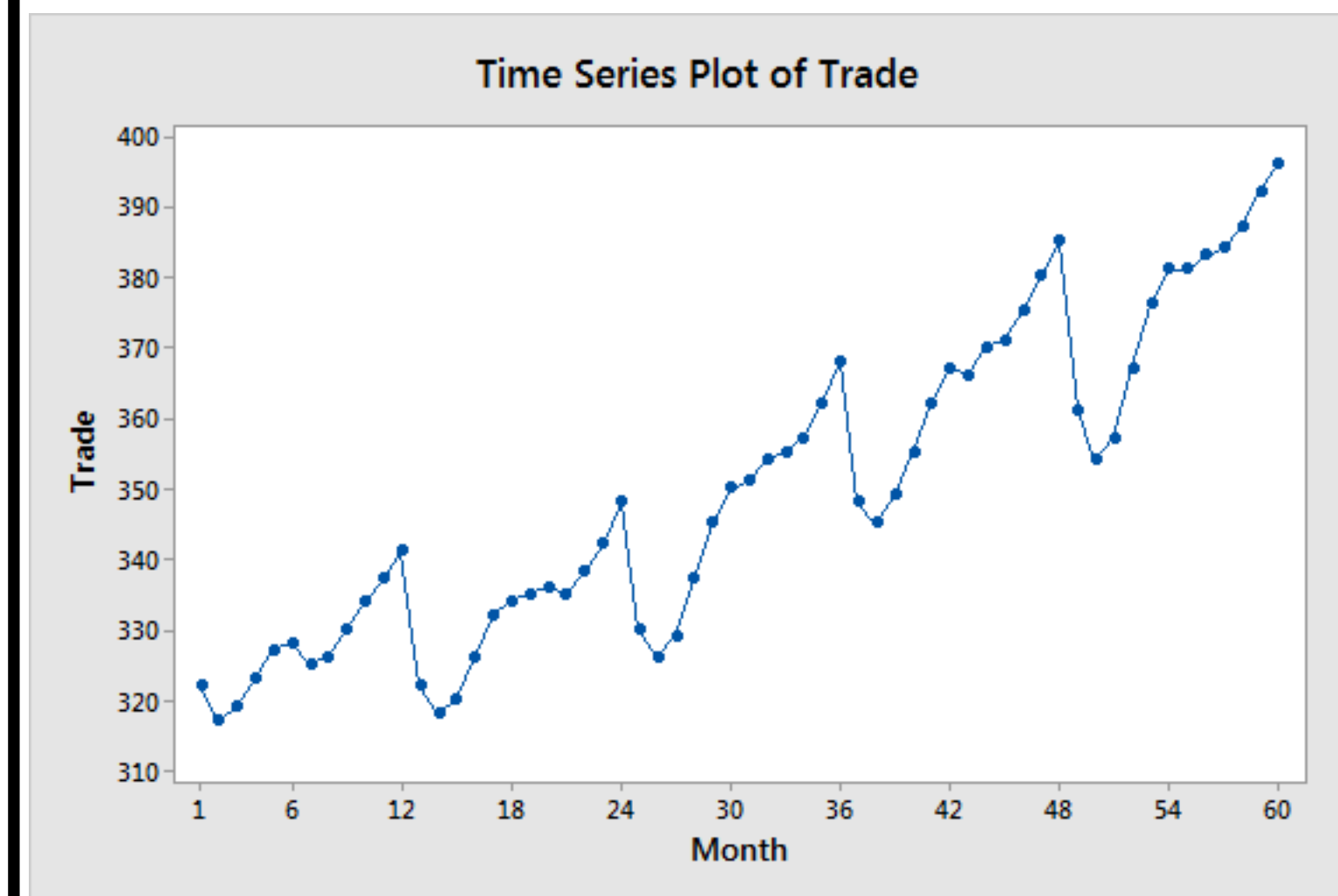
Weather Prediction

Monday, April 6, 2026. In Pittsburgh, the current nighttime condition is clear with periodic clouds with a temperature of 43°F. The wind is blowing from the west at 6 mph, and humidity is at 72% with no current precipitation. For the remainder of the day, the forecast indicates a low of 32°F and a 15% chance of snow tonight.

What's the high likely to be tomorrow?

Time Series Reasoning

What scenario could have produced this time series?



Classification of Multimodal data

Given the patient's temperature, blood pressure, and doctor's written notes, what is the recommended diagnoses?

Why are numbers hard for language models?

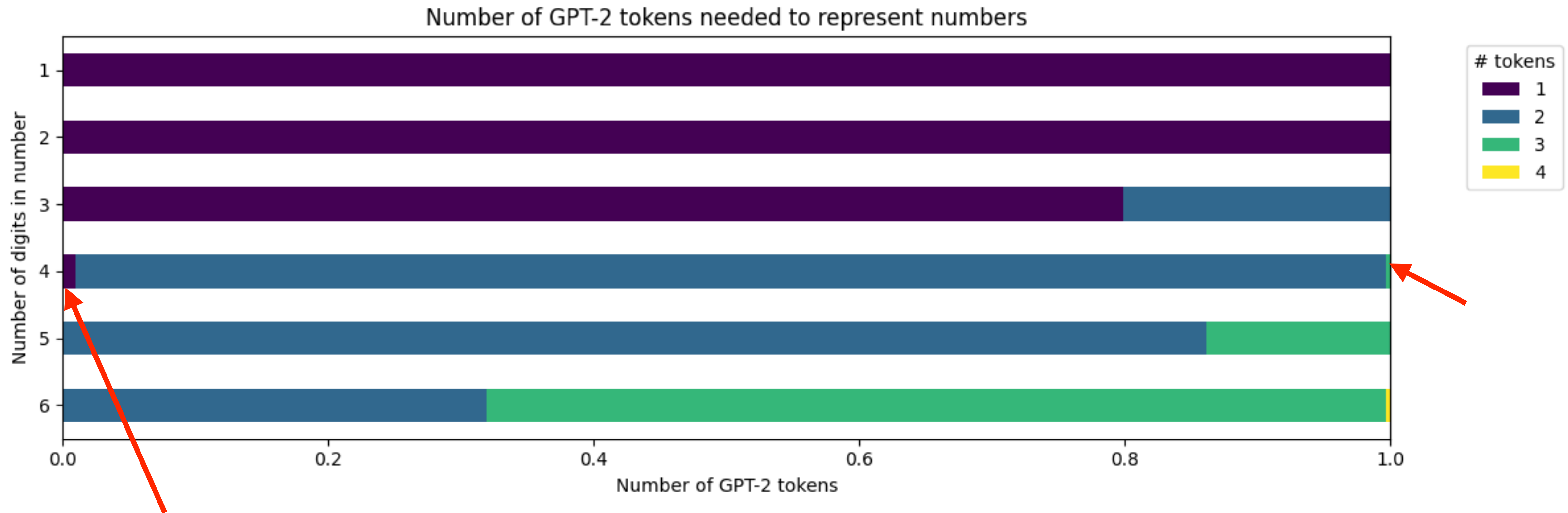
Agenda

- Tokenization methods for numbers
- Alternative methods for encoding numbers
- Does tool-use solve all problems with LLM number understanding?

Tokenization methods for numbers

How do LLMs represent numbers?

2020: Tokenizers learned how to tokenize numbers the same way they learned how to tokenize language.



1000, 1001, 1920,
1945, 2001, 6666, ...

How do LLMs represent numbers?

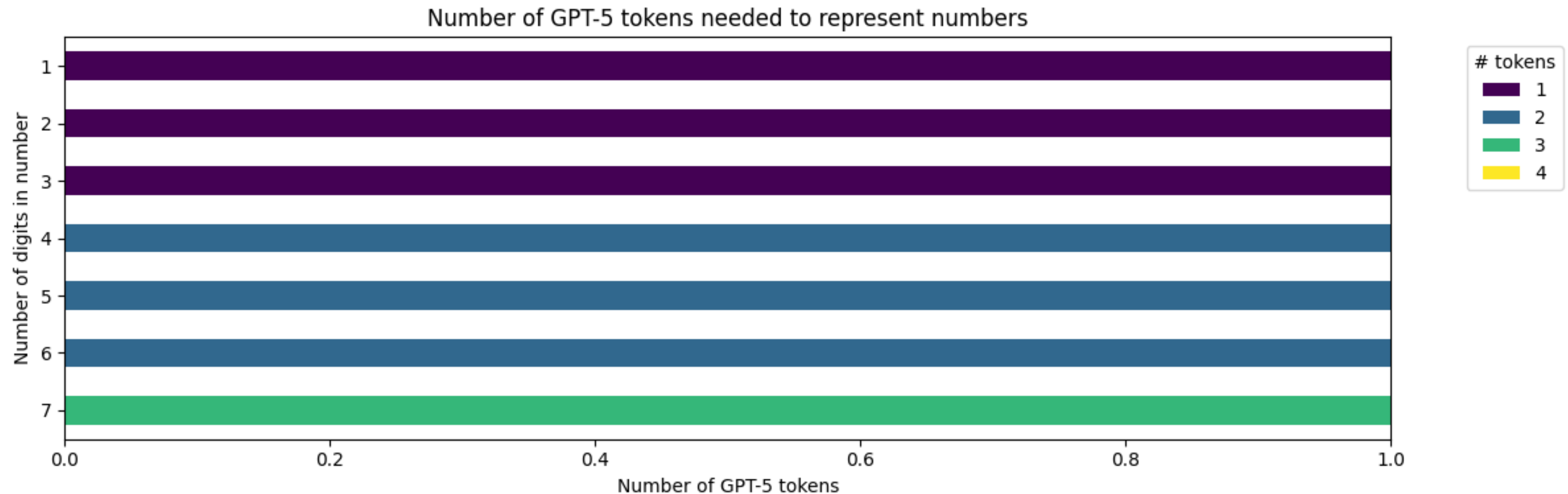
2026: GPT-5 and Llama4 approach

- All numbers between 1 and 1,000 get their own tokens.
- Split longer integers as sub-chunks of these tokens always to the left

Example: 1234567 $\rightarrow$ {123, 456, 7}

How do LLMs represent numbers?

2026: GPT-5 and Llama4 approach



Left-to-right vs right-to-left tokenization

What's weird about tokenizing $12345678 \rightarrow [123, 456, 78]$?

Left-to-right vs right-to-left tokenization

What's weird about tokenizing $12345678 \rightarrow [123, 456, 78]$?

The human approach: $12345678 \rightarrow 12,345,678$

How do LLMs represent numbers?

