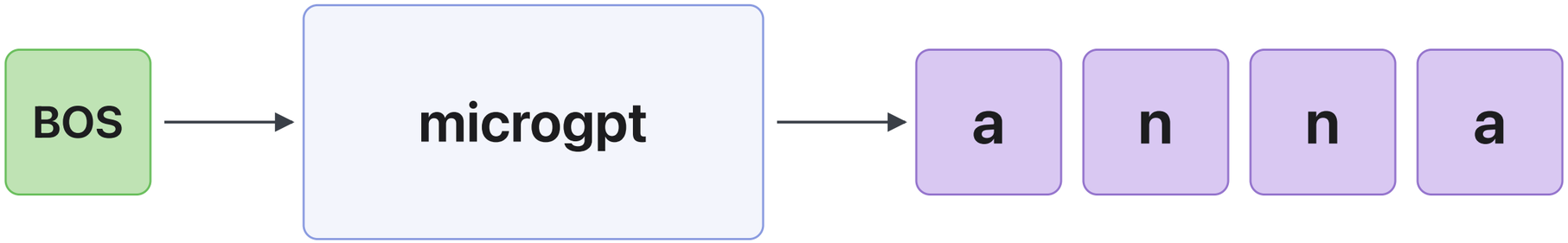
microgpt

A GPT in 200 lines of Python

Computational Physics II · PHY4008 · Sogang University · November 19, 2026

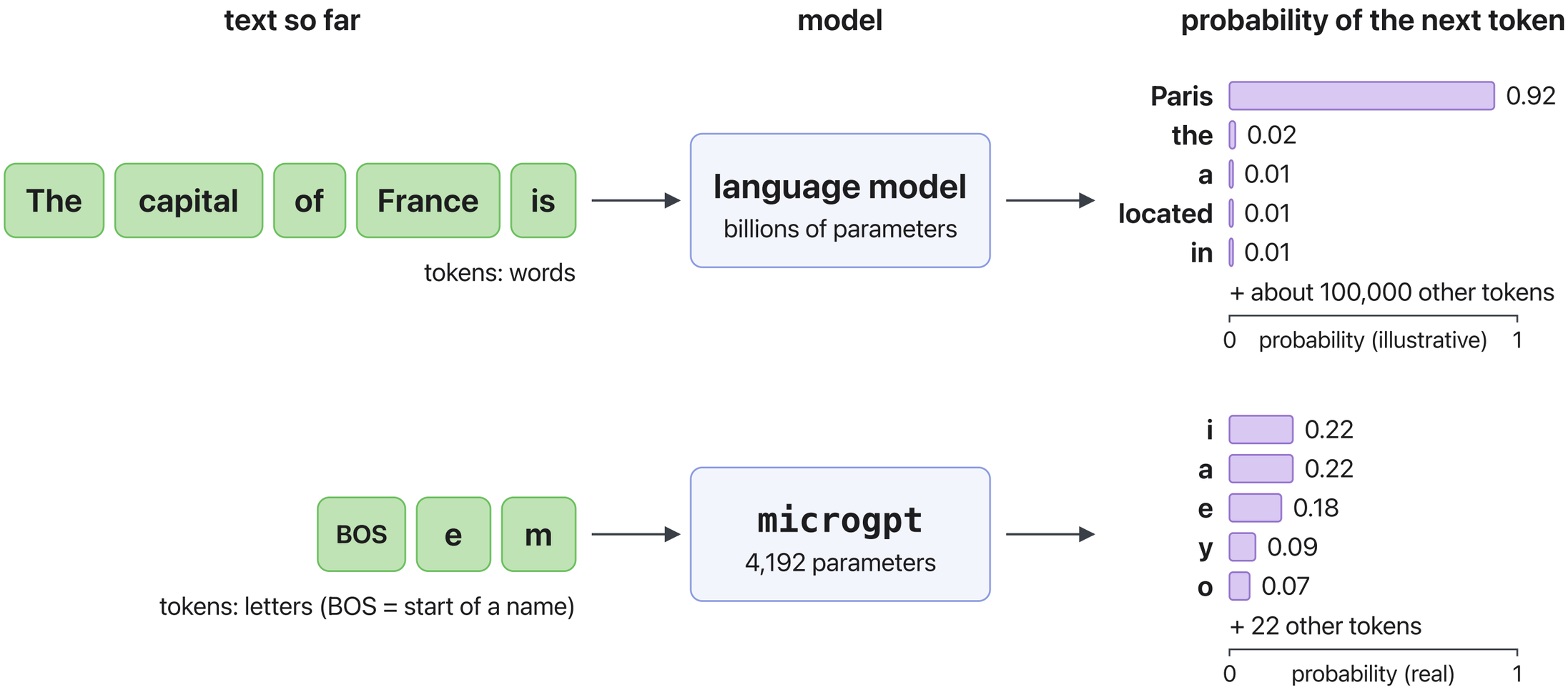


Last week, this week

	Ising classifier	microgpt
Input	one spin configuration	the characters so far
Output	2 class probabilities	27 next-character probabilities
Loss	cross-entropy	cross-entropy
Gradients	backpropagation by hand	automatic (autograd)
Network	one hidden layer	one transformer layer

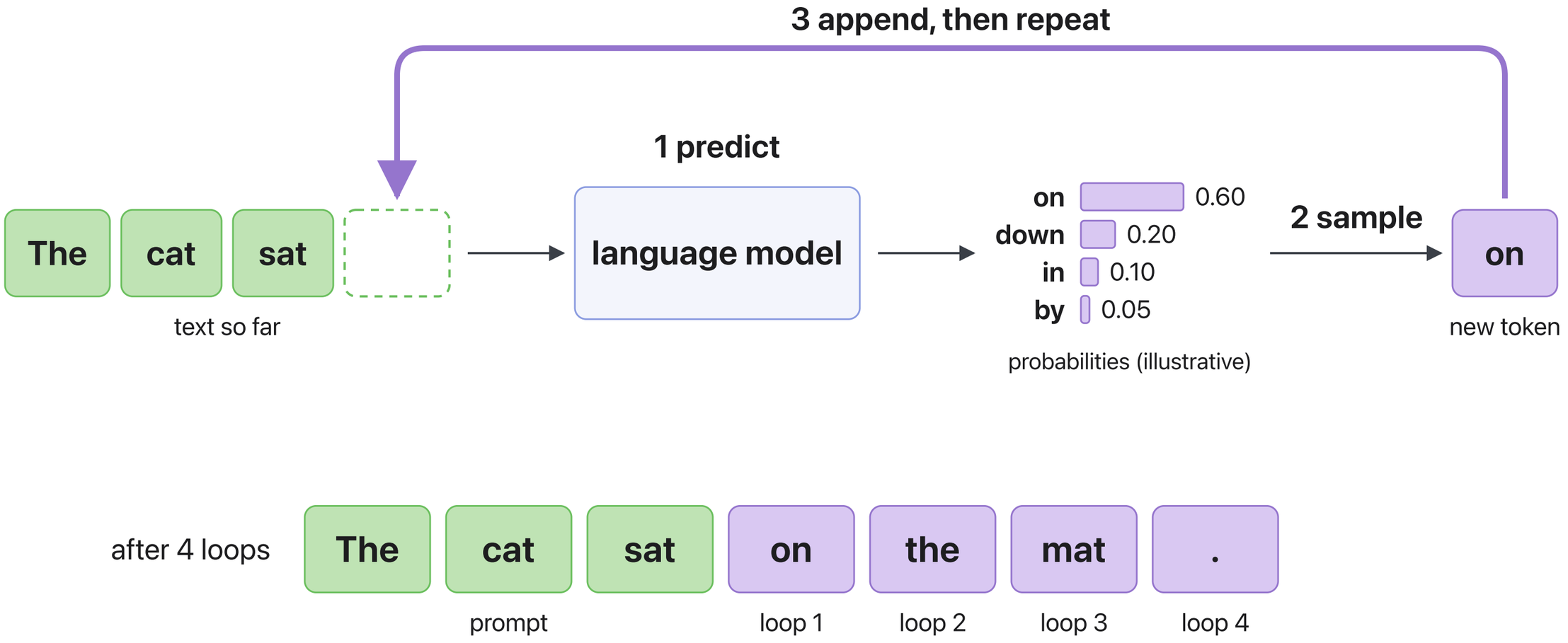
The ingredients are the same as last week: a network, a cross-entropy loss, gradients. New today: the task of predicting the next character, and attention.

Language model



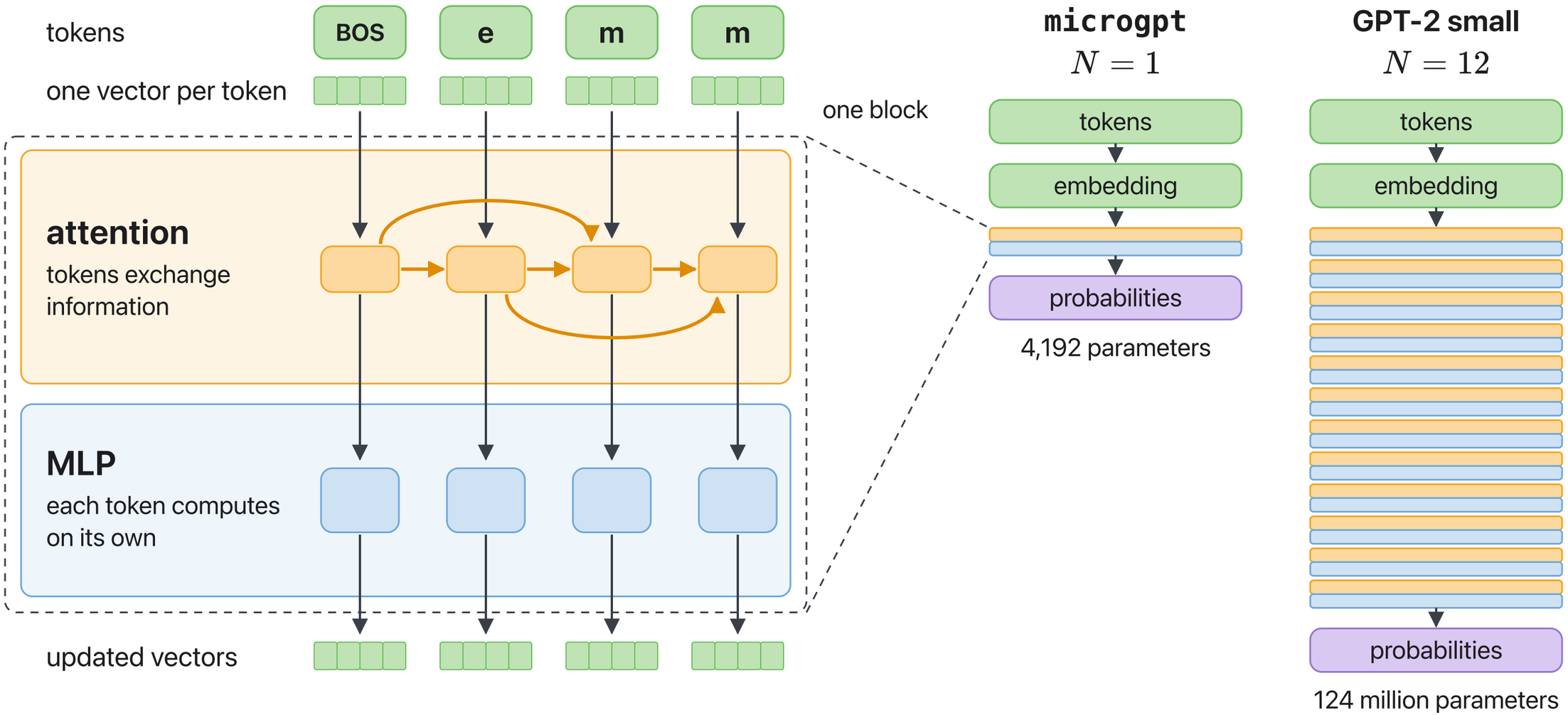
A language model maps the text so far to a probability for every possible next token. A large model does this for words, microgpt for letters.

Generating text



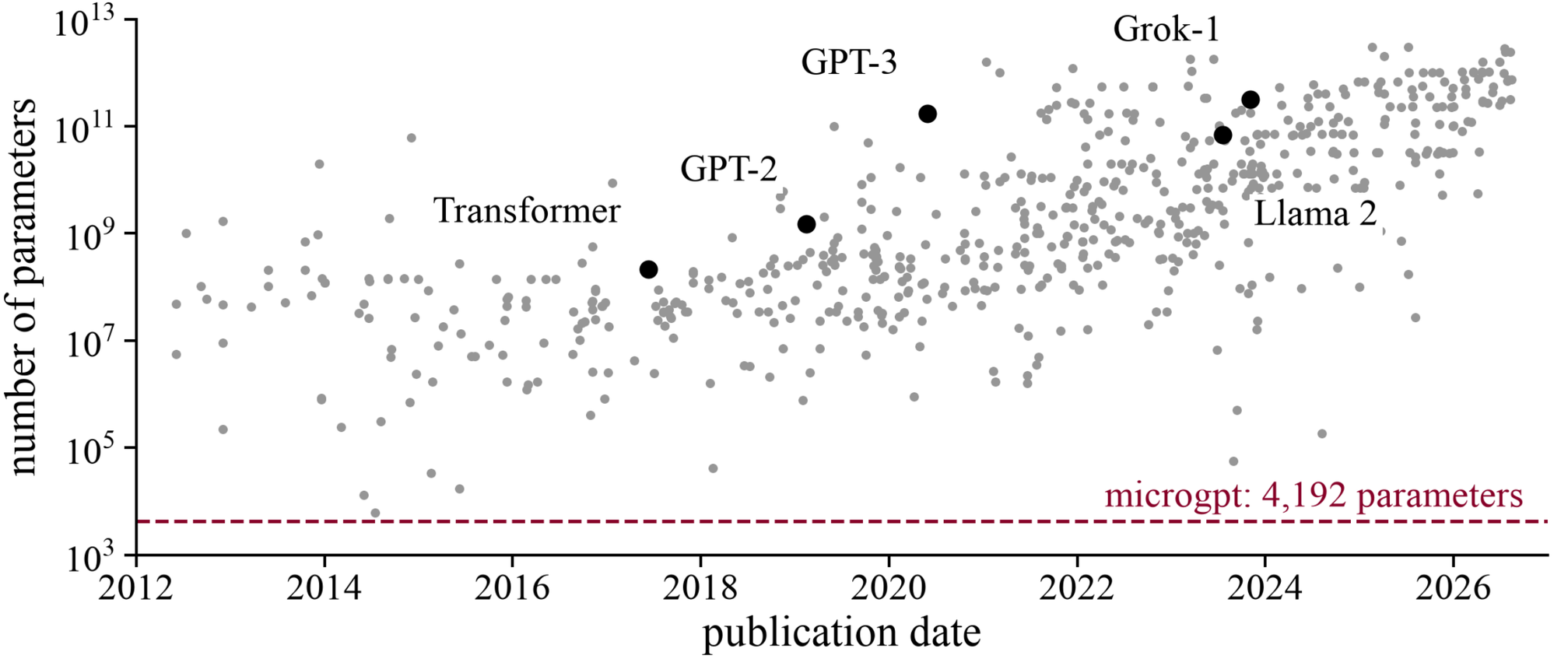
Text is generated one token at a time: predict, sample, append, repeat. ChatGPT runs this same loop.