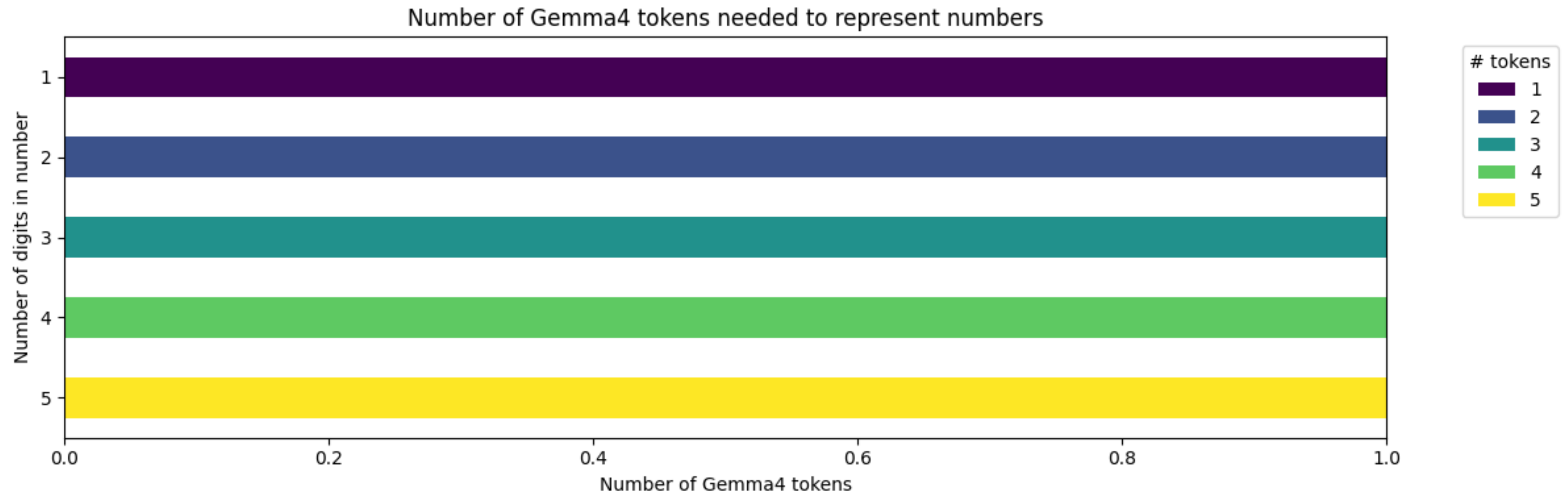
2026: Deepseek, Gemma, and Gemini approach

- All digits are tokens

Example: 1234567 $\rightarrow$ [1, 2, 3, 4, 5, 6, 7]

How do LLMs represent numbers?

2026: Deepseek, Gemma, and Gemini approach



Alternative methods for encoding numbers

Alternative methods for encoding numbers

methods we'll discuss

- xVal
- FoNE
- BitTokens

Alternative methods for encoding numbers

desiderata

- D1 – Token efficiency:** Every number is represented by a single token.
- D2 – Uniqueness:** Each value has exactly one valid encoding, with a unique inverse mapping.
- D3 – Structured:** The encoding geometry reflects numeric order and distance.
- D4 – Scale invariance:** A wide range of magnitudes can be represented with high precision.
- D5 – Normalization:** Encodings are bounded and information preserving under standard normalization functions used in language models (e.g., LayerNorm, RMSNorm).
- D6 – Numerical stability:** Representations remain accurate when using low-precision data type formats (e.g., fp8).
- D7 – Continuity:** Encodings vary smoothly with the underlying value, making them compatible with gradient-based optimization.
- D8 – Robustness:** Values can be decoded reliably even under stochastic prediction noise.
- D9 – Arithmetic:** Encodings admit learnable algorithms for core mathematical operations.

xVal

training method

“5 trials at pH 7.4”

xVal

training method

“5 trials at pH 7.4”



x_{text}	[	[NUM]	trials	at	pH	[NUM]	]
x_{num}	[	5	-	-	-	7.4	]

xVal

training method

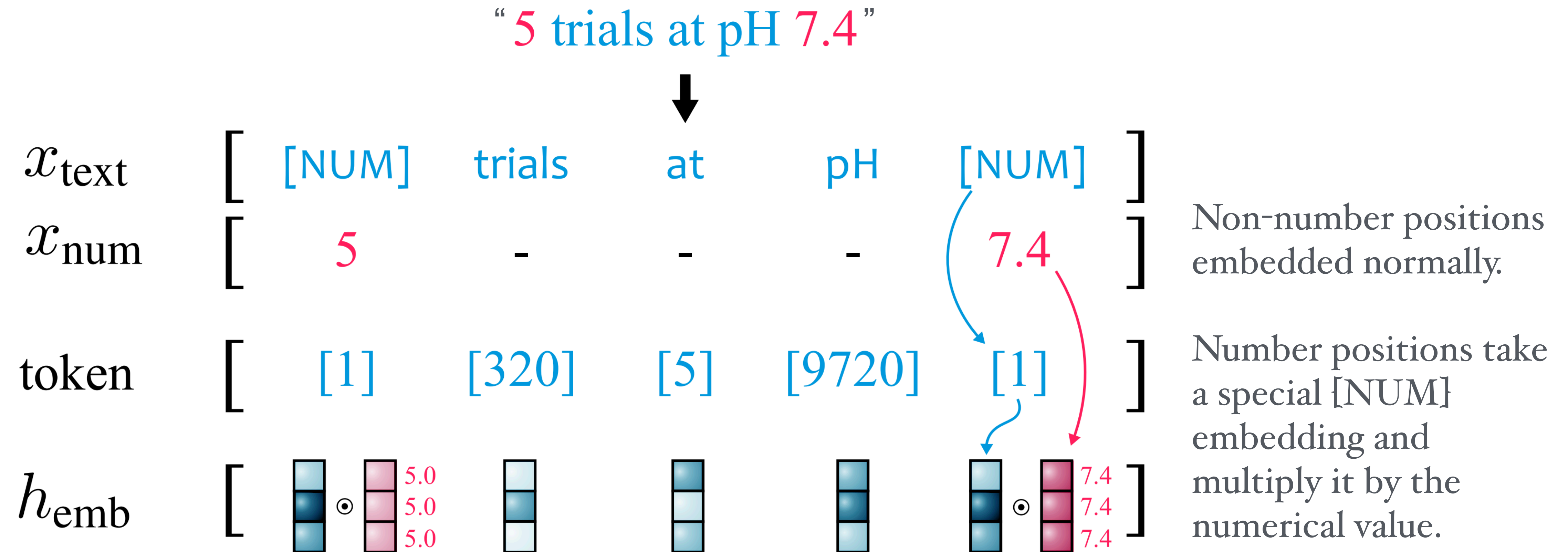
“5 trials at pH 7.4”



x_{text}	[	[NUM]	trials	at	pH	[NUM]	]
x_{num}	[	5	-	-	-	7.4	]
token	[	[1]	[320]	[5]	[9720]	[1]	]

xVal

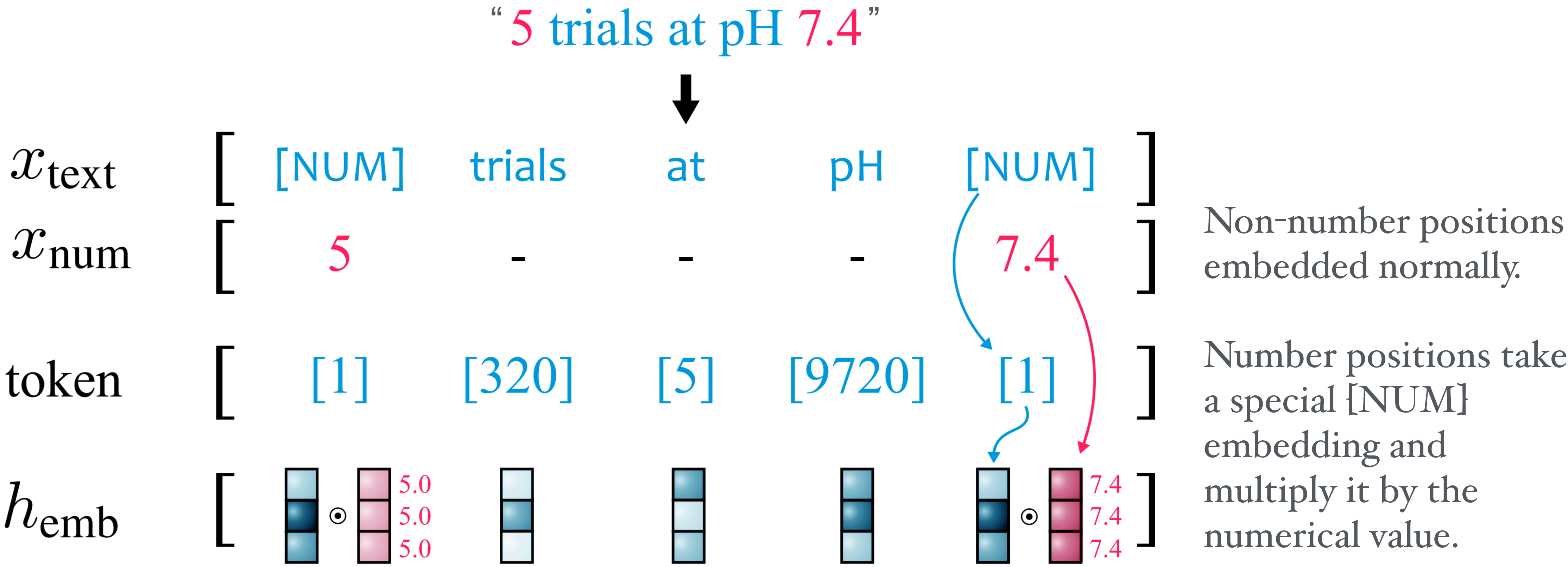
training method



xVal

What potential problem do you notice?

training method

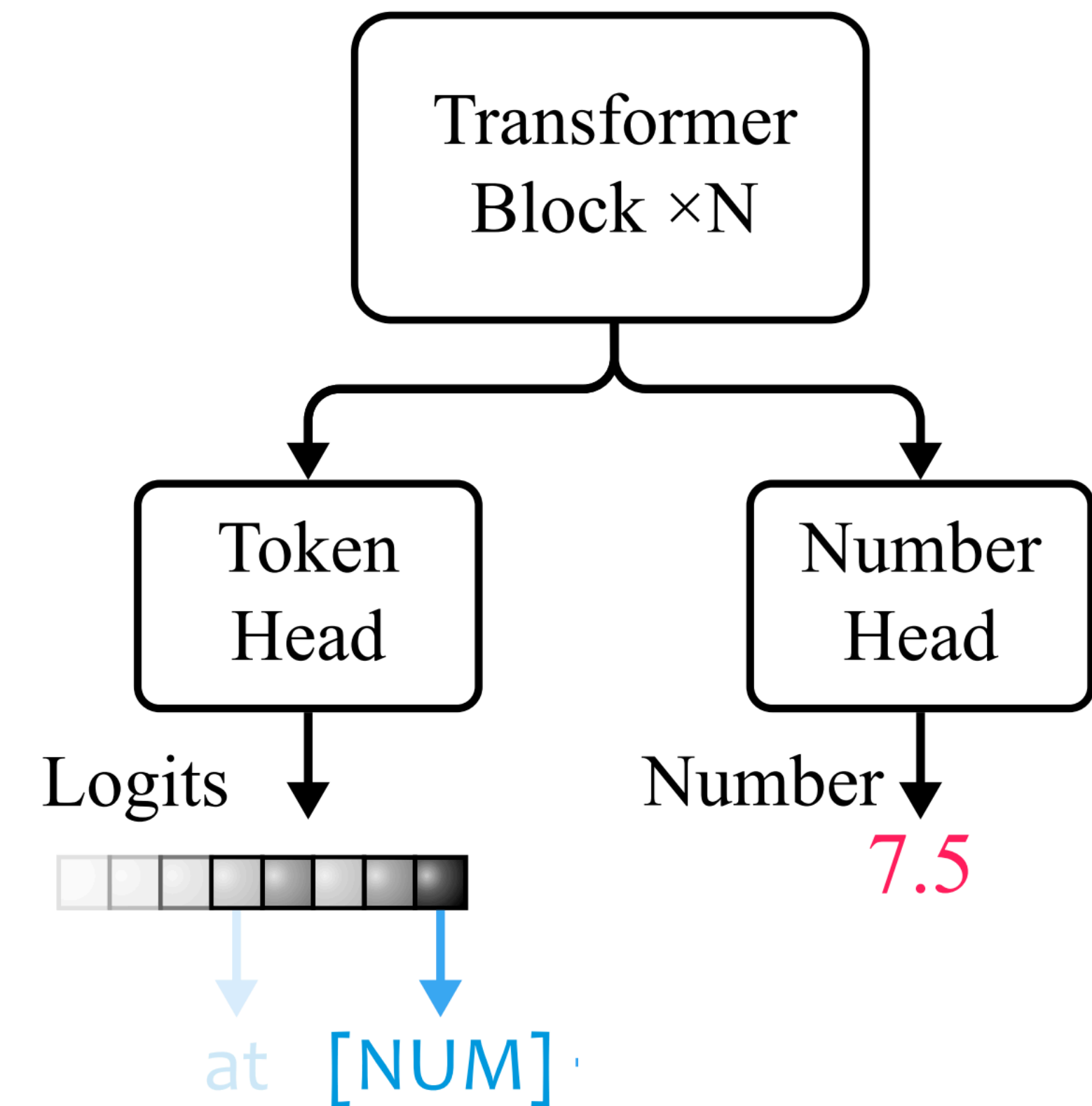


xVal

training

- In addition to the normal head that produces logits, add a second head that outputs numbers.
- Train with normal NLL loss on the number-masked sequences, using the outputs of the token head.
- For each sequence position where there is a “[NUM]” token, add an MSE loss on the predicted number.

What potential problem do you notice?



xVal

dealing with wide range of numbers

“The radius of the Earth is about 251,108,661 inches; in contrast, a SARS-CoV-2 virus has a radius of about 0.0000023622 inches.”

xVal

dealing with wide range of numbers

“The radius of the Earth is about 251,108,661 inches; in contrast, a SARS-CoV-2 virus has a radius of about 0.0000023622 inches.”

Proposed solution: k numerical embeddings, each centered around a different order of magnitude. Coefficients are always scaled to be between -1 and 1.

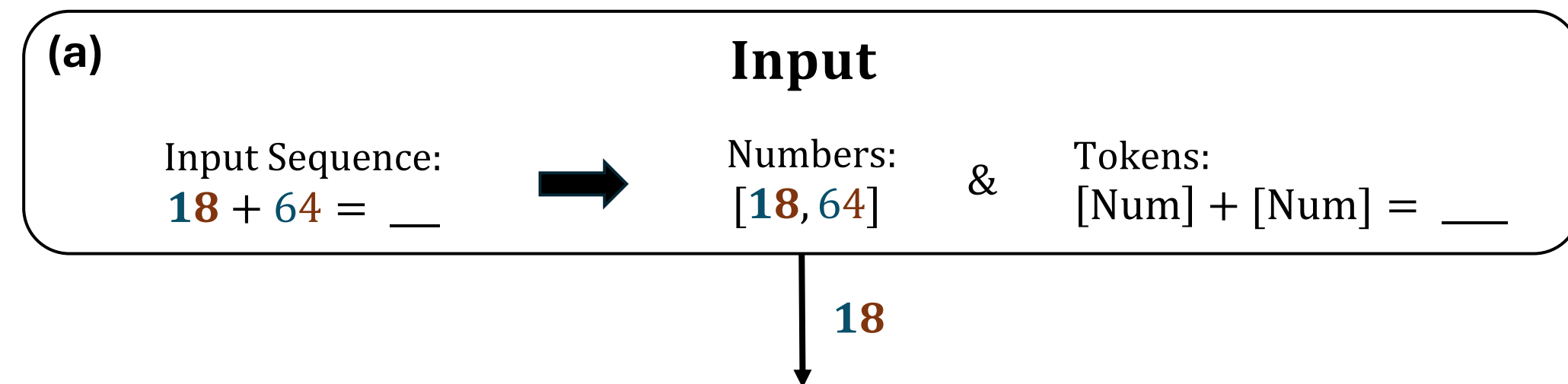
xVal

does it have the desiderata?

- ✓ - **Token efficiency:** Every number is represented by a single token.
- ✓ - **Uniqueness:** Each value has exactly one valid encoding, with a unique inverse mapping.
- ✓ - **Structured:** The encoding geometry reflects numeric order and distance.
- ✗ - **Scale invariance:** A wide range of magnitudes can be represented with high precision.
- ✗ - **Normalization:** Encodings are bounded and information preserving under standard normalization functions used in language models (e.g., LayerNorm, RMSNorm).
- ✗ - **Numerical stability:** Representations remain accurate when using low-precision data type formats (e.g., fp8).
- ✓ - **Continuity:** Encodings vary smoothly with the underlying value, making them compatible with gradient-based optimization.
- ✗ - **Robustness:** Values can be decoded reliably even under stochastic prediction noise.
- ✗ - **Arithmetic:** Encodings admit learnable algorithms for core mathematical operations.