Transformer



A transformer repeats one block: attention lets tokens exchange information, the MLP processes each token separately. GPT is a stack of such blocks.

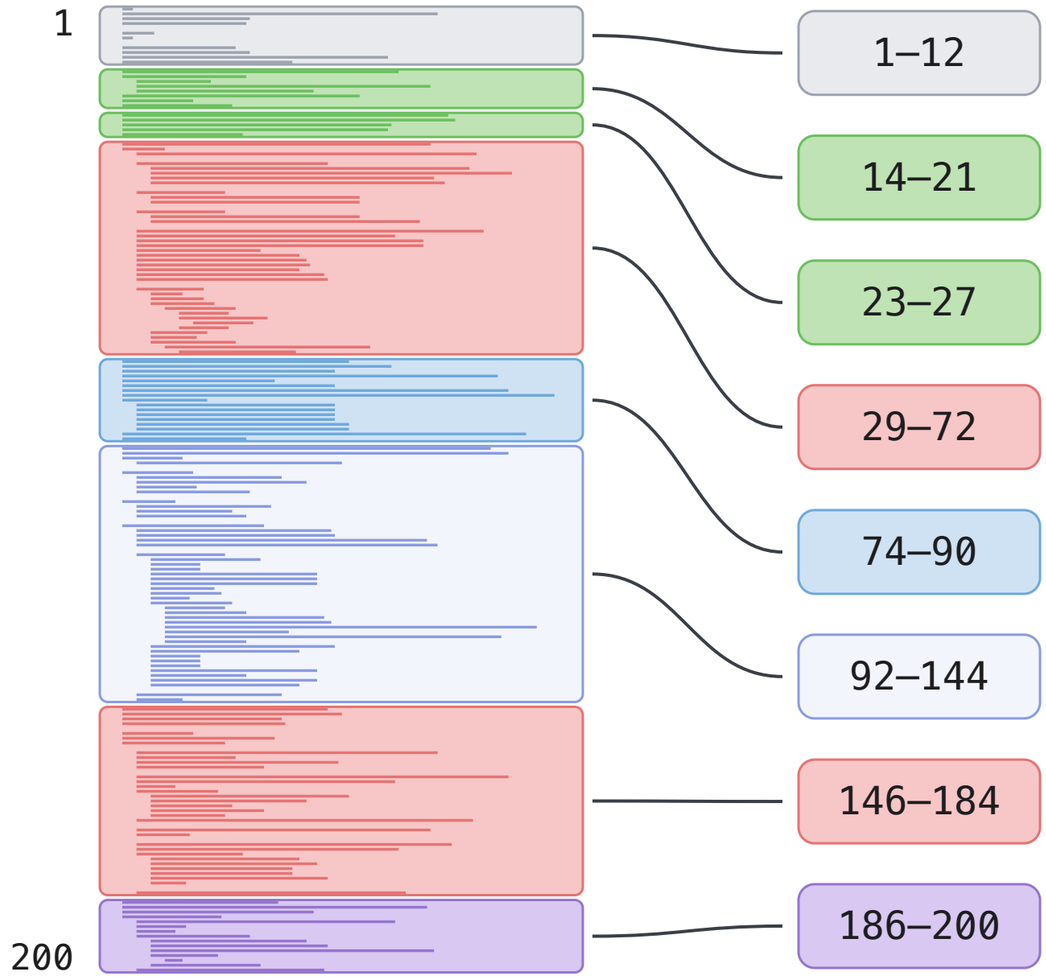
Scale



Data: Epoch AI (2026), via Our World in Data, CC BY

Models have grown by many orders of magnitude, with many improvements along the way. The core idea is unchanged: a transformer that predicts the next token. microgpt is that core alone.

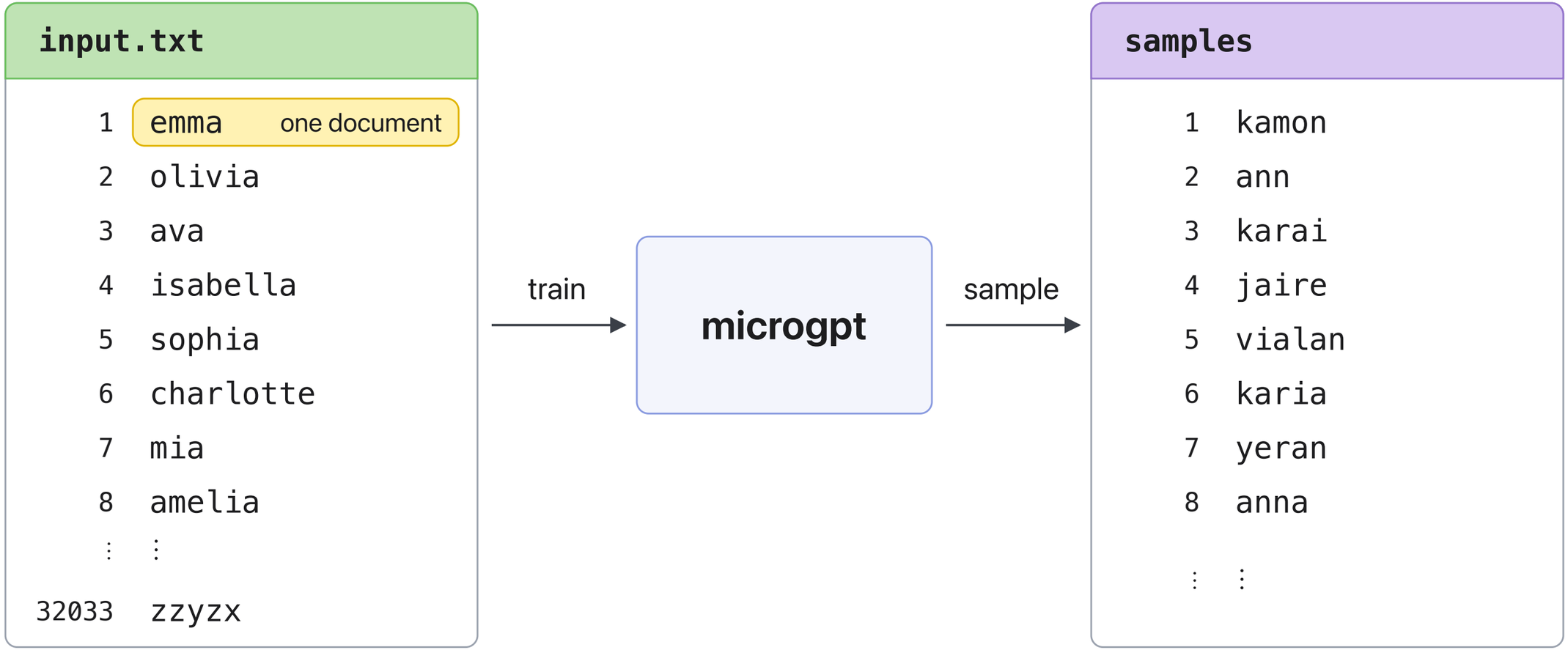
microgpt.py



1–12	**imports**	`os, math, random`
14–21	**Dataset**	32,033 names
23–27	**Tokenizer**	characters ↔ integers
29–72	**Autograd**	chain rule on a graph
74–90	**Parameters**	4,192 numbers
92–144	**Architecture**	token → 27 scores
146–184	**Training loop**	1,000 Adam steps
186–200	**Inference**	sample new names

Seven short parts make a complete GPT.
“This file is the complete algorithm. Everything else is just efficiency.” (A. Karpathy)

Dataset

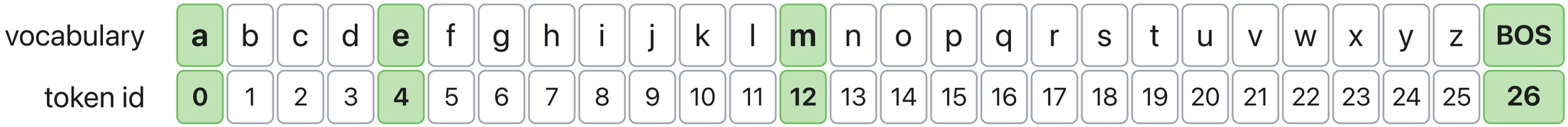


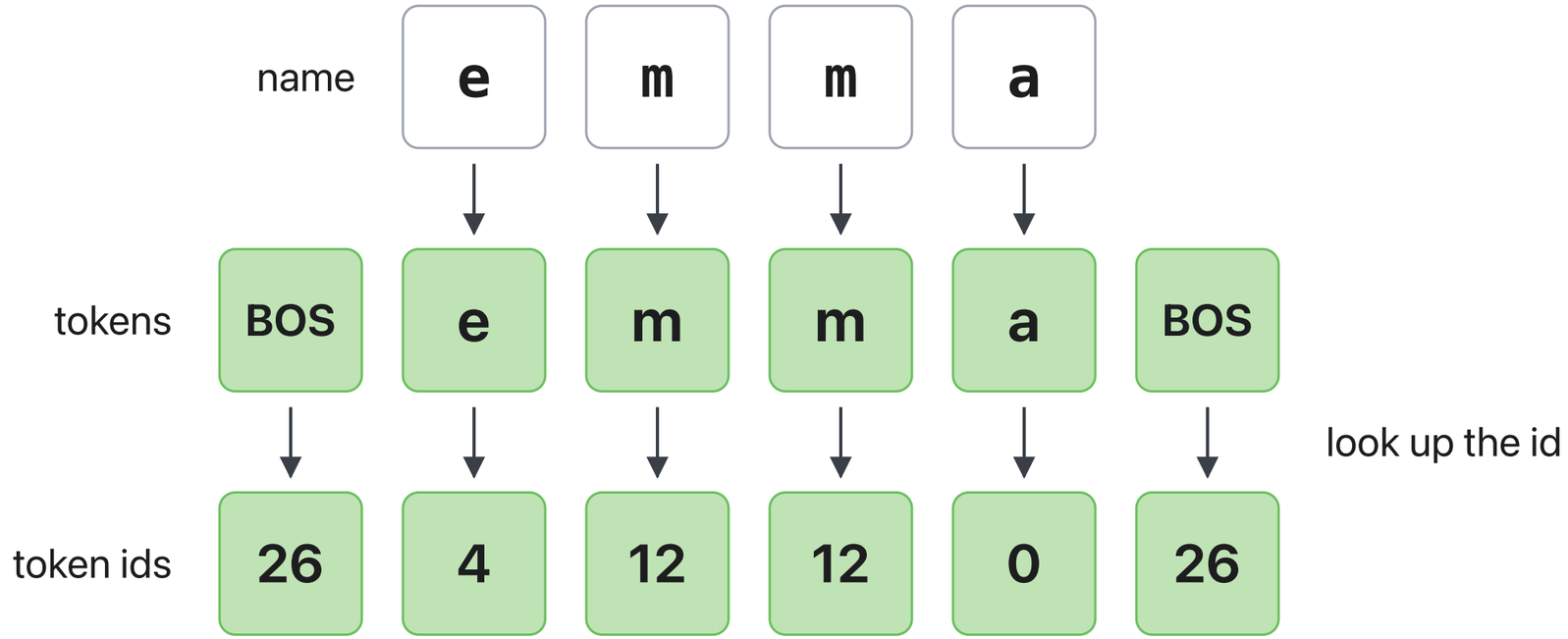
training data: 32,033 names

model output: new names

The model sees nothing but a list of names. It learns their statistics and produces new strings that look like names; some coincide with real ones.