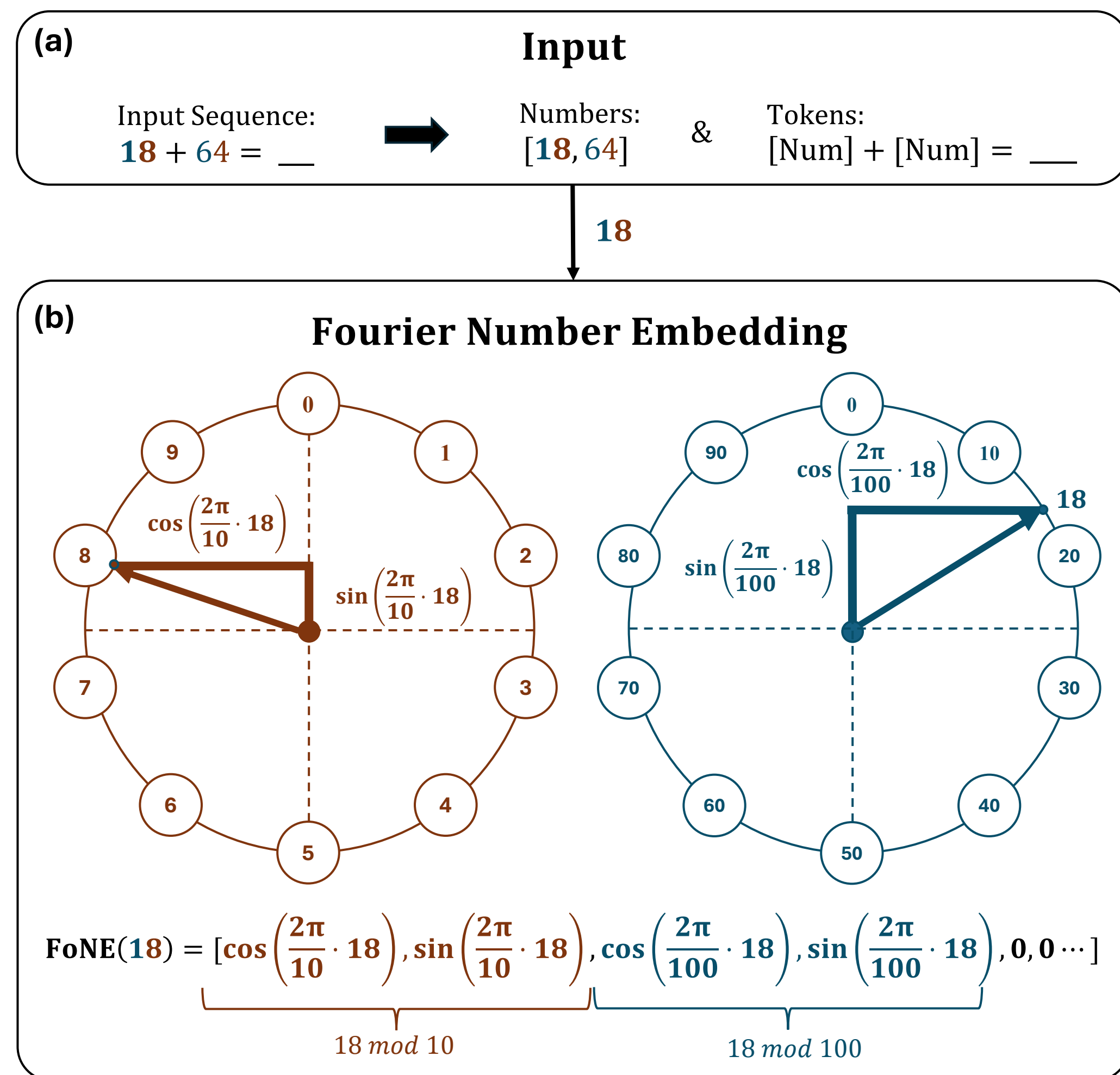
FoNE

method



FoNE

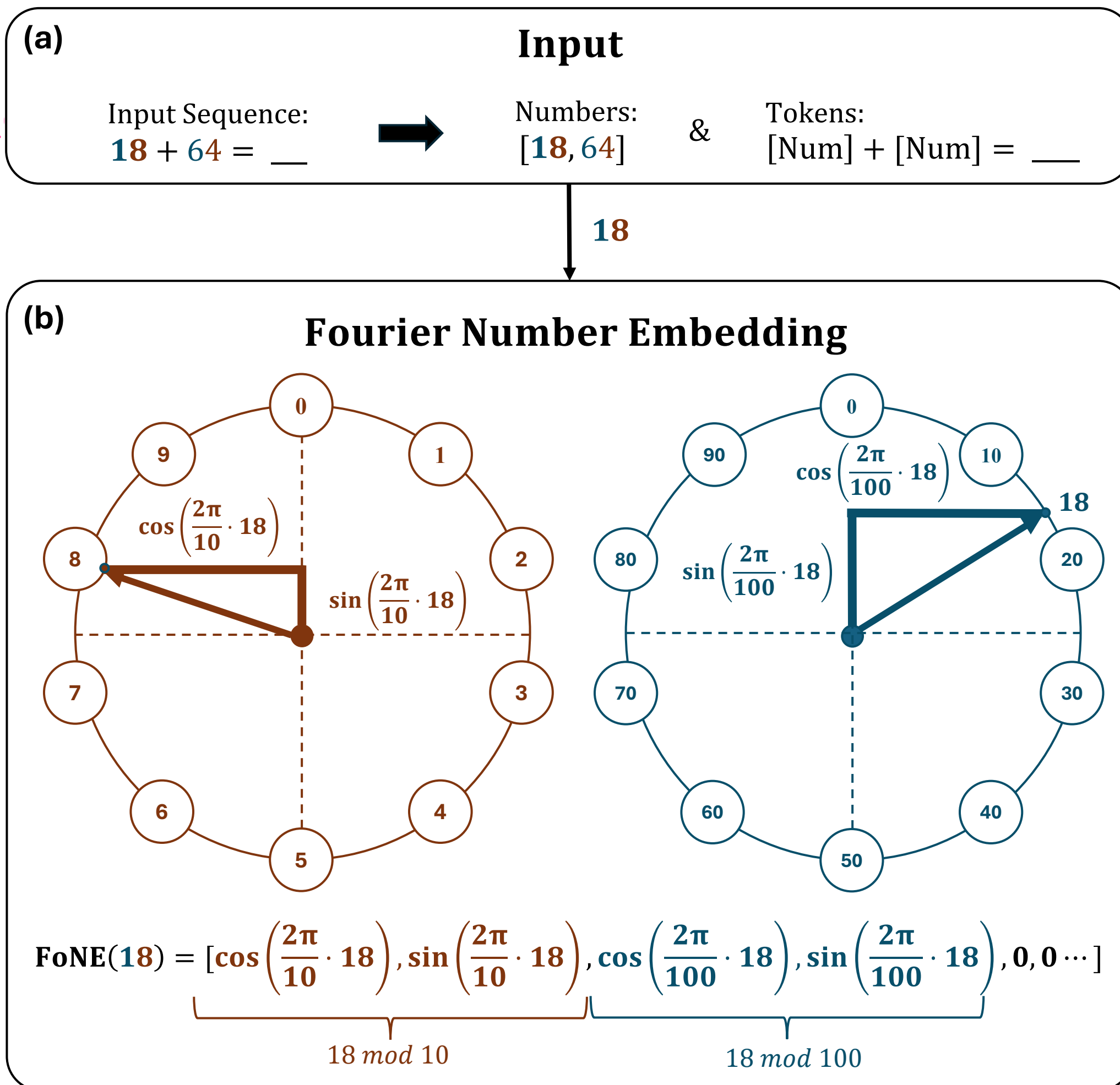
method



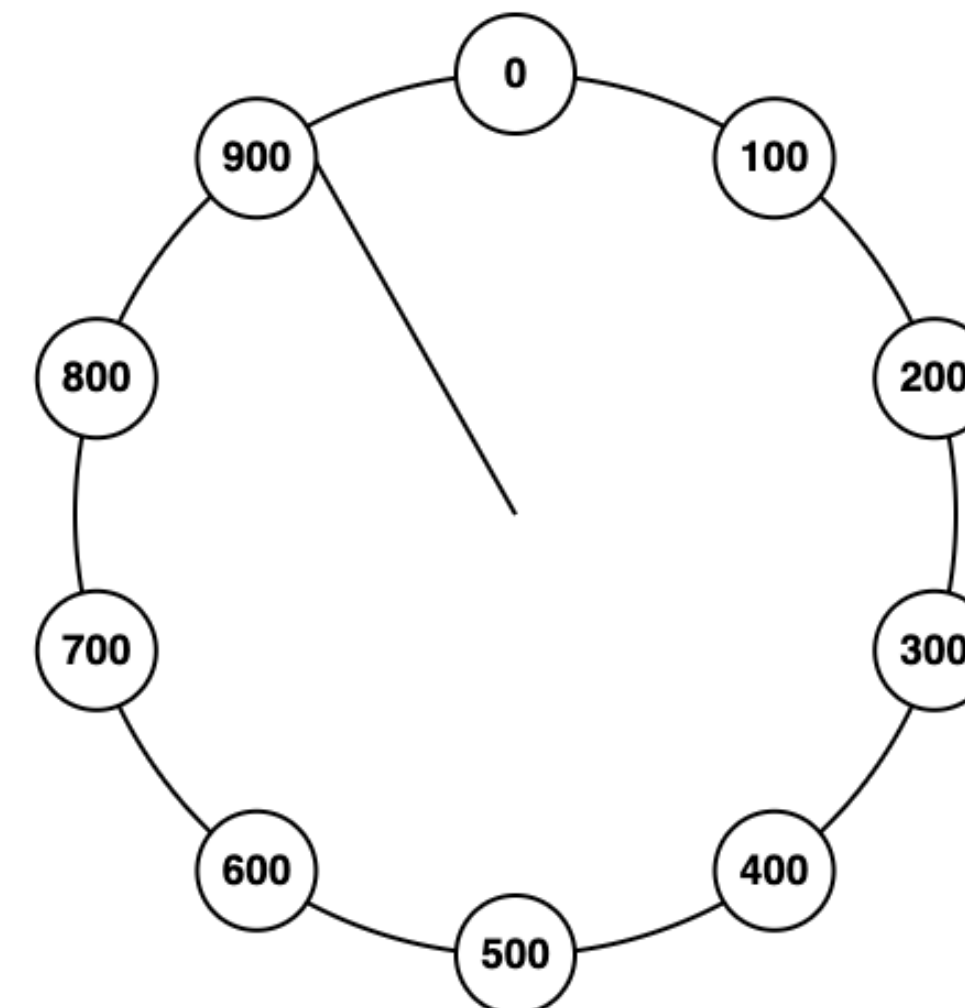
- Like xVal, each number gets mapped to a single embedding vector.
- Each digit in the number is represented by *two* positions in the embedding vector.
- In the example on the right:
 - For the **ones** position:
 - Numbers 0 to 9 are arranged around the circle. “8” is represented by the corresponding direction.
 - For **tens** position:
 - Numbers 0 to 99 are arranged around the circle. “18” is represented by the corresponding direction.
- In this way, each digit is encoded into the embedding.
- Unused part of the embedding is zero-padded.

FoNE

m

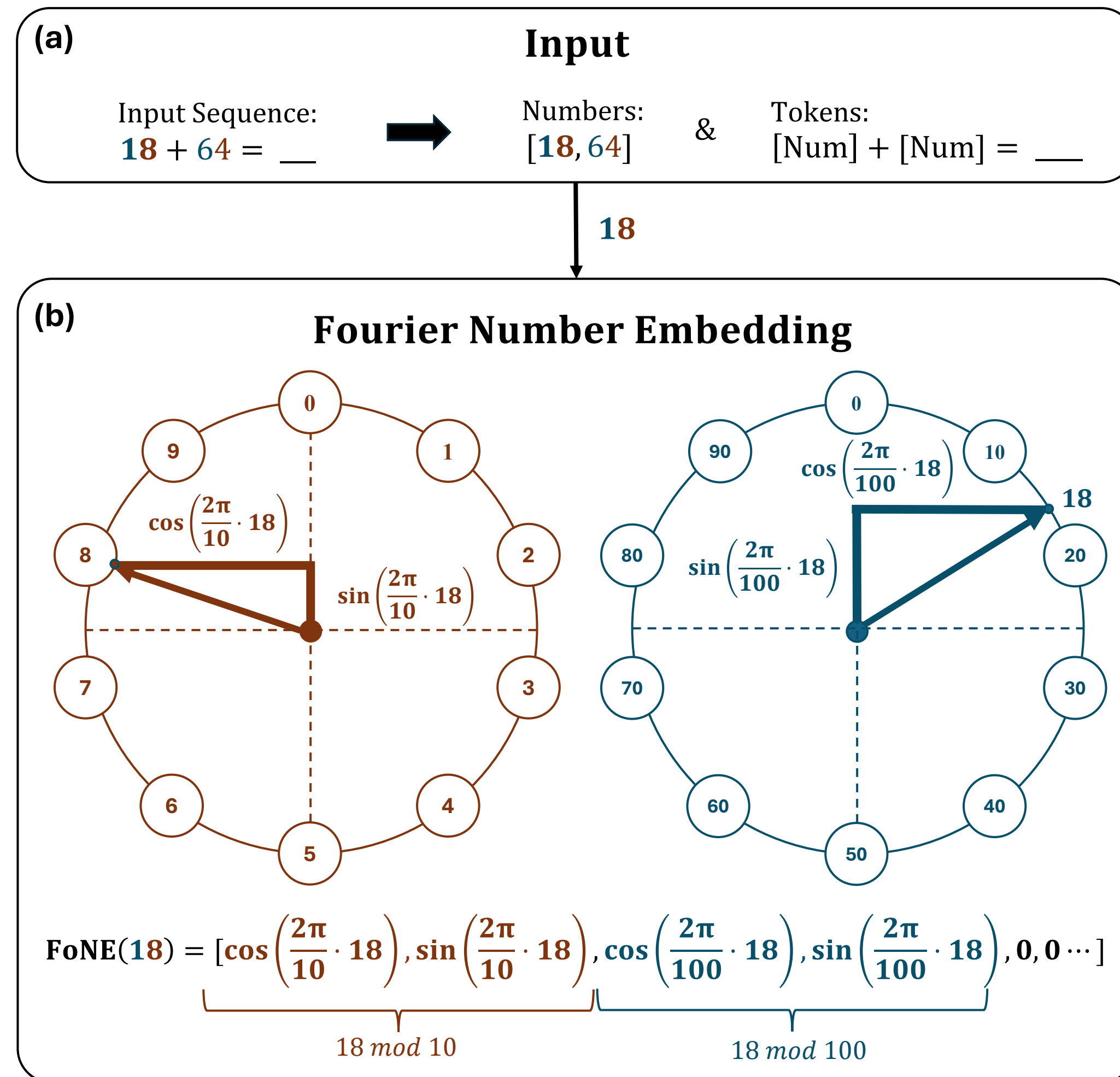


If the number were 918, we'd have another position to add to embedding.