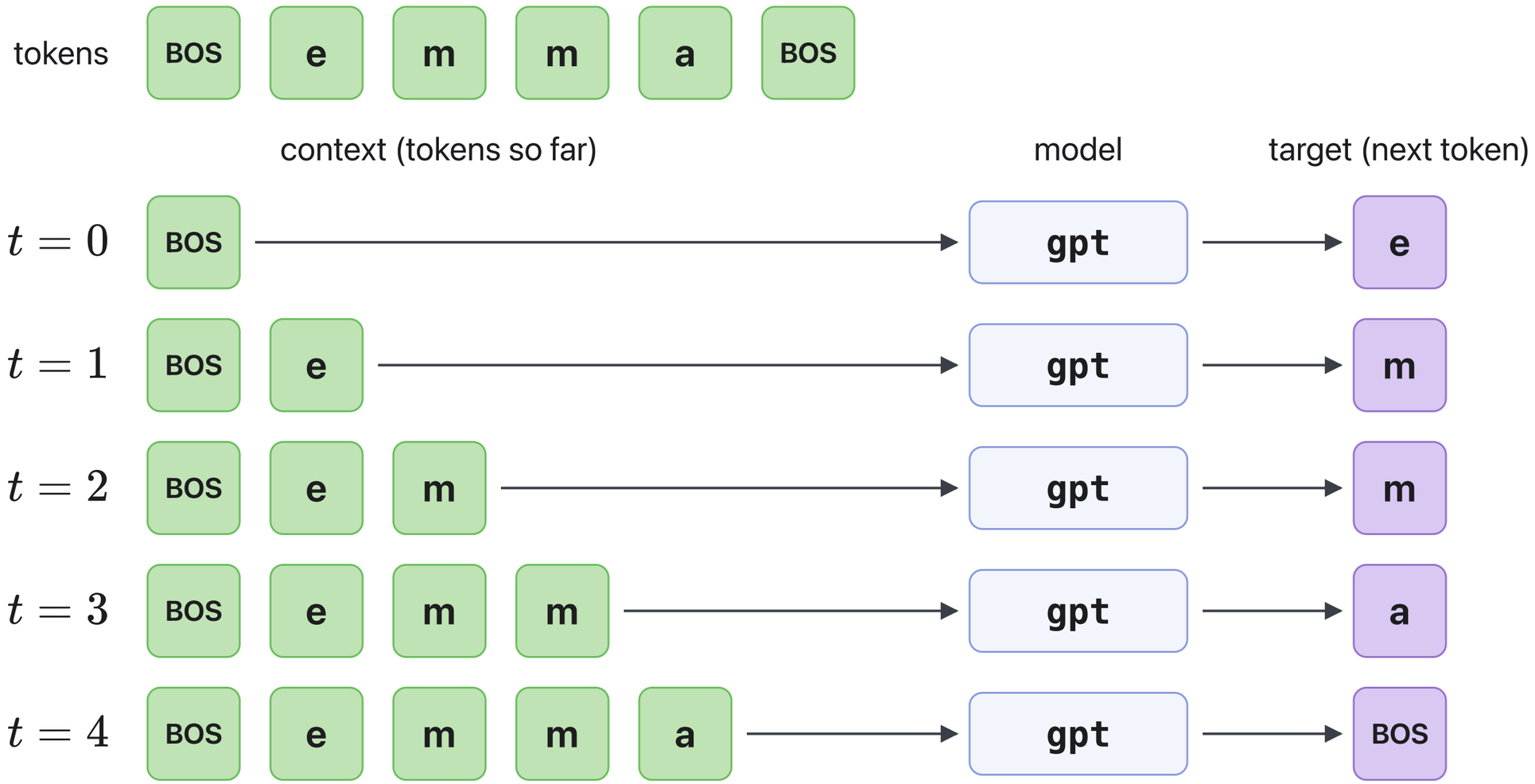
Tokenizer





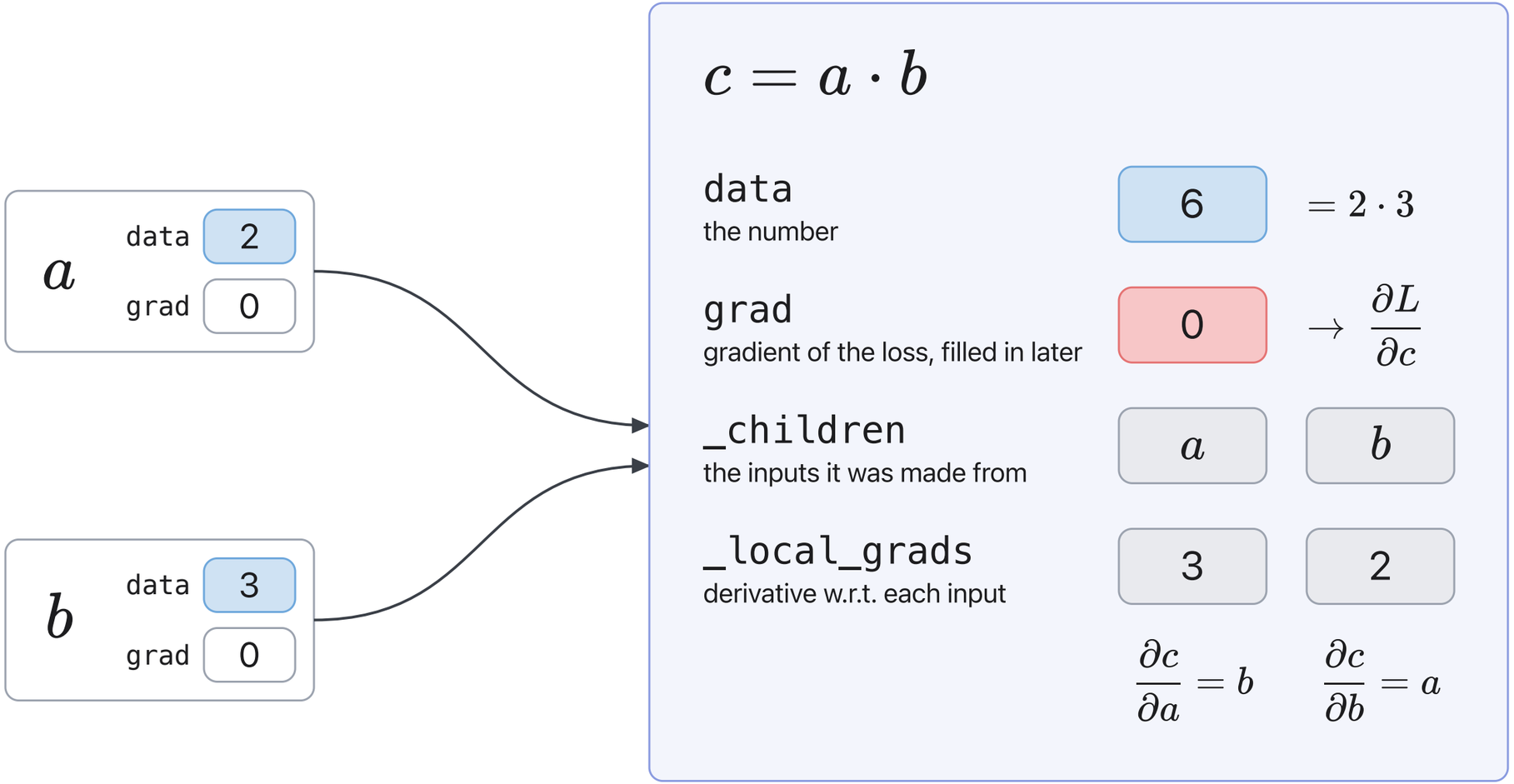
A tokenizer turns text into integers. Each letter is one token; the extra token BOS marks both ends of a name.

Next-token prediction



The model is trained to predict the next token from the tokens before it.
One four-letter name gives five training examples.

Value



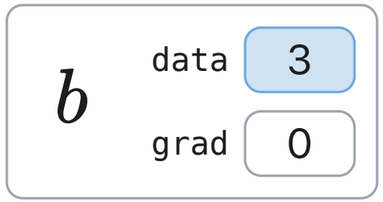
Every number in the model is a `Value`: it remembers its inputs and the local derivatives, so the chain rule can be applied later.

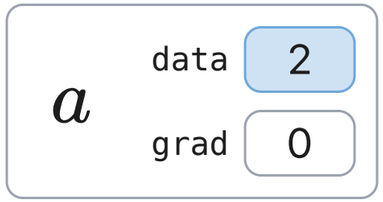
Lego blocks

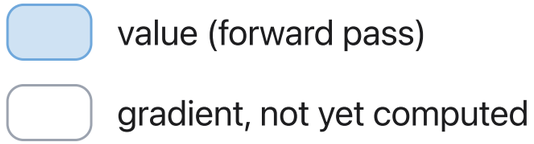
Operation	Forward	Local gradients $\partial/\partial a,\ \partial/\partial b$
`a + b`	$a + b$	$1, \quad 1$
`a * b`	ab	$b, \quad a$
`a ** n`	a^n	$n\,a^{n-1}$
`a.log()`	$\ln a$	$1/a$
`a.exp()`	e^a	e^a
`a.relu()`	$\max(0, a)$	$1 \text{ if } a > 0, \text{ else } 0$

The whole model is built from these six operations. Each one knows only its own local derivative; the chain rule does the rest.

Forward pass on a graph

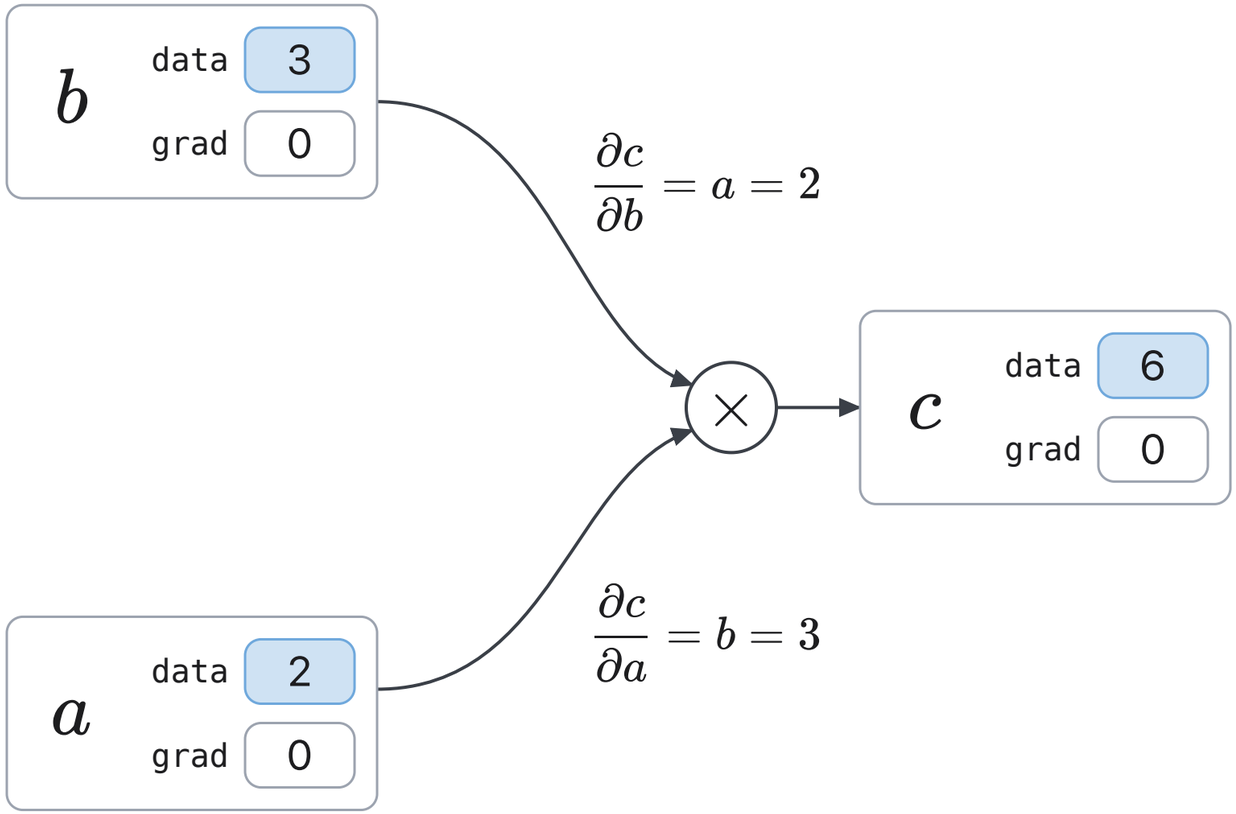






The inputs are leaf nodes: each holds a value, and a gradient that starts at 0.

Forward pass on a graph



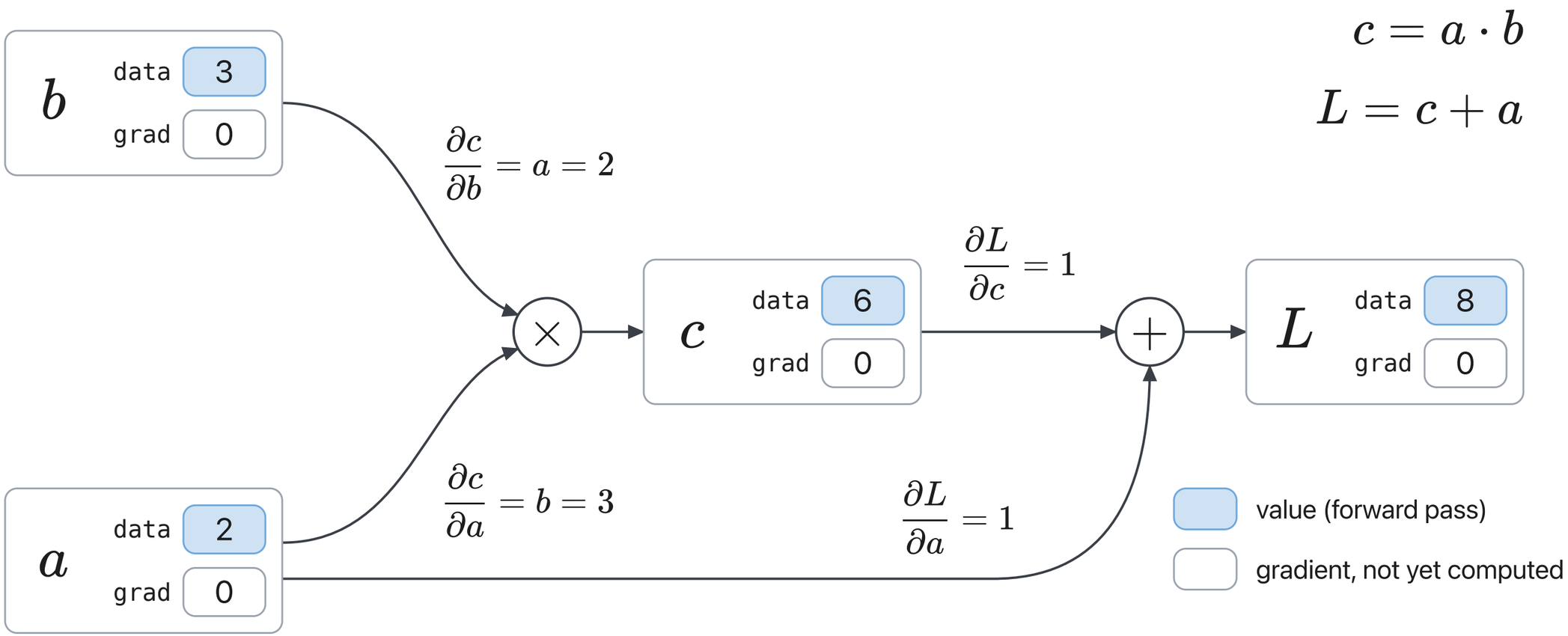
$$c = a \cdot b$$

value (forward pass)

gradient, not yet computed

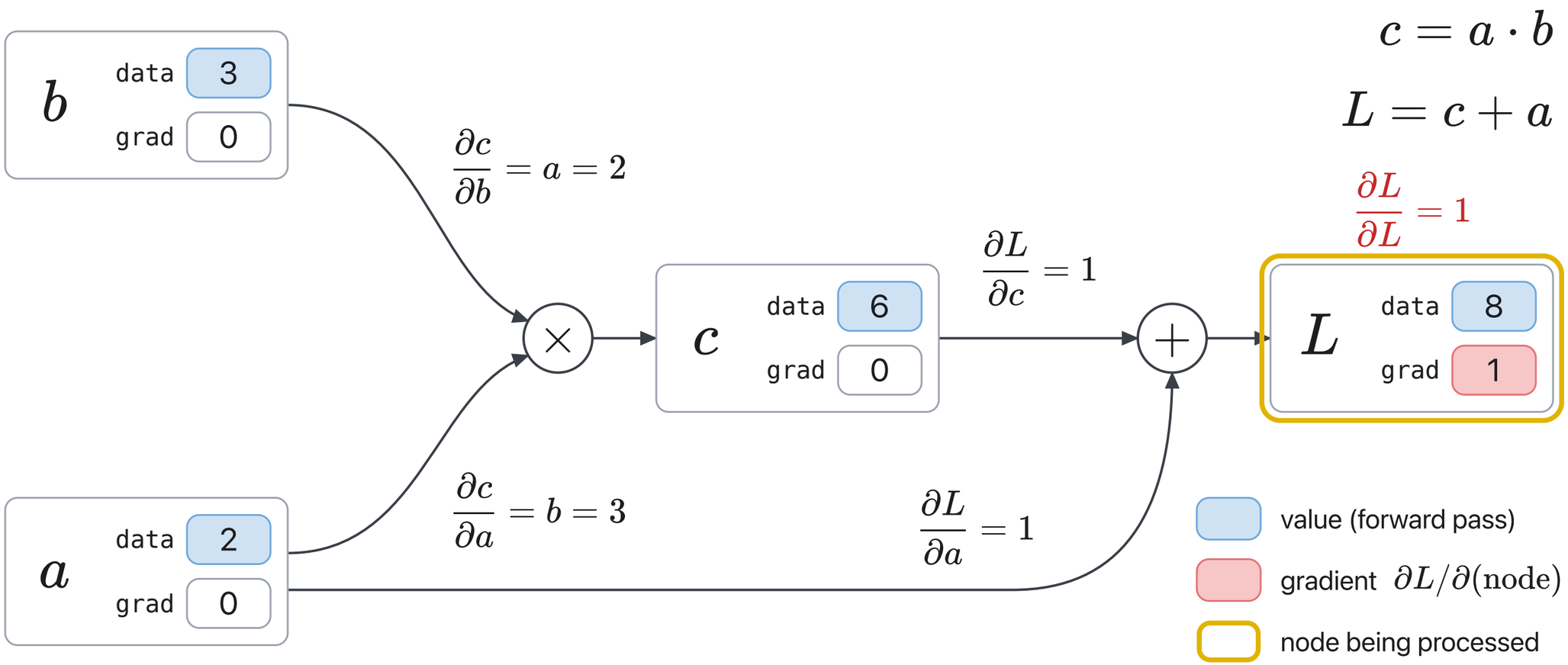
Each operation creates a new node and records the local derivative with respect to each input.

Forward pass on a graph



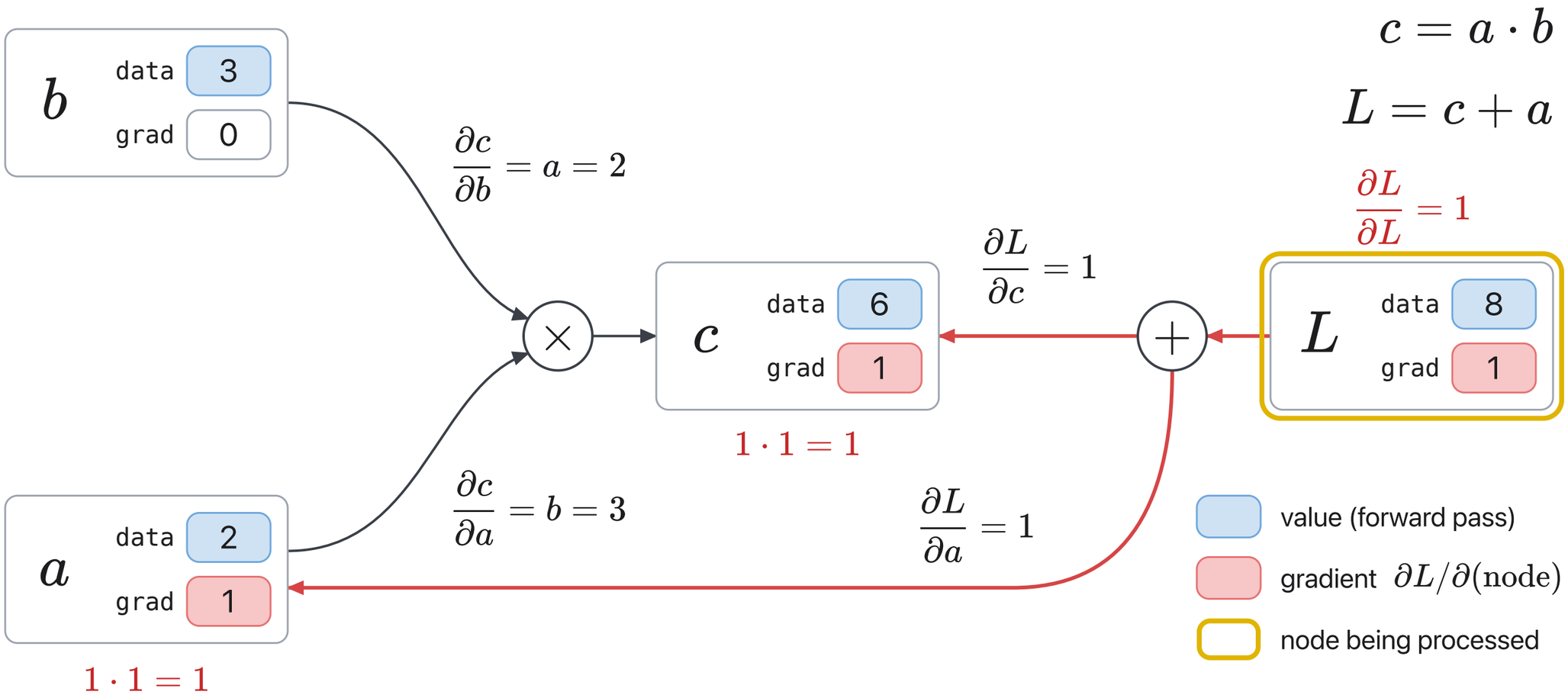
The forward pass ends at L. Every edge now carries a local derivative; no gradient has been computed yet.

Backward pass



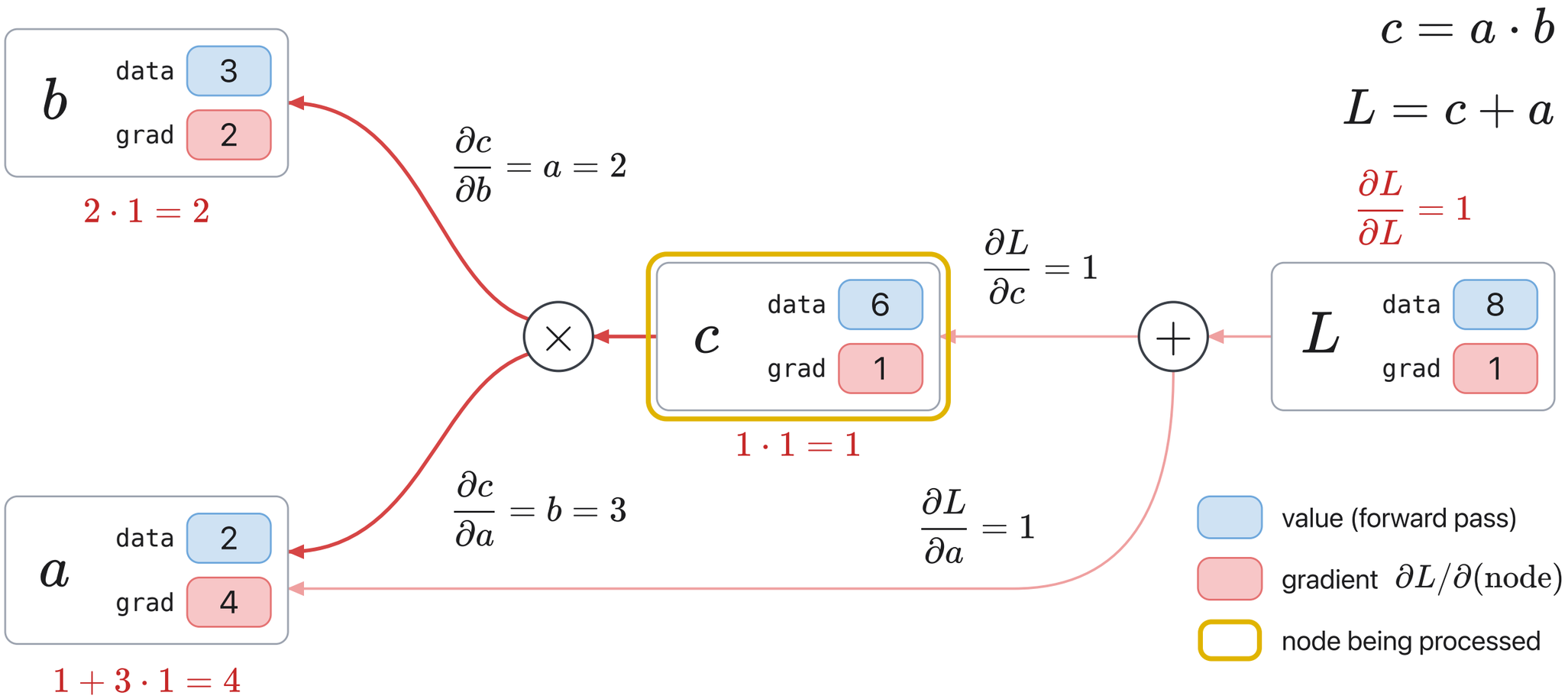
The backward pass starts at the output: the derivative of L with respect to itself is 1.

Backward pass



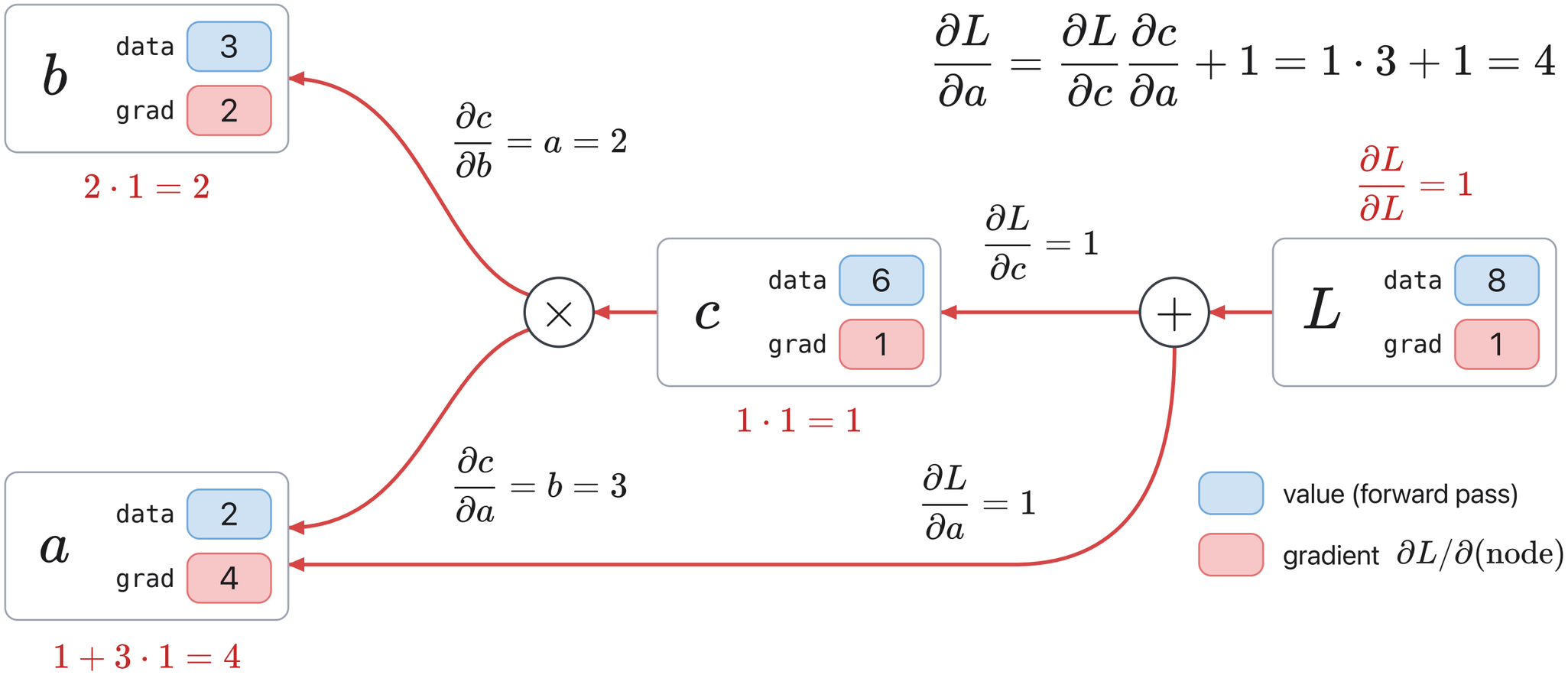
Each node passes its gradient to its inputs, multiplied by the local derivative on the edge.

Backward pass



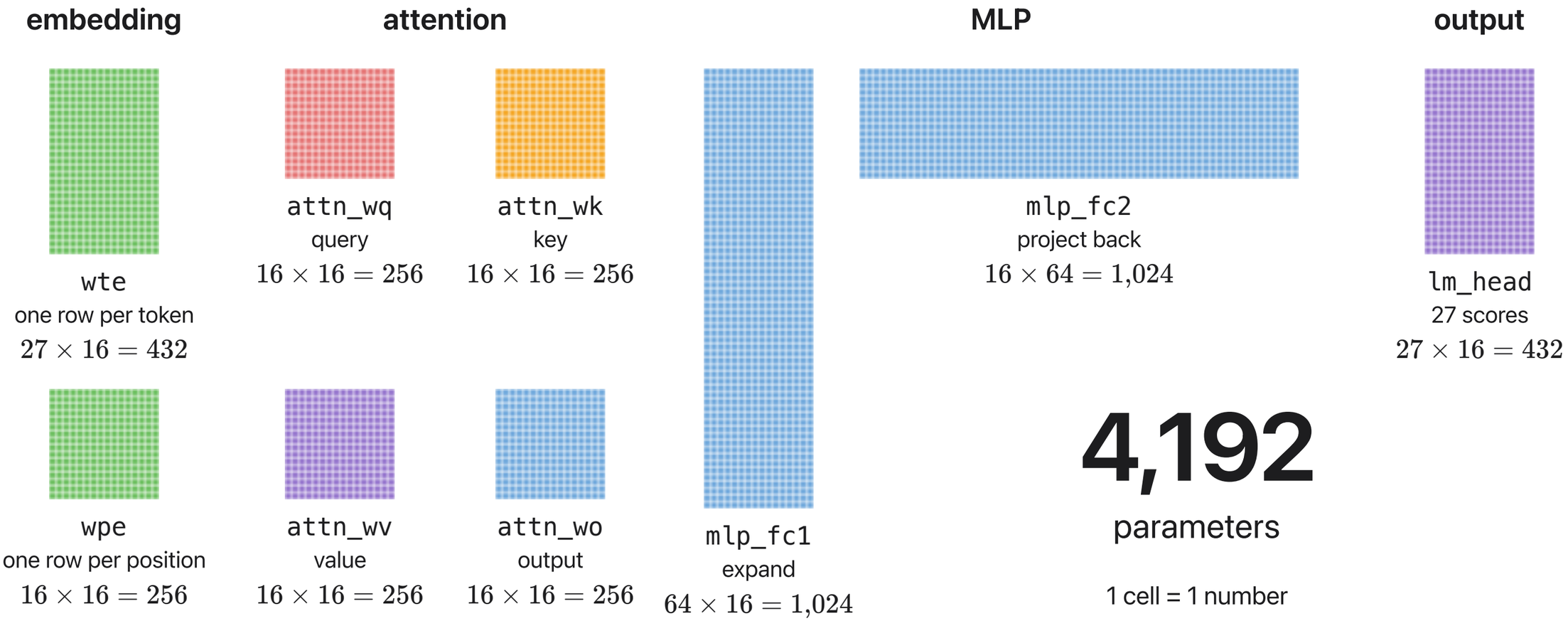
A value used twice receives a gradient from each path, and the contributions add up.