FoNE

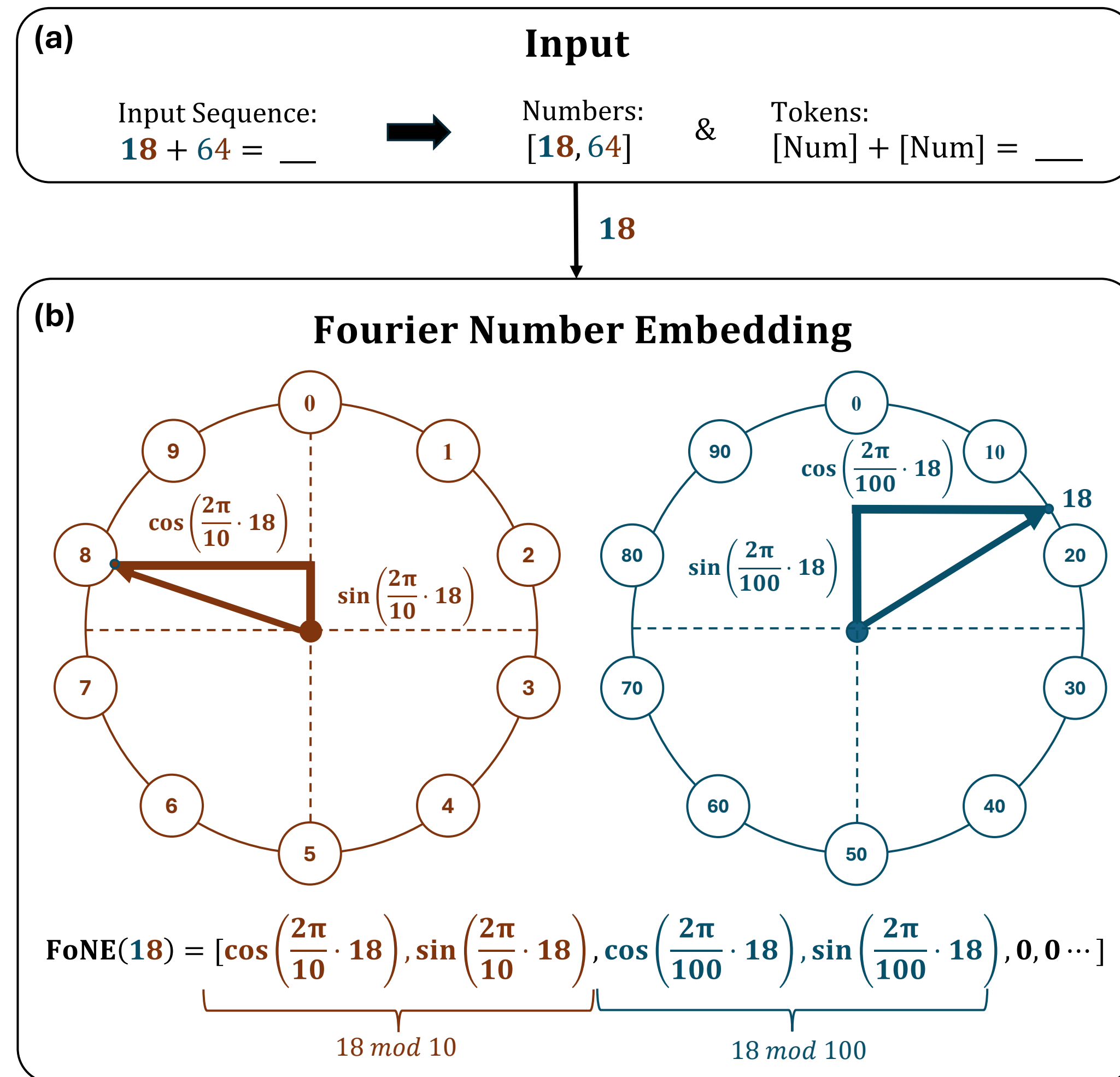
method



Suppose the embedding size is 16. What is the largest number this method could encode?

FoNE

method



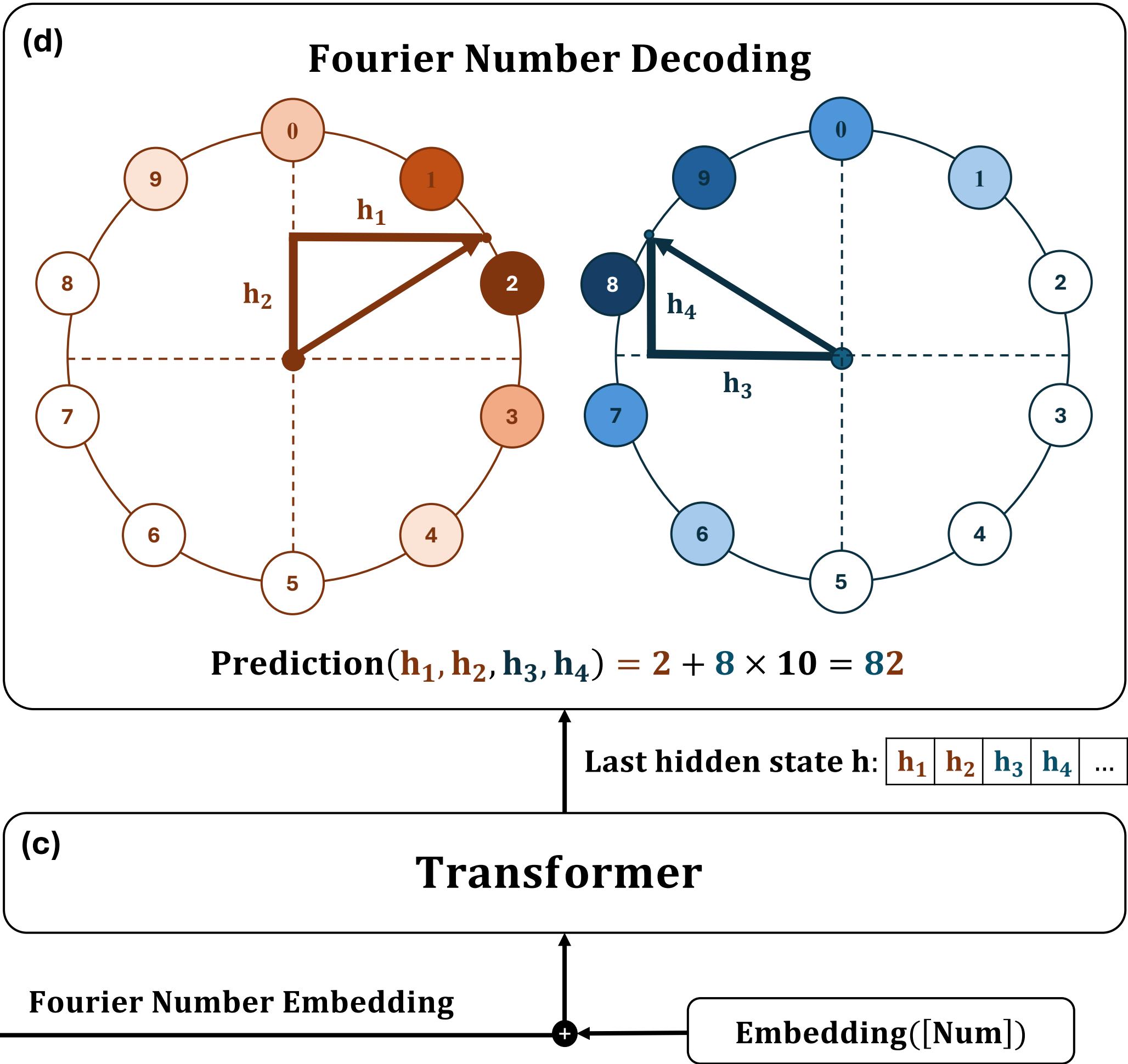
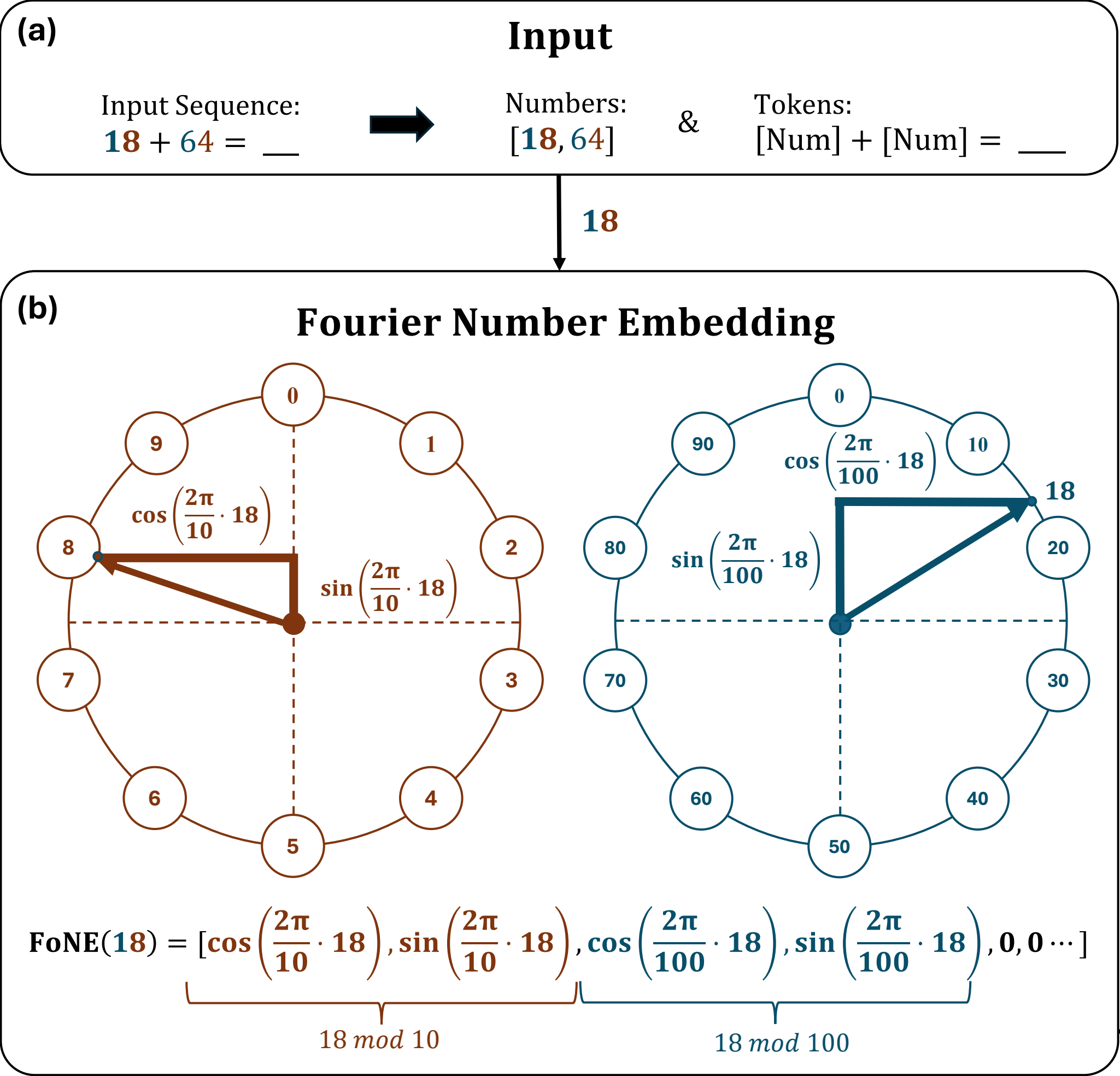
Suppose the embedding size is 16. What is the largest number this method could encode (assuming we're only encoding integers)?