Backward pass



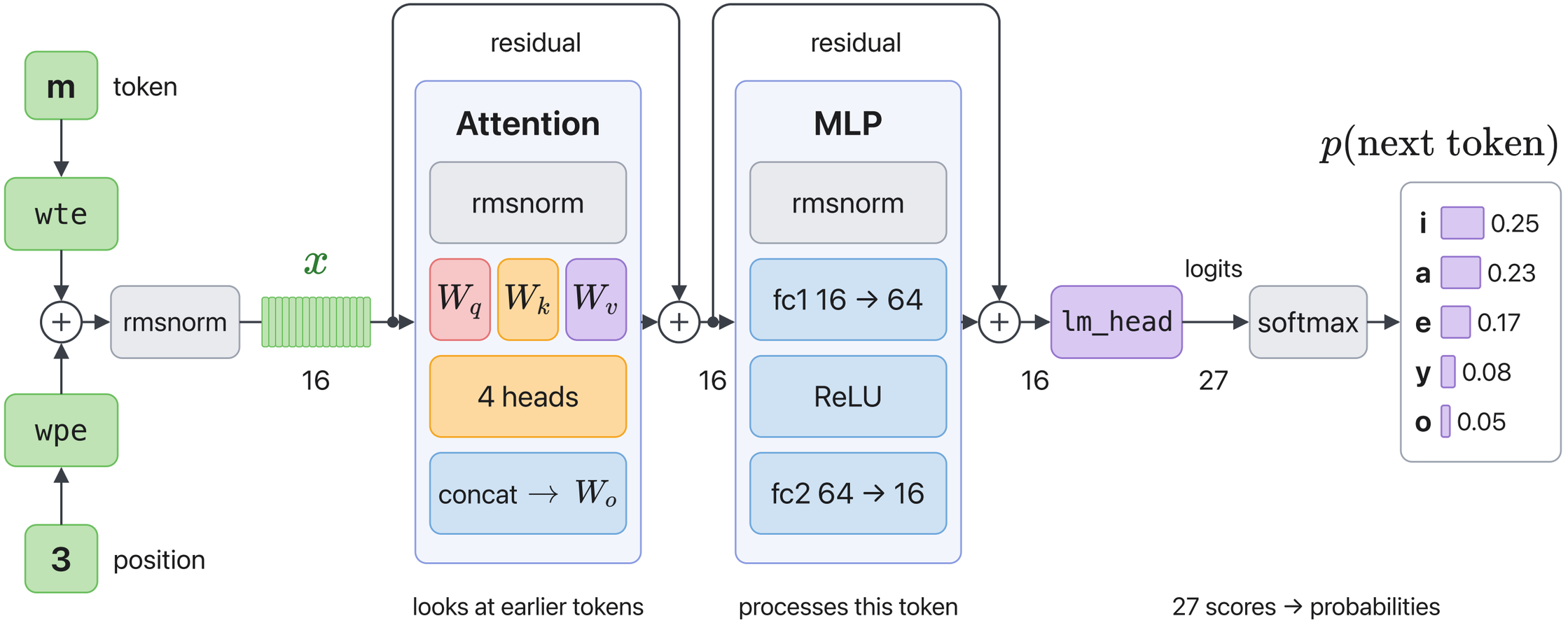
One backward sweep gives the derivative of L with respect to every node: the chain rule, applied edge by edge.

Parameters



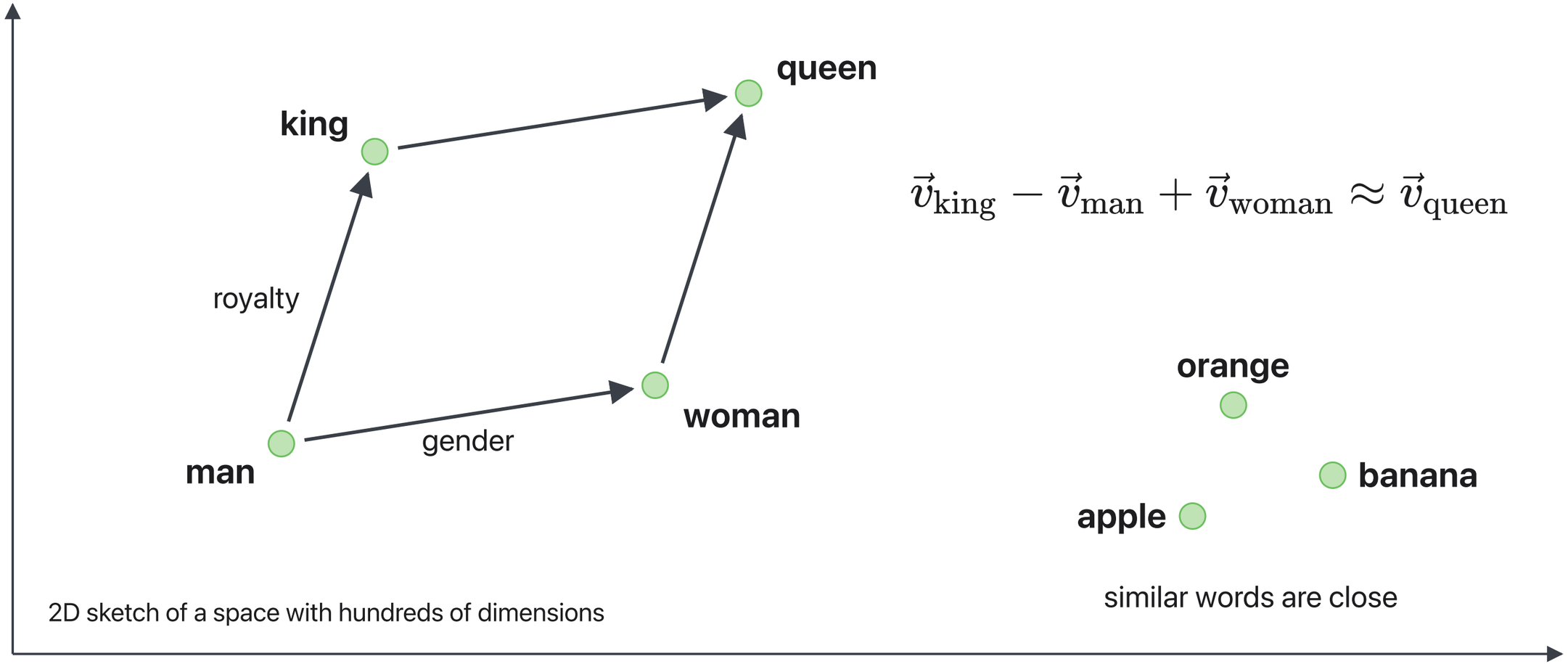
The model is nothing but these 4,192 numbers, initially random. Training adjusts them; the architecture only fixes how they are combined.

Forward pass



One token and its position become a vector of 16 numbers. Attention and the MLP each add a correction to it; the result is turned into 27 next-token probabilities.

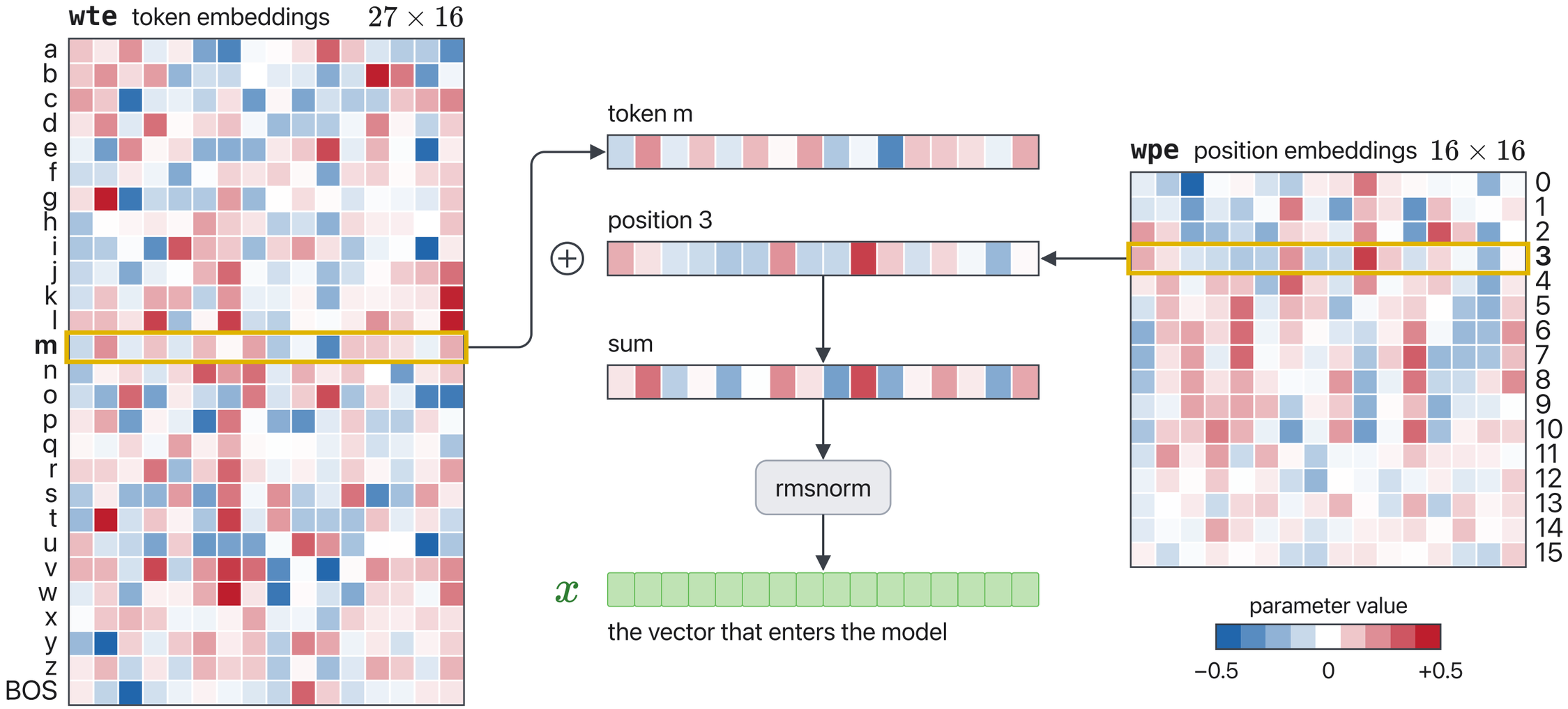
Vector space



Mikolov et al., 2013 (word2vec)

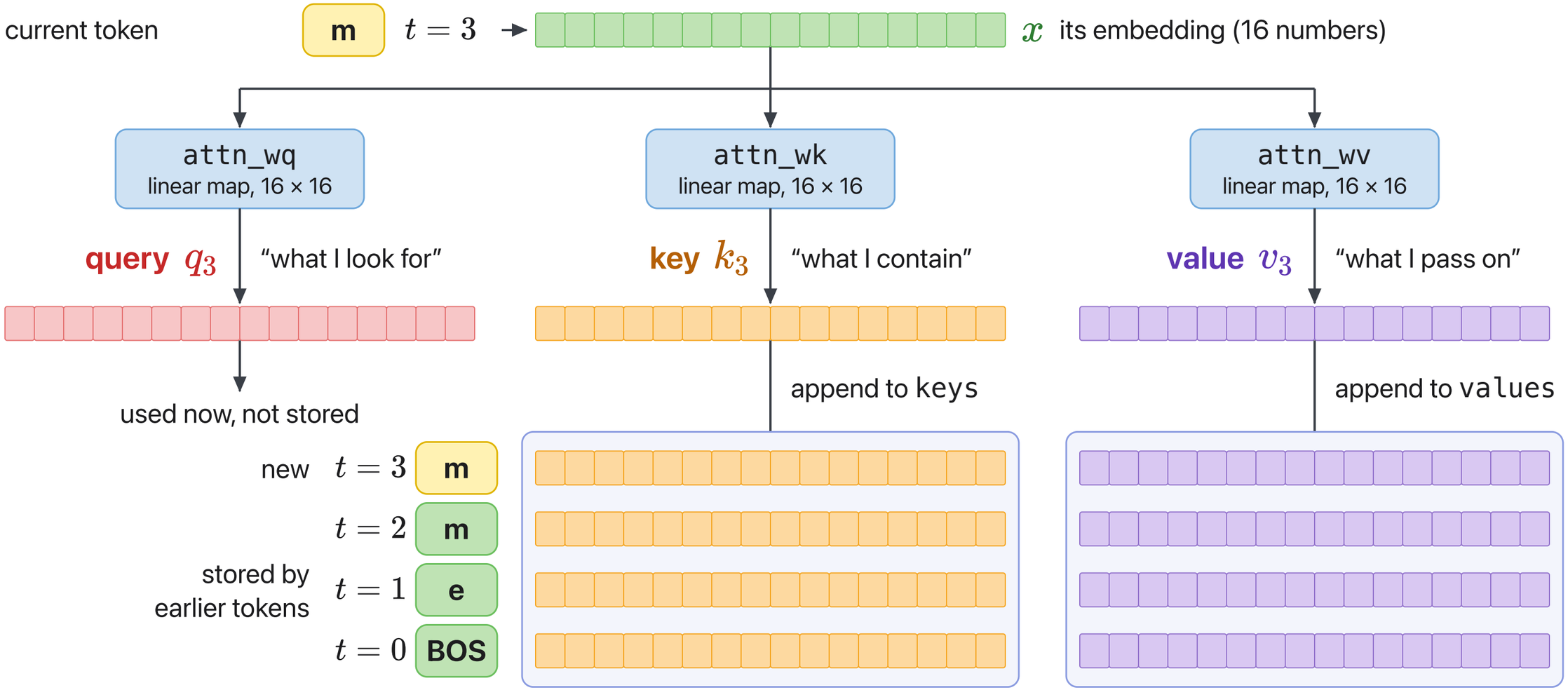
An embedding gives every token a position in a vector space. After training, nearby vectors mean similar things and directions carry meaning; microgpt does the same for letters, in 16 dimensions.

Embedding



Every token and every position owns one row of 16 learned numbers. Their sum, rescaled, is the vector x that represents “m at position 3”.

Query, key, value



Each token is turned into three vectors: a query (what it looks for), a key (what it offers) and a value (what it passes on). Keys and values are stored for later tokens.

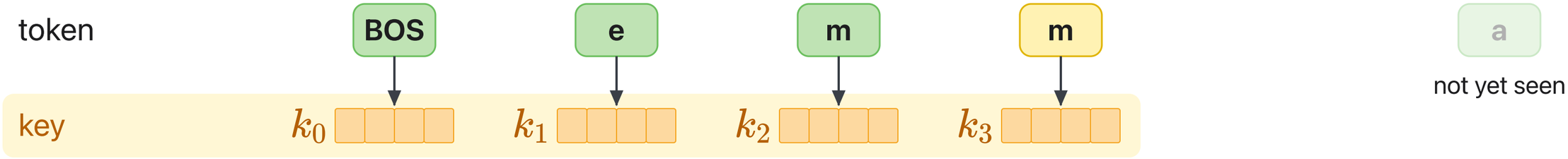
softmax

$$p_i = \frac{e^{l_i}}{\sum_j e^{l_j}}$$

l_i: logits, any real numbers · p_i: probabilities, positive and summing to 1

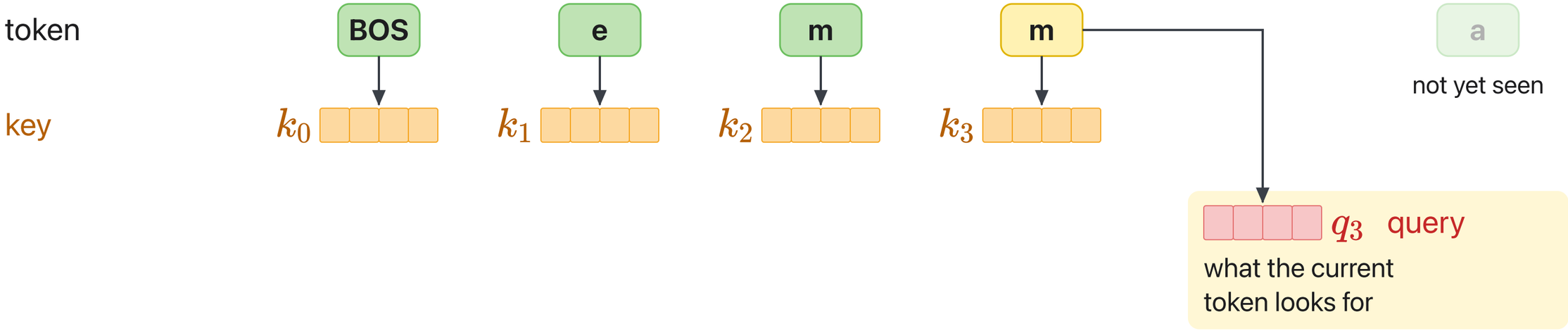
softmax turns scores into probabilities. It is the Boltzmann distribution with $E_i = -l_i$ and $k_B T = 1$: a higher score means a more probable state.

One attention head



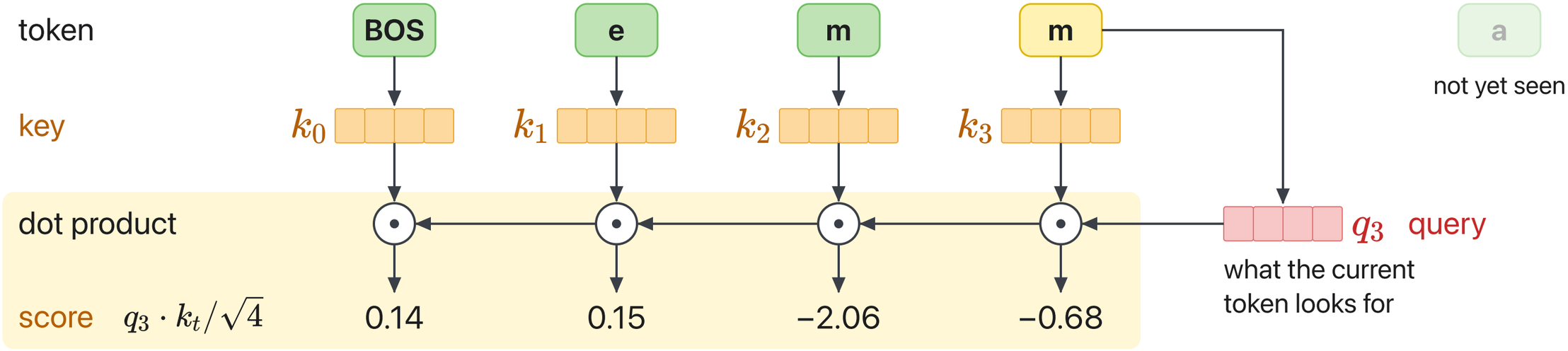
Every token seen so far has left a key in the cache: a short vector that says what that token contains.

One attention head



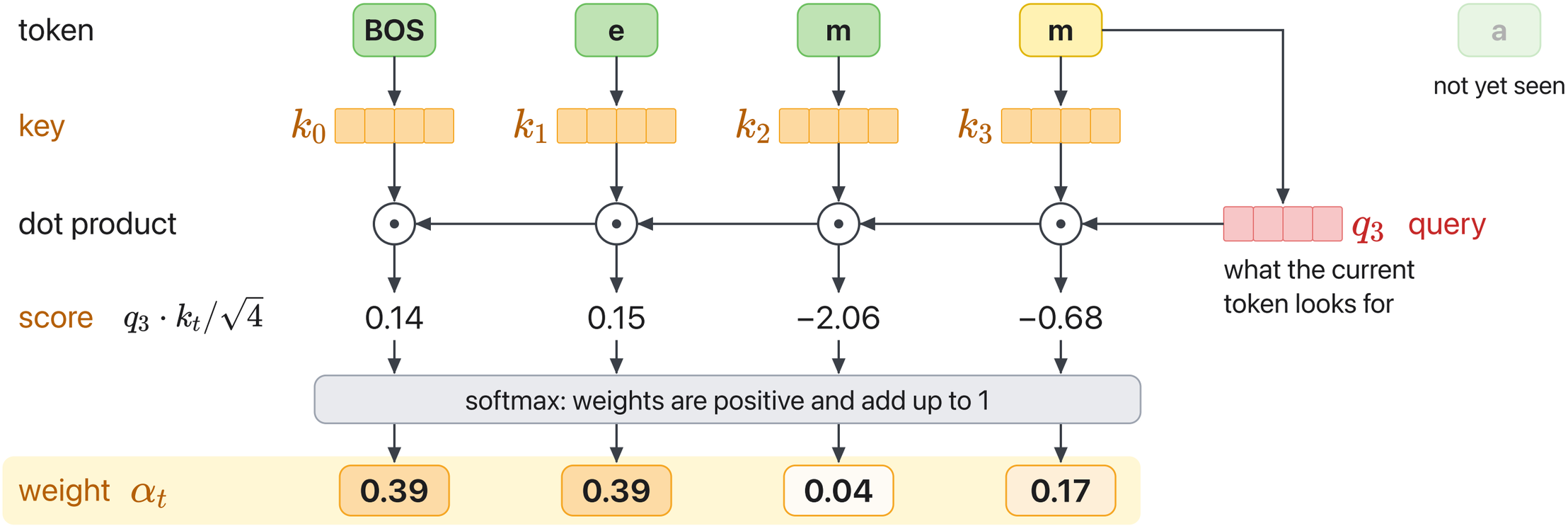
The current token makes a query: a vector that says what it is looking for.

One attention head



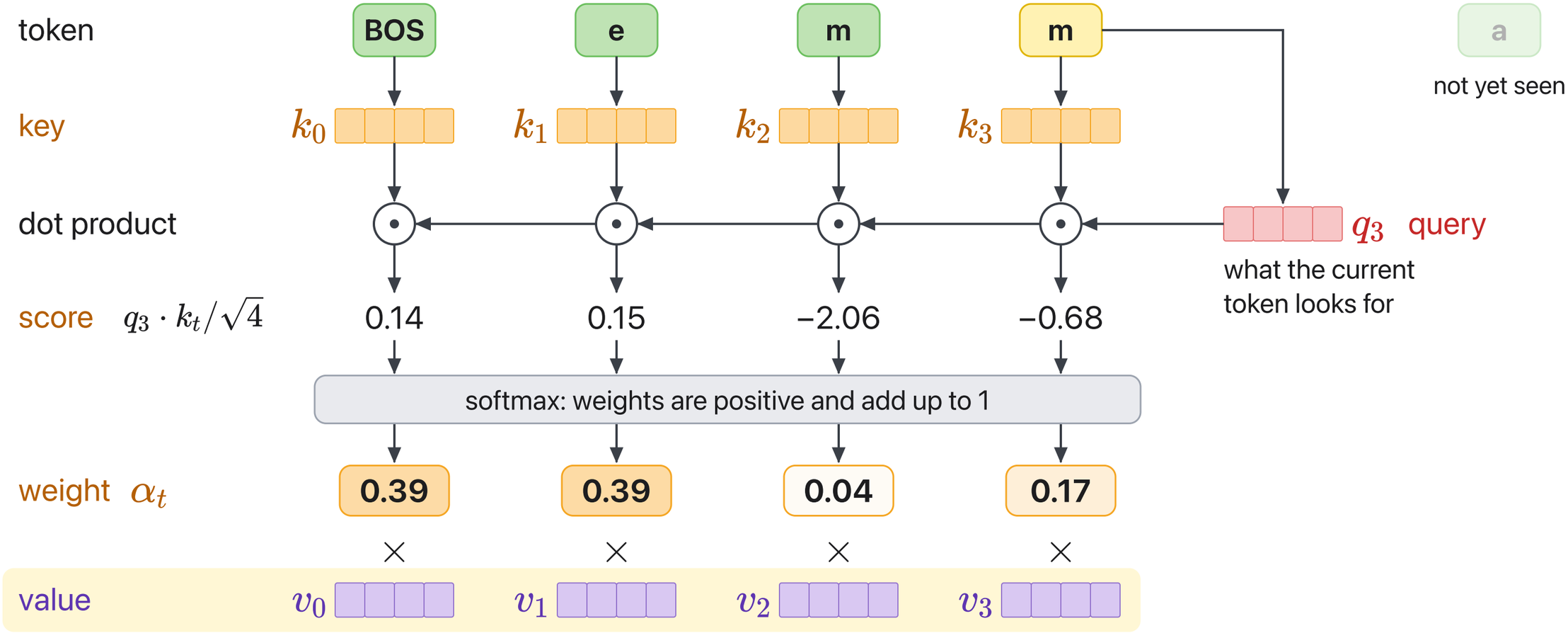
The dot product of the query with each key gives one score per token. A higher score means a better match.

One attention head



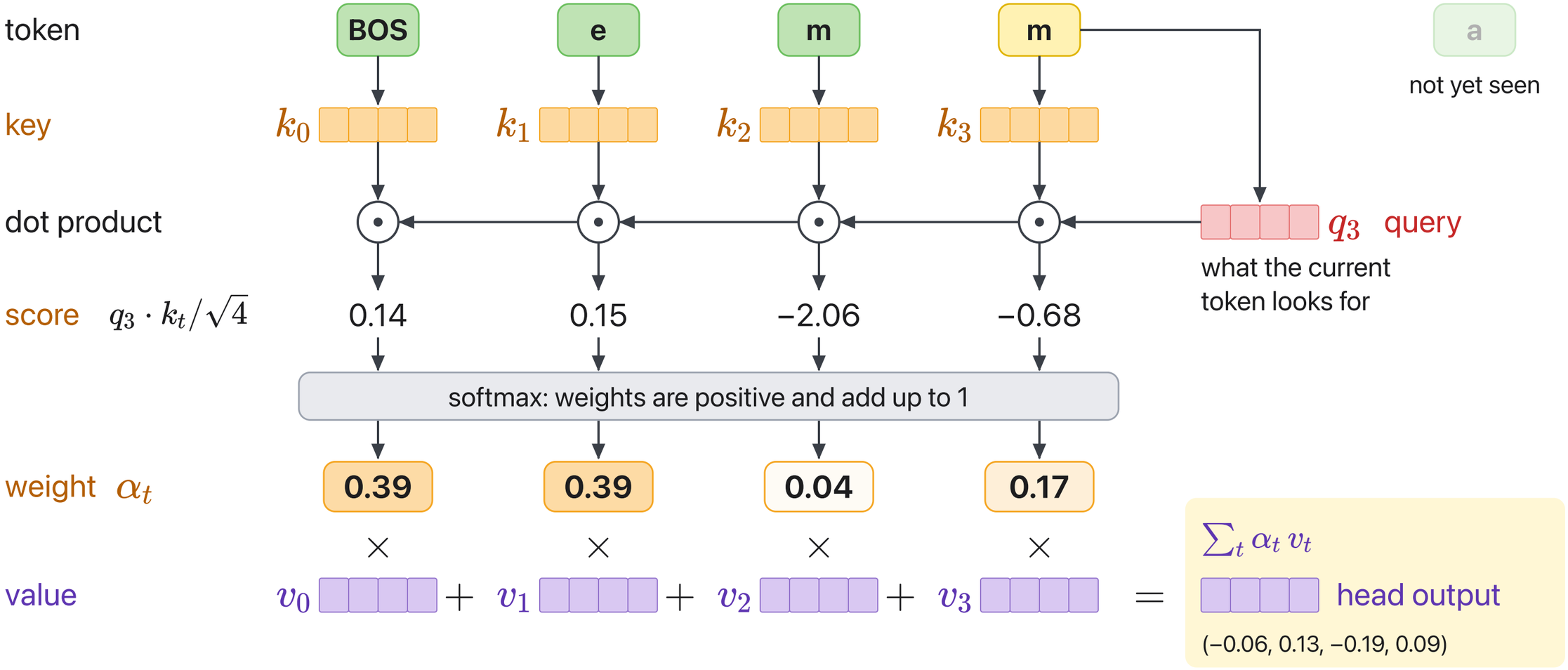
Softmax turns the scores into weights that are positive and add up to 1. The best-matching tokens get the largest weights.

One attention head



Every token has also stored a value: the information it passes on. Each value is multiplied by the weight of its token.

One attention head



The head output is the weighted average of the values: the current token collects information mostly from the tokens it attends to.