A: 99,999,999

(actual method uses fraction of the embedding to encode integer and remainder to encode decimal)

FoNE

method



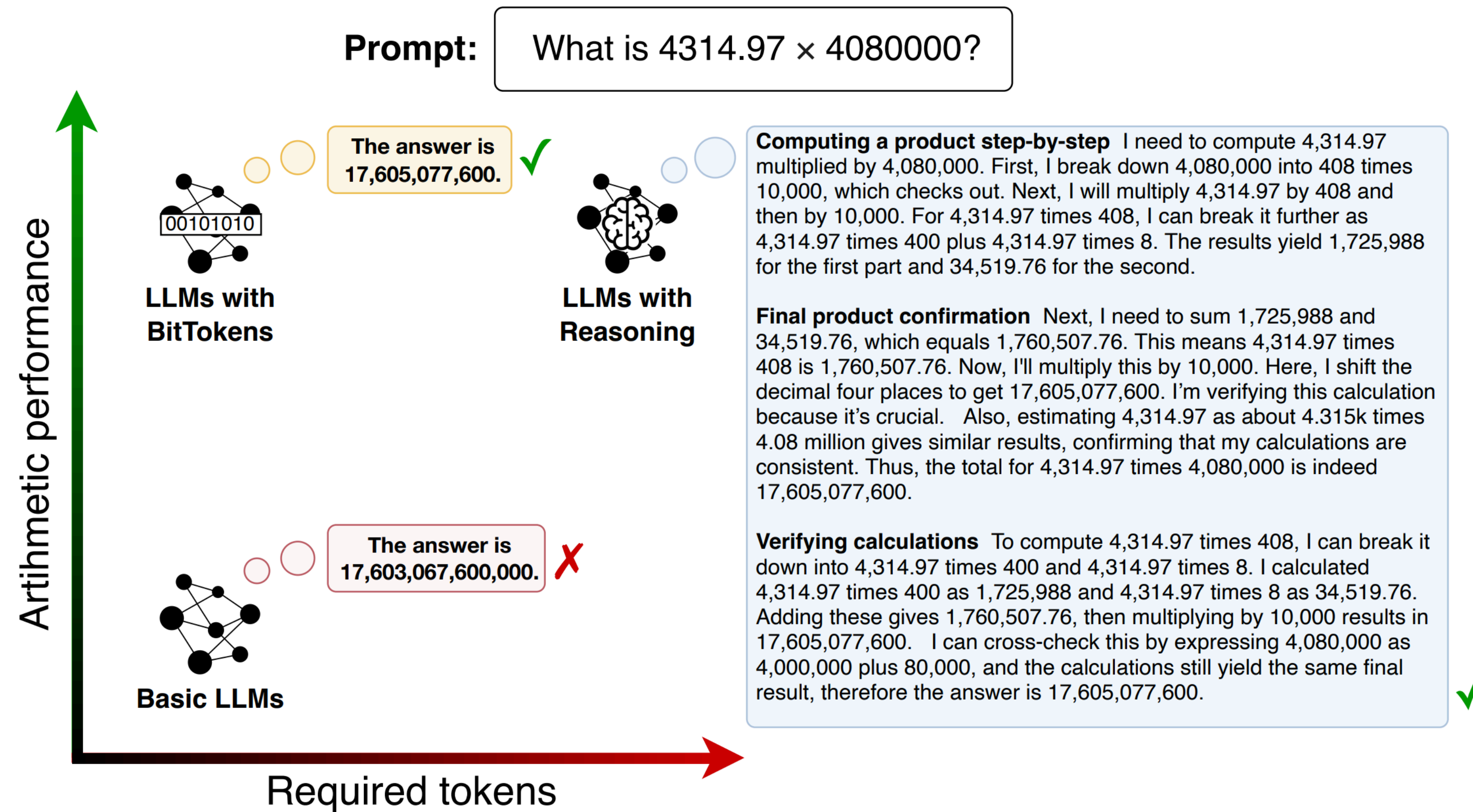
FoNE

does it have the desiderata?

- ✓ - **Token efficiency:** Every number is represented by a single token.
- ✓ - **Uniqueness:** Each value has exactly one valid encoding, with a unique inverse mapping.
- ✓ - **Structured:** The encoding geometry reflects numeric order and distance.
- ✓ - **Scale invariance:** A wide range of magnitudes can be represented with high precision.
- ✓ - **Normalization:** Encodings are bounded and information preserving under standard normalization functions used in language models (e.g., LayerNorm, RMSNorm).
- ✓ - **Numerical stability:** Representations remain accurate when using low-precision data type formats (e.g., fp8).
- ✓ - **Continuity:** Encodings vary smoothly with the underlying value, making them compatible with gradient-based optimization.
- ✓ - **Robustness:** Values can be decoded reliably even under stochastic prediction noise.
- ✗ - **Arithmetic:** Encodings admit learnable algorithms for core mathematical operations.

BitTokens

motivation



xVal and FoNE were cool ideas, but they didn't actually improve arithmetic performance across common operations.

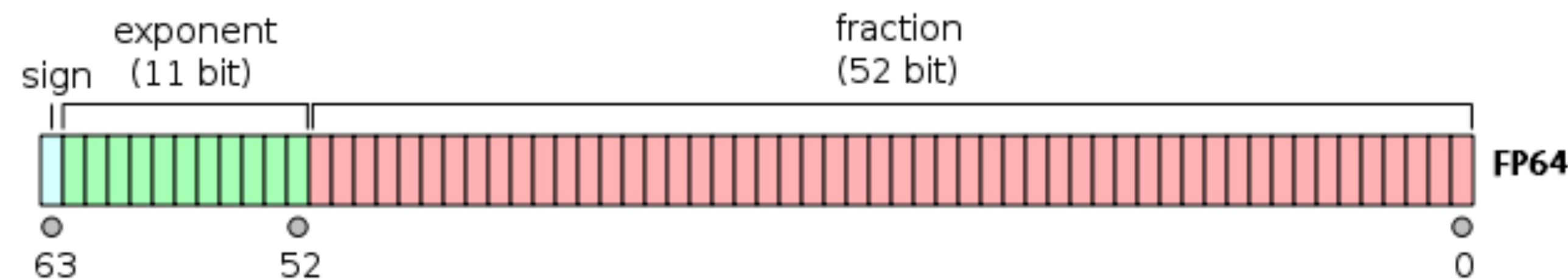
BitTokens

how do computers encode numbers?

- In the IEEE float64 standard, a signed real value v is encoded as:

$$v = (-1)^s \left(1 + \sum_{i=1}^{52} b_{52-i} 2^{-i} \right) \times 2^{E-1023}$$

with $s \in \{0,1\}$ denoting the sign bit, $E \in \{0,\dots,247\}$ for the exponent bit offset, and $b_j \in \{0,1\}$ for each of the 52 significant bits.

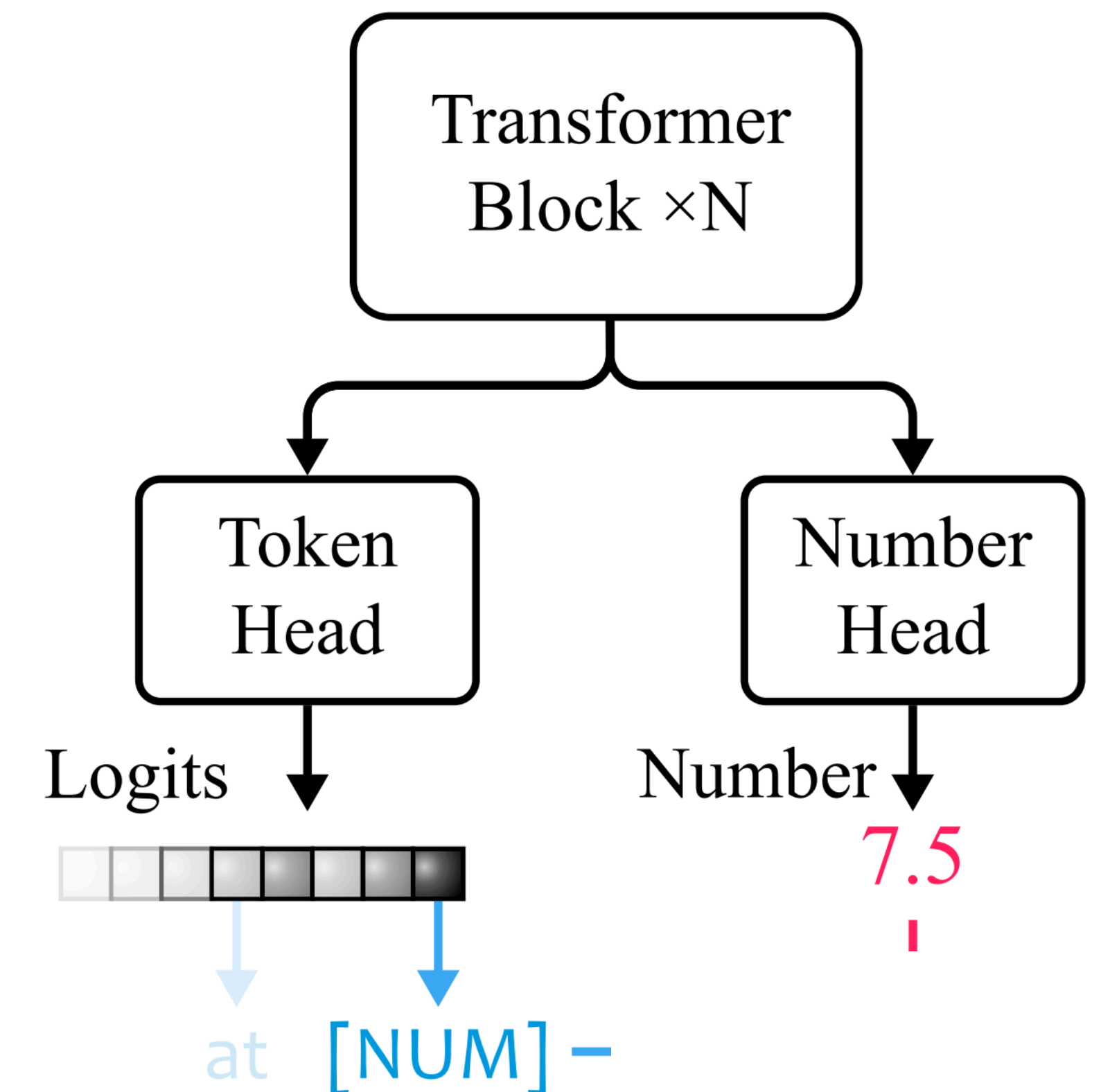


- Binary encoding like this are nice for arithmetic.
 - We've had efficient algorithms for binary addition and multiplication since the 1700s.

BitTokens

method

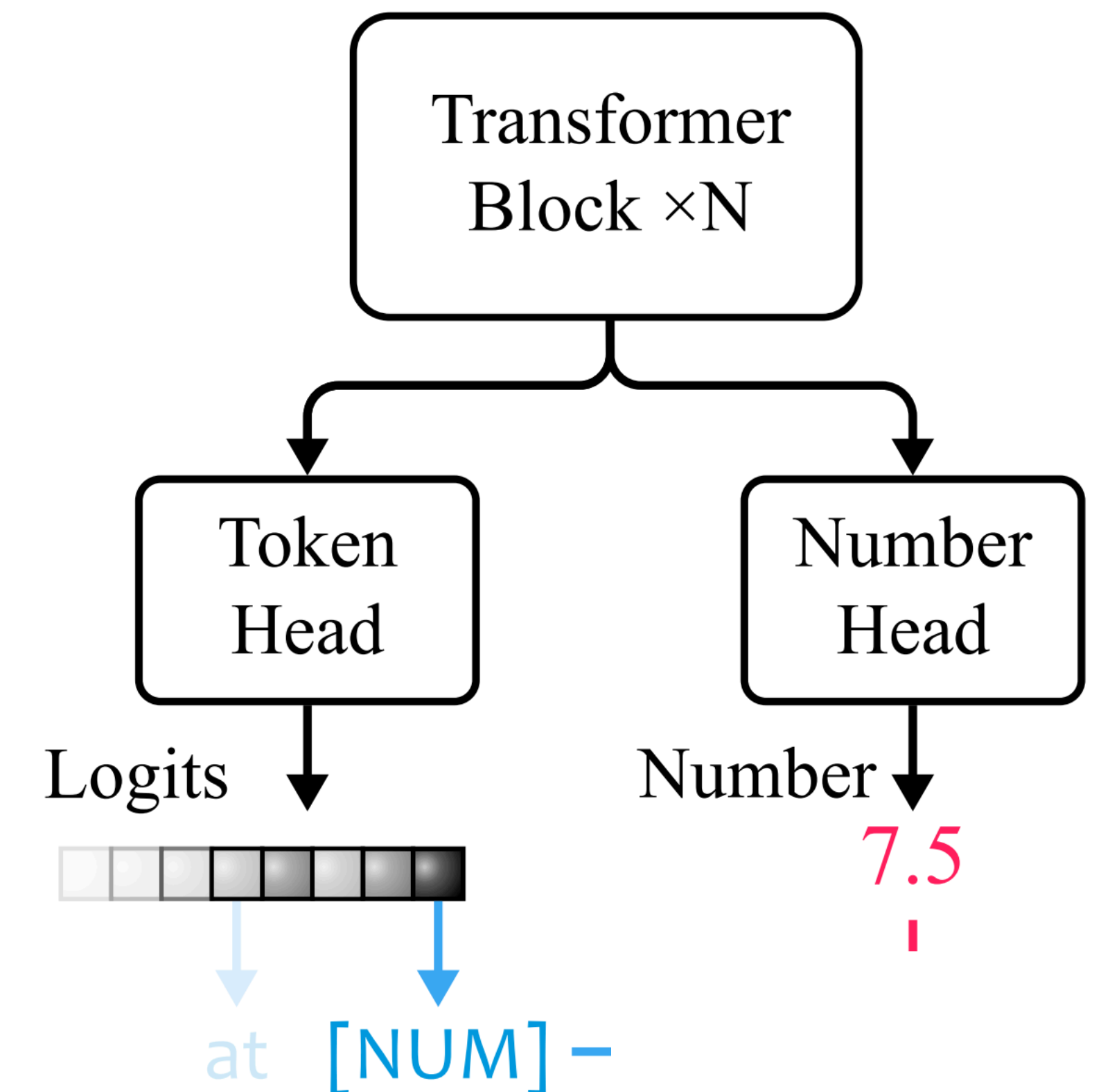
- Concatenate the binary representation of a number with the binary representation of its reciprocal value.
- Scale the bit vector to $[-1, 1]$ range so that encodings stay bounded and information-preserving under standard Transformer layer normalization (RMSNorm)



BitTokens

method

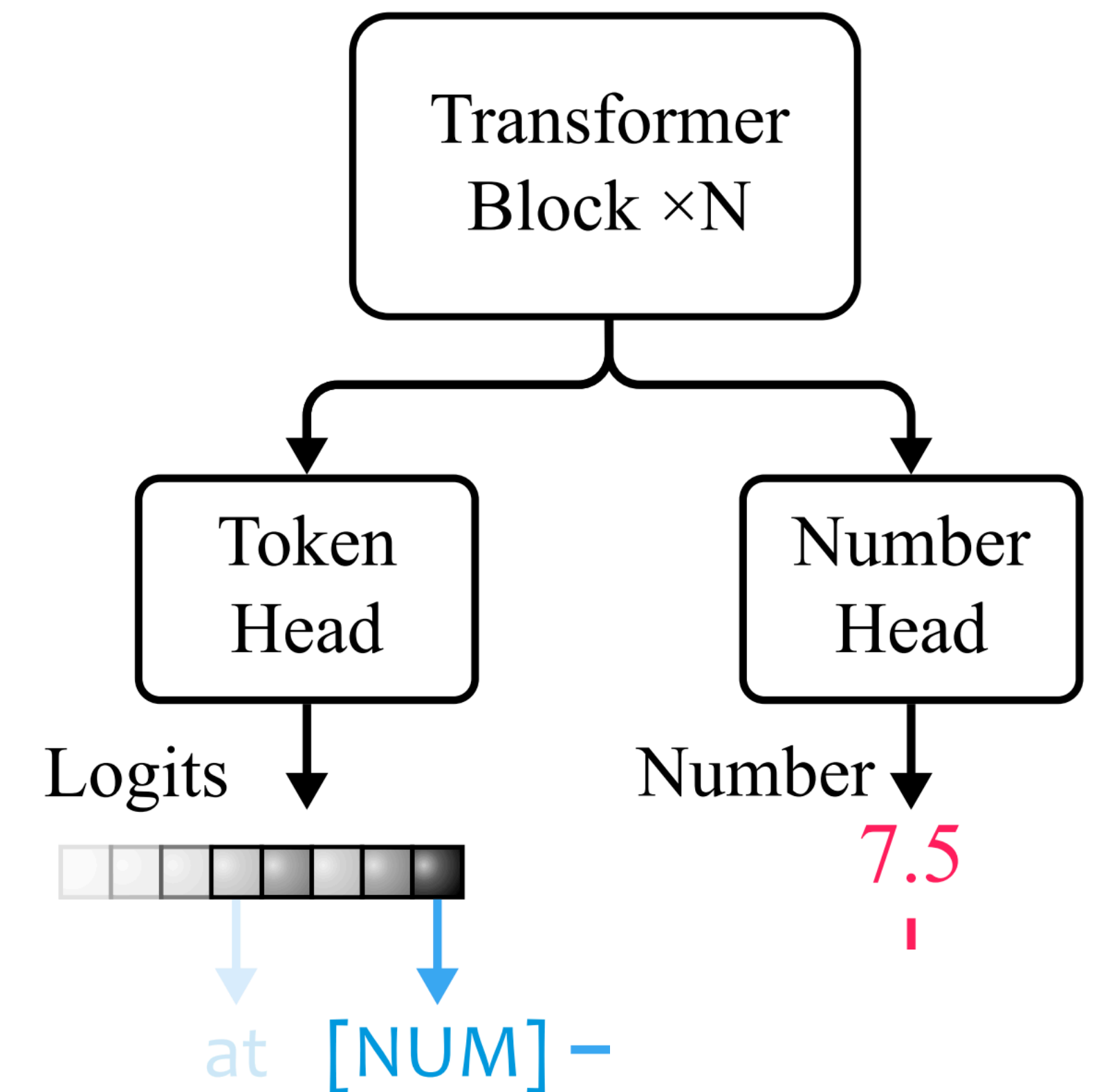
- Concatenate the binary representation of a number with the binary representation of its reciprocal value.
- Scale the bit vector to $[-1, 1]$ range so that encodings stay bounded and information-preserving under standard Transformer layer normalization (RMSNorm)
- Training loss
 - Why would we *not* want to use an L2 loss?