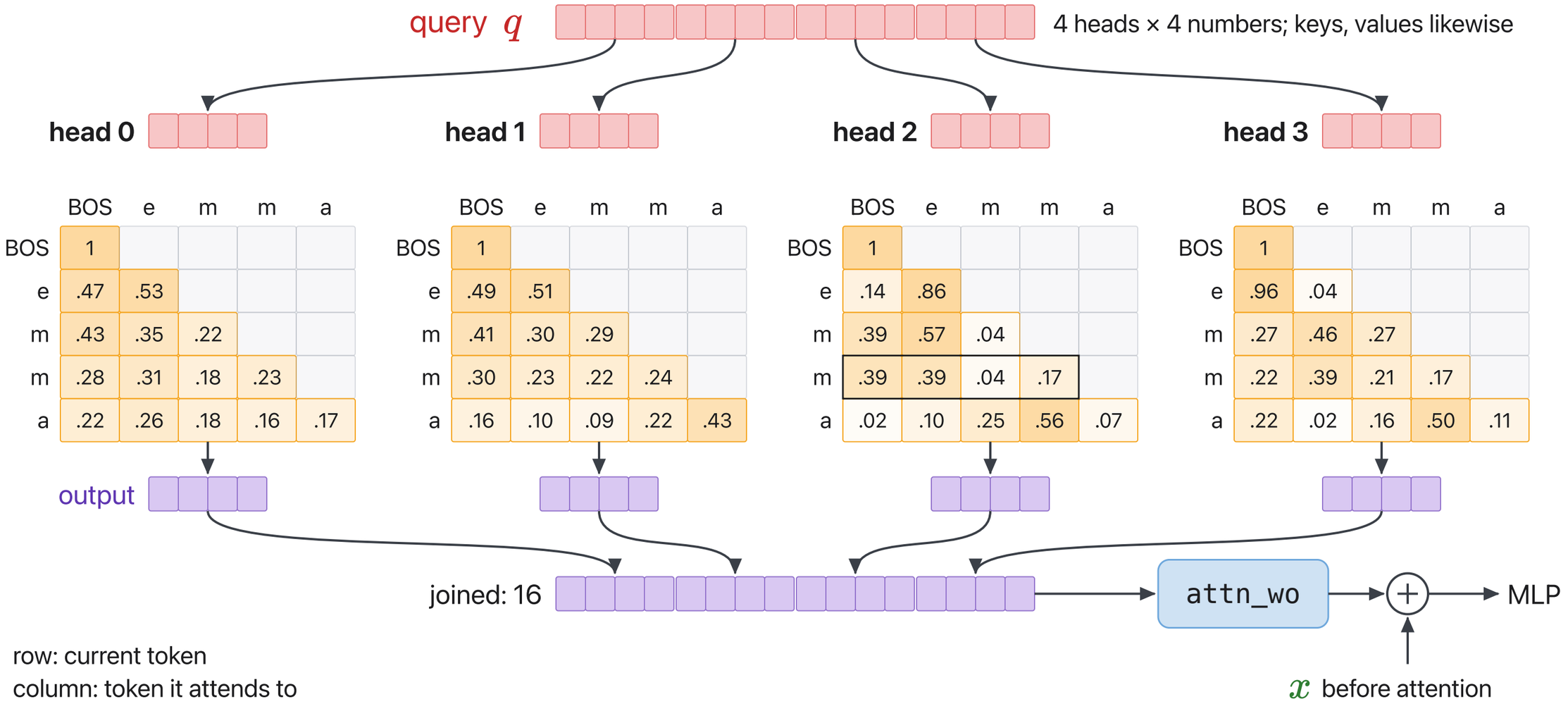
Attention

$$\alpha_t = \frac{\exp\left(q \cdot k_t / \sqrt{d}\right)}{\sum_{s \le \text{pos}} \exp\left(q \cdot k_s / \sqrt{d}\right)}, \qquad \text{out} = \sum_{t \le \text{pos}} \alpha_t \, v_t$$

q: query of the current token · k_t, v_t: key and value of token t · $d = 4$: dimension of one head

The previous figure in one line: a softmax over the tokens seen so far gives the weights α_t, and the output is the weighted average of their values.

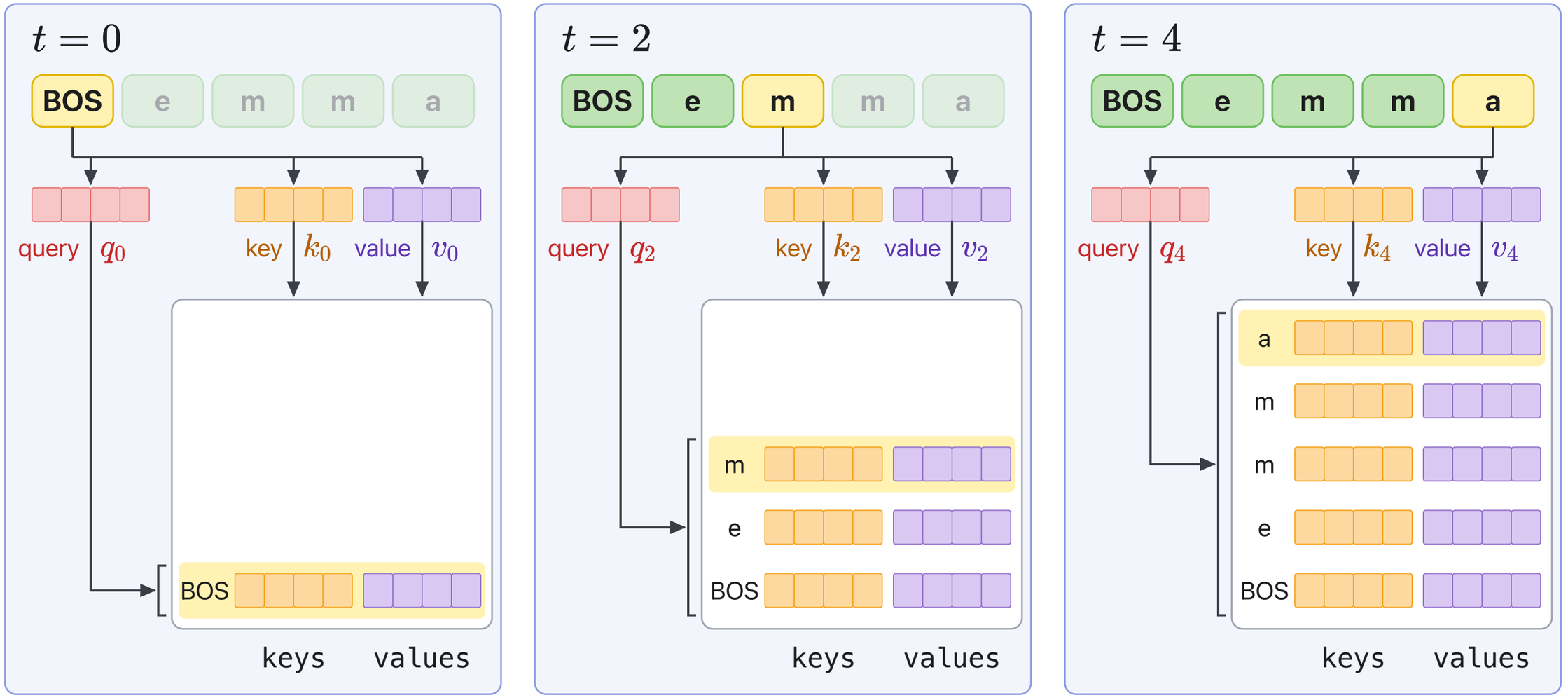
Four heads



row: current token
column: token it attends to

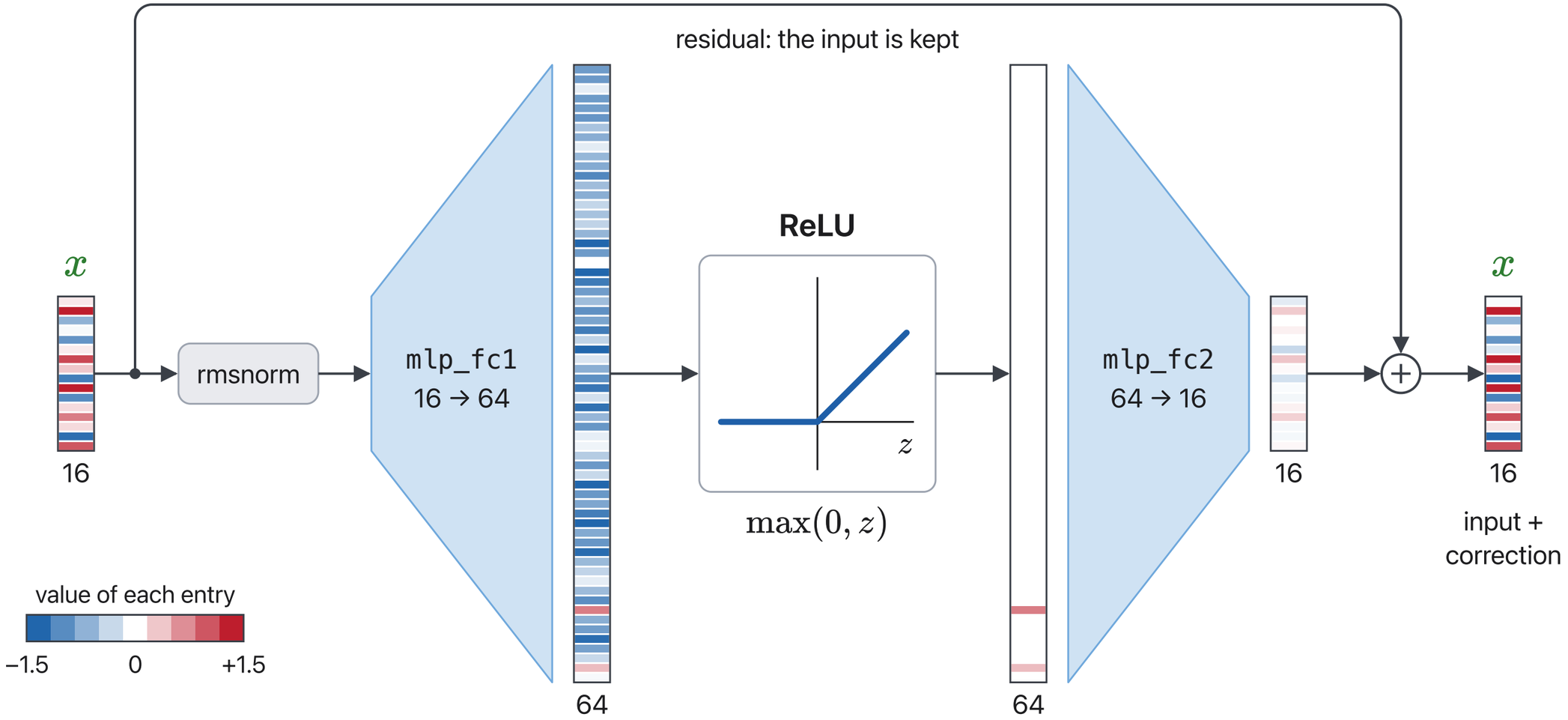
Each head works on its own 4 numbers and learns its own attention pattern. The four outputs are joined and mixed by one linear map.

KV cache



A token can attend only to what is already stored: itself and the earlier tokens. The future is not in the cache, so no mask is needed.

MLP and residual



Attention moves information between tokens; the MLP computes on one token alone. It widens the vector to 64, keeps only the positive entries, and adds the result to its input.

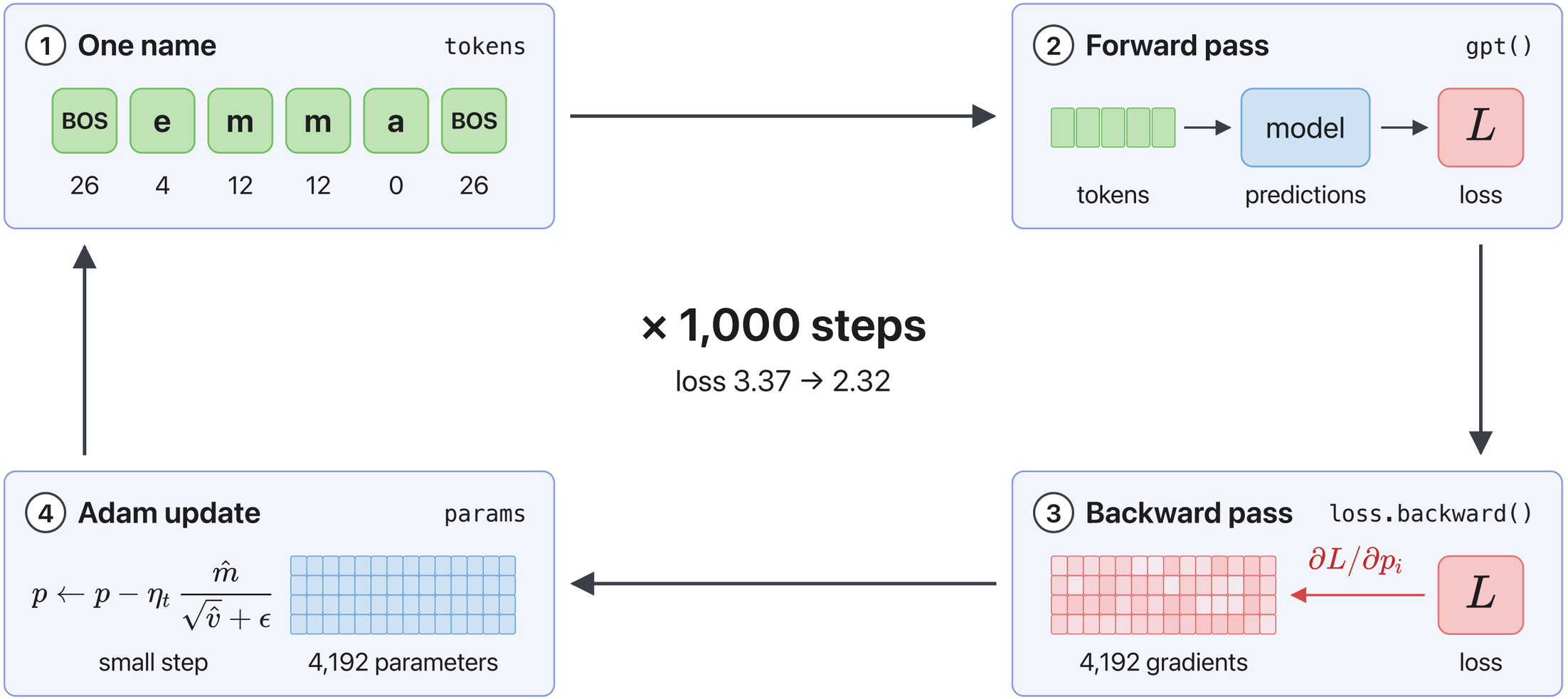
rmsnorm

$$\mathrm{rmsnorm}(x)_i = \frac{x_i}{\sqrt{\frac{1}{n}\sum_{j=1}^{n} x_j^2 + \varepsilon}}$$