BitTokens

method

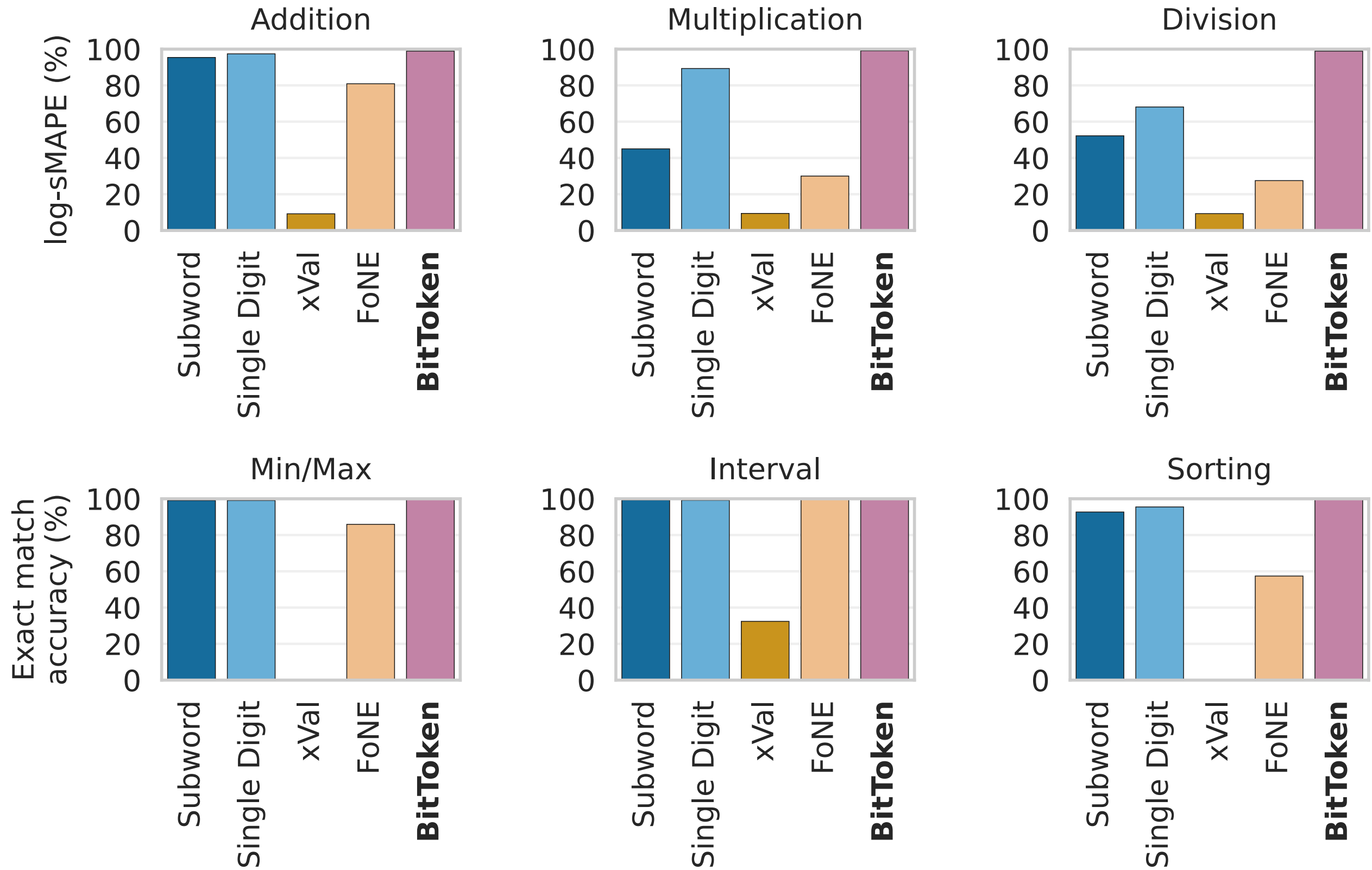
- Concatenate the binary representation of a number with the binary representation of its reciprocal value.
- Scale the bit vector to $[-1, 1]$ range so that encodings stay bounded and information-preserving under standard Transformer layer normalization (RMSNorm)
- Training loss
 - Why would we *not* want to use a regression loss?
 - Requires decoding the embeddings back into numbers at train-time
 - A error in a single bit could result in a *huge* distance between the numbers
 - Instead, they use a bit-wise binary cross entropy loss



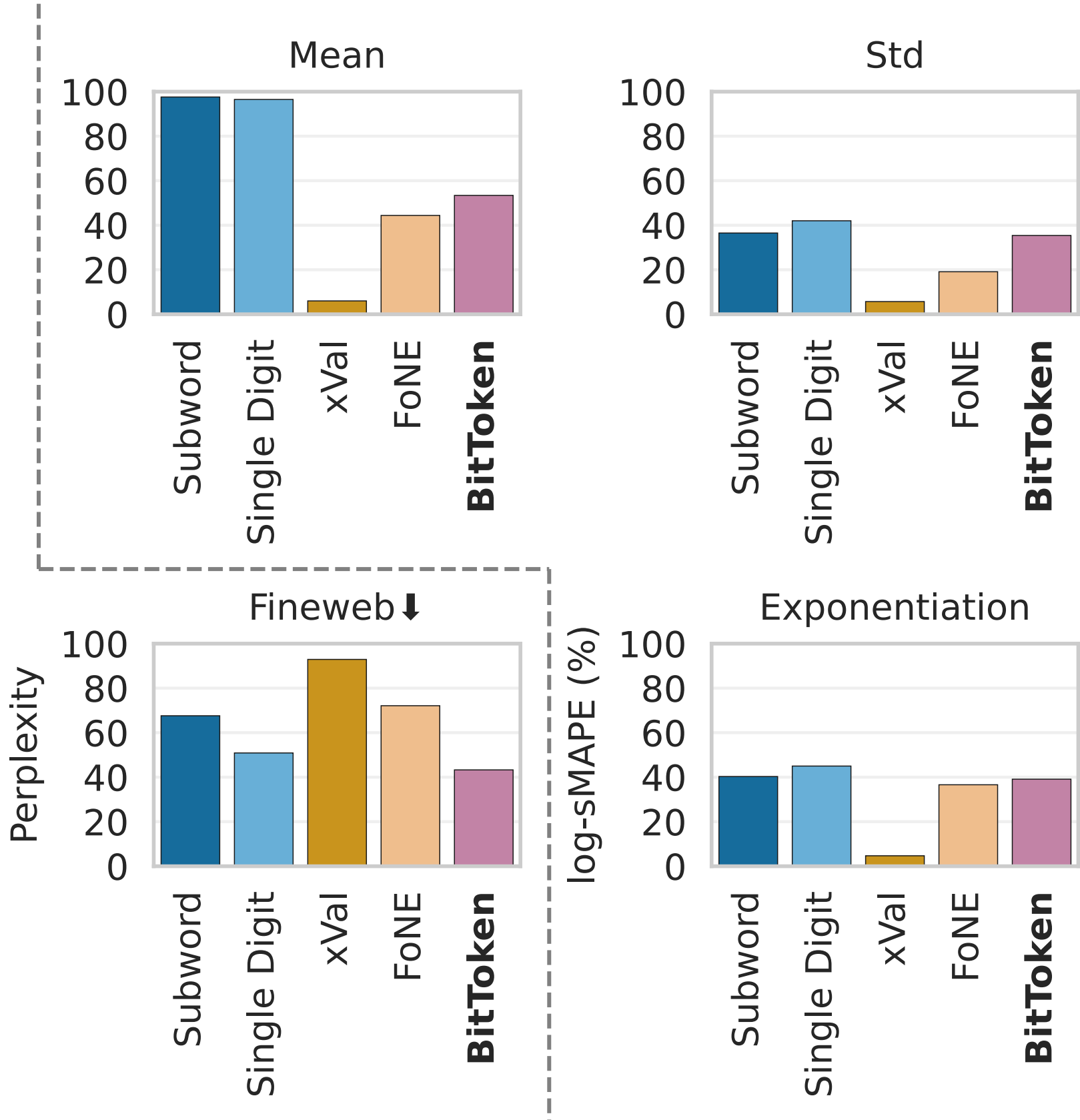
Comparing methods

Trained nanoGPT-2 on 9 arithmetic tasks and 1 language modeling task, with automatic task rebalancing to prioritize seeing examples for task model is currently performing poorly at.

Multi-task



Solo-task



Automatic detection of generated text

Two classes of methods

- Watermarking methods
 - OpenAI or Google secretly encode a secret into the text their models generate.
 - Those with knowledge of the key can check if generated text has the secret in it or not.
- Detection in the wild
 - Text may be generated by any model using any decoding strategy.
 - It may be in any domain.
 - It may be edited by humans.

Watermarking

general idea

- separate all vocabulary items into a “red” list and a “green” list
- over-sample tokens from one list or another according to a secret pattern

Watermarking

algorithm

Algorithm 1 Text Generation with Hard Red List

Input: prompt, $s^{(-N_p)} \dots s^{(-1)}$

for $t = 0, 1, \dots$ **do**

1. Apply the language model to prior tokens $s^{(-N_p)} \dots s^{(t-1)}$ to get a probability vector $p^{(t)}$ over the vocabulary.
2. Compute a hash of token $s^{(t-1)}$, and use it to seed a random number generator.
3. Using this seed, randomly partition the vocabulary into a “green list” G and a “red list” R of equal size.
4. Sample $s^{(t)}$ from G , never generating any token in the red list.

end for

Watermarking

algorithm

Algorithm 1 Text Generation with Hard Red List

Input: prompt, $s^{(-N_p)} \dots s^{(-1)}$

for $t = 0, 1, \dots$ **do**

1. Apply the language model to prior tokens $s^{(-N_p)} \dots s^{(t-1)}$ to get a probability vector $p^{(t)}$ over the vocabulary.
2. Compute a hash of token $s^{(t-1)}$, and use it to seed a random number generator.
3. Using this seed, randomly partition the vocabulary into a “green list” G and a “red list” R of equal size.
4. Sample $s^{(t)}$ from G , never generating any token in the red list.

end for

Algorithm 2 Text Generation with Soft Red List

Input: prompt, $s^{(-N_p)} \dots s^{(-1)}$
green list size, $\gamma \in (0, 1)$
hardness parameter, $\delta > 0$

for $t = 0, 1, \dots$ **do**

1. Apply the language model to prior tokens $s^{(-N_p)} \dots s^{(t-1)}$ to get a logit vector $l^{(t)}$ over the vocabulary.
2. Compute a hash of token $s^{(t-1)}$, and use it to seed a random number generator.
3. Using this random number generator, randomly partition the vocabulary into a “green list” G of size $\gamma|V|$, and a “red list” R of size $(1 - \gamma)|V|$.
4. Add δ to each green list logit. Apply the softmax operator to these modified logits to get a probability distribution over the vocabulary.

$$\hat{p}_k^{(t)} = \begin{cases} \frac{\exp(l_k^{(t)} + \delta)}{\sum_{i \in R} \exp(l_i^{(t)}) + \sum_{i \in G} \exp(l_i^{(t)} + \delta)}, & k \in G \\ \frac{\exp(l_k^{(t)})}{\sum_{i \in R} \exp(l_i^{(t)}) + \sum_{i \in G} \exp(l_i^{(t)} + \delta)}, & k \in R. \end{cases}$$

5. Sample the next token, $s^{(t)}$, using the watermarked distribution $\hat{p}^{(t)}$.

end for

Detection in the wild

- While there are clever, training-free approaches SoTA mostly involves finetuning a Transformer on large amounts of challenging and highly varied data.
- Tricks from Pangram:
 - hard negative mining: prioritize training on examples the model is misclassifying
 - synthetic mirrors: for each real human training example, generate AI text that matches the topic and length of the human example