$n = 16$: vector dimension $\cdot$ $\varepsilon = 10^{-5}$

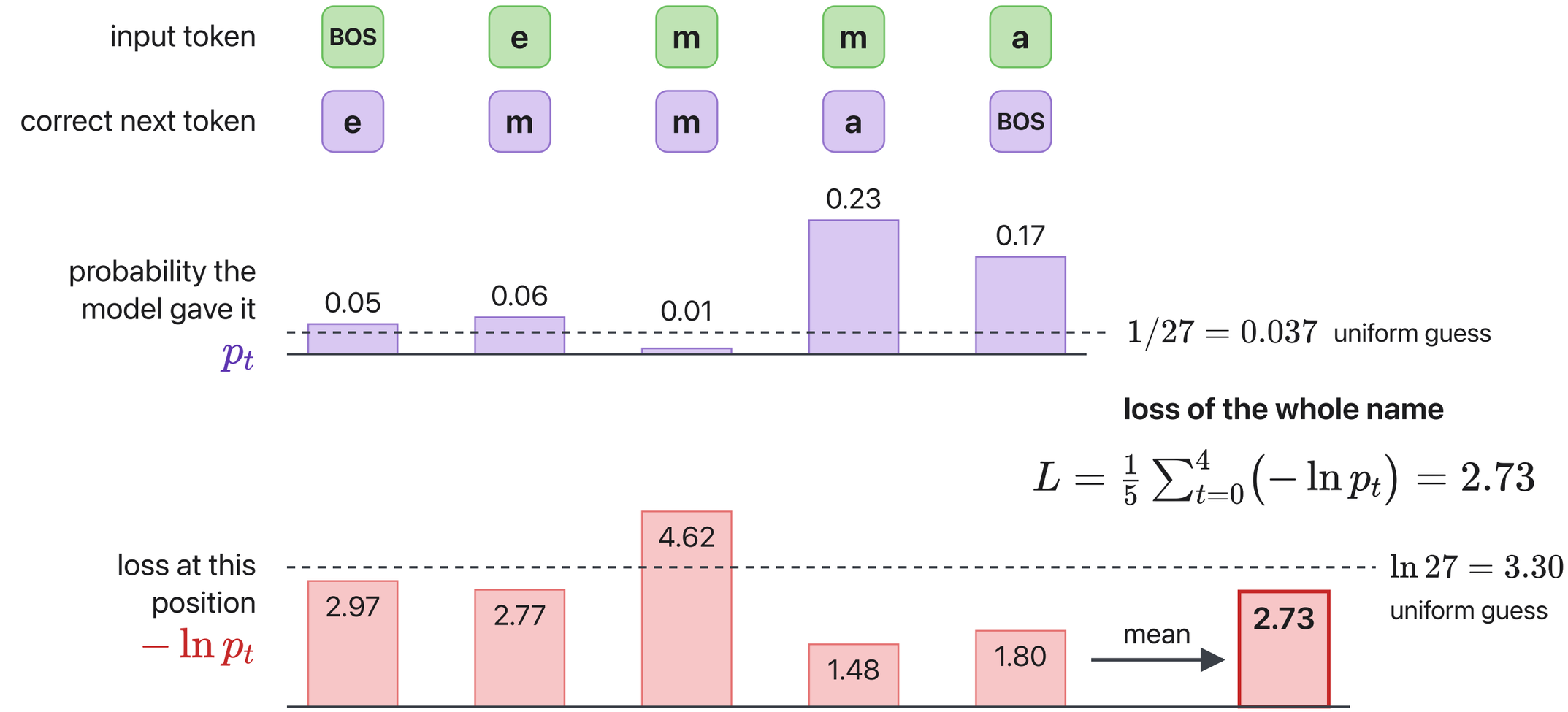
The vector keeps its direction and its root-mean-square is rescaled to 1. This keeps the numbers from growing or vanishing as they pass through the blocks.

Training loop



One step = one name, one forward pass, one backward pass, one small change to all 4,192 parameters. Repeating it 1,000 times is the whole training.

Loss



At each position the loss is $-\ln p$ of the correct next token: high probability costs little, low probability costs a lot.

The loss of the name is the mean, 2.73.

Cross-entropy

$$L = \frac{1}{n} \sum_{t=0}^{n-1} \Big[- \ln p_t(\text{target}_t) \Big]$$

n: number of positions in the name $\cdot$ $p_t(\text{target}_t)$: probability the model gave to the correct next token

The loss is low when the correct next character gets high probability. A uniform guess gives $\ln 27 \approx 3.30$; anything below that has been learned.

Adam

$$m \leftarrow \beta_1 m + (1 - \beta_1)\, g, \qquad \hat{m} = \frac{m}{1 - \beta_1^t}$$

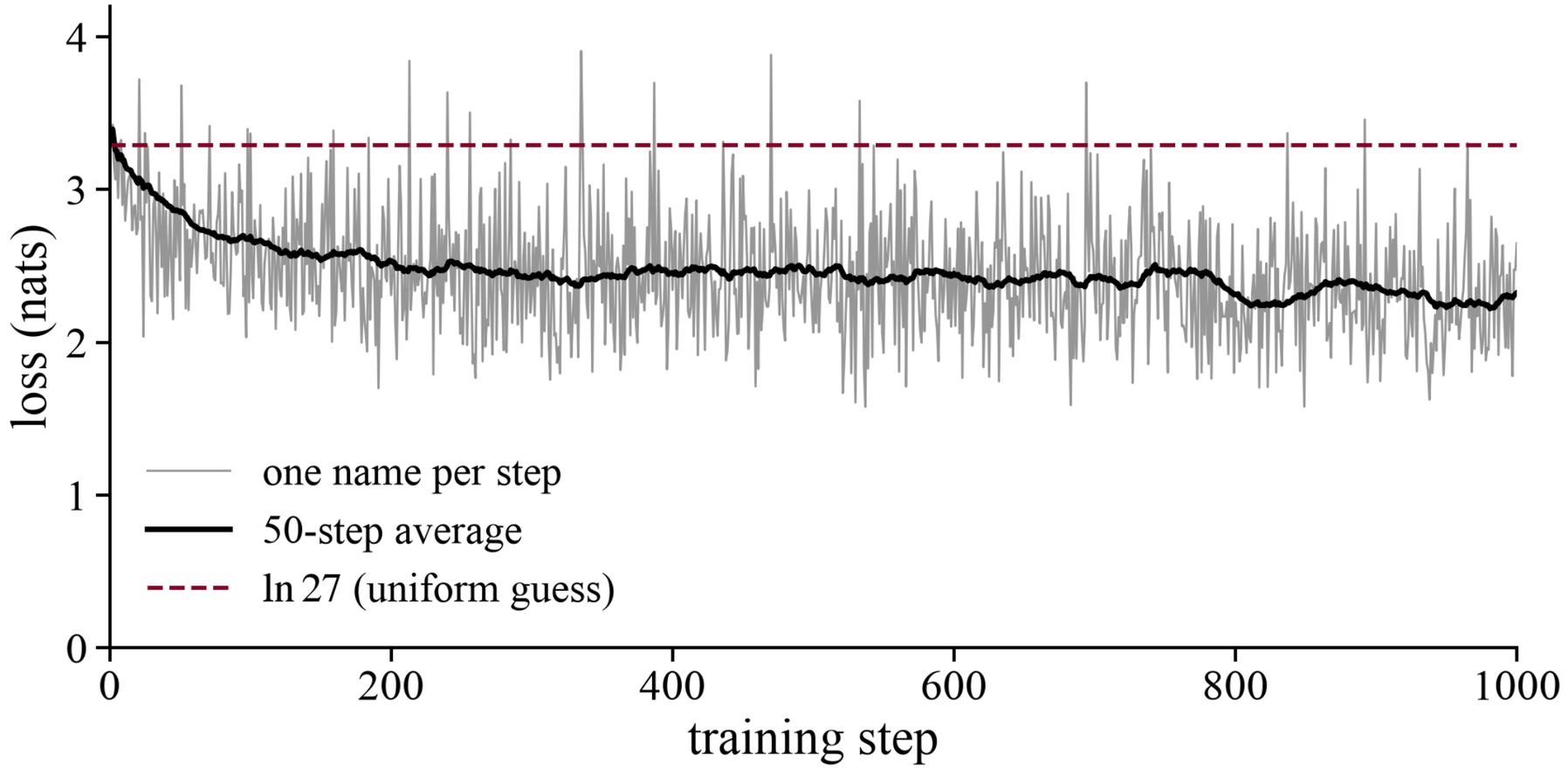
$$v \leftarrow \beta_2 v + (1 - \beta_2)\, g^2, \qquad \hat{v} = \frac{v}{1 - \beta_2^t}$$

$$p \leftarrow p - \eta_t \, \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon}$$

$$g = \partial L / \partial p \;\cdot\; \beta_1 = 0.85, \beta_2 = 0.99 \;\cdot\; \eta_t = 0.01\,(1 - t/1000)$$

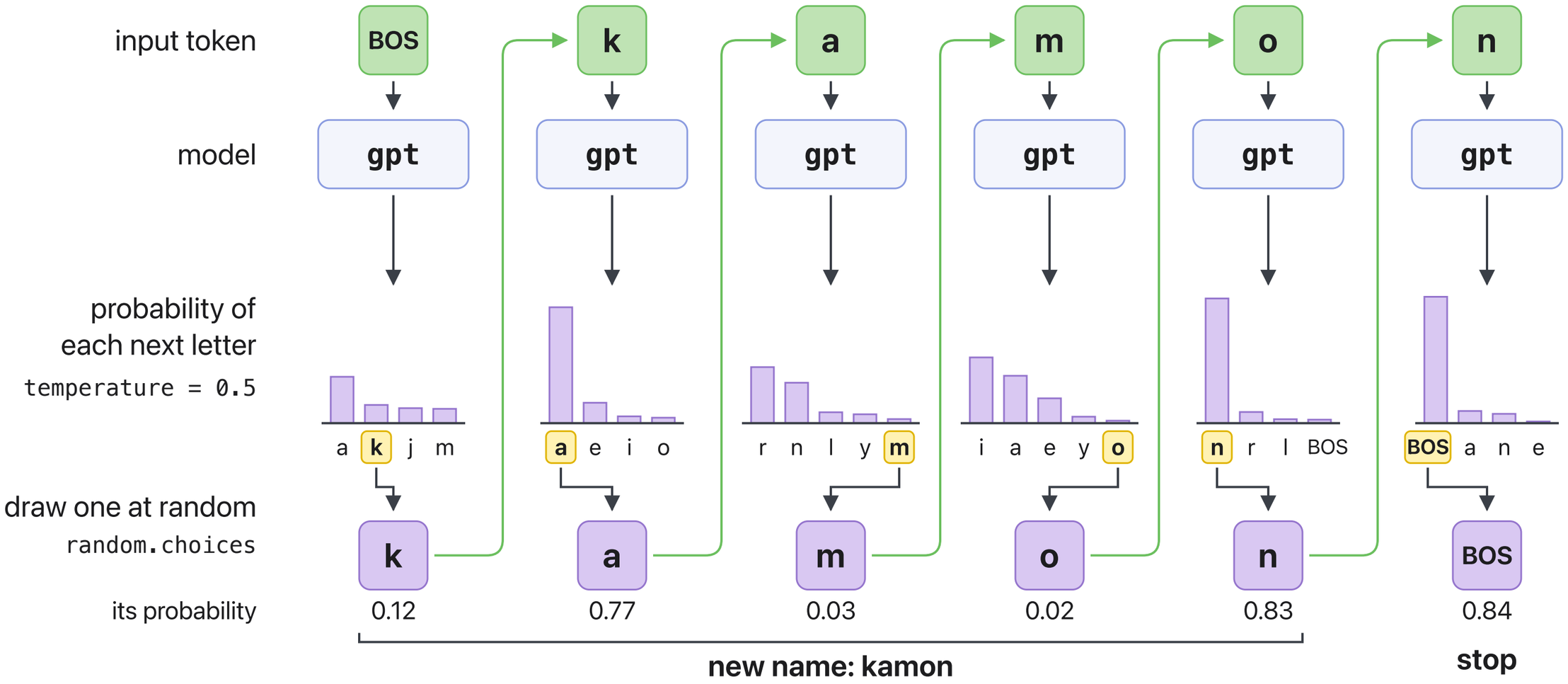
Gradient descent with two additions: m averages recent gradients (momentum), and $\sqrt{v}$ gives each parameter its own step size.

Loss curve



The average loss falls from $\ln 27 \approx 3.3$ to about 2.3 in 1,000 steps. It fluctuates because each step uses a different name.

Inference



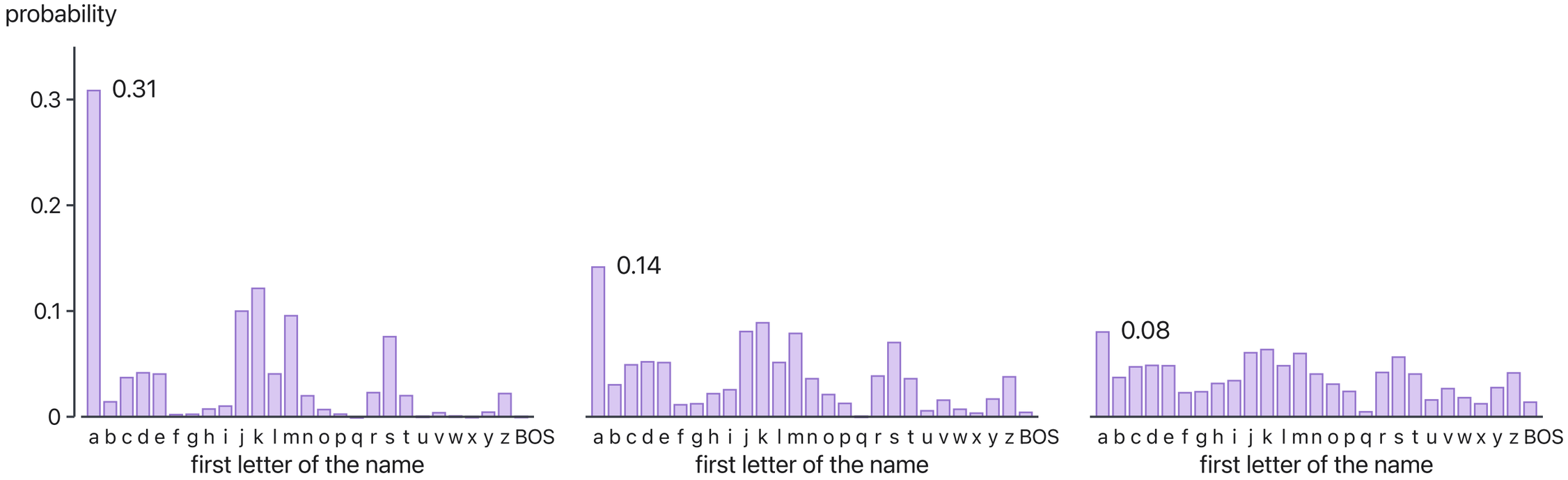
Generation is prediction in a loop: predict, draw one letter at random, feed it back as the next input. The name ends when BOS is drawn.

Temperature

$T = 0.5$ sharper

$T = 1$ as trained

$T = 2$ flatter



$$p_i = \frac{e^{l_i/T}}{\sum_j e^{l_j/T}}$$

logits l_i from the model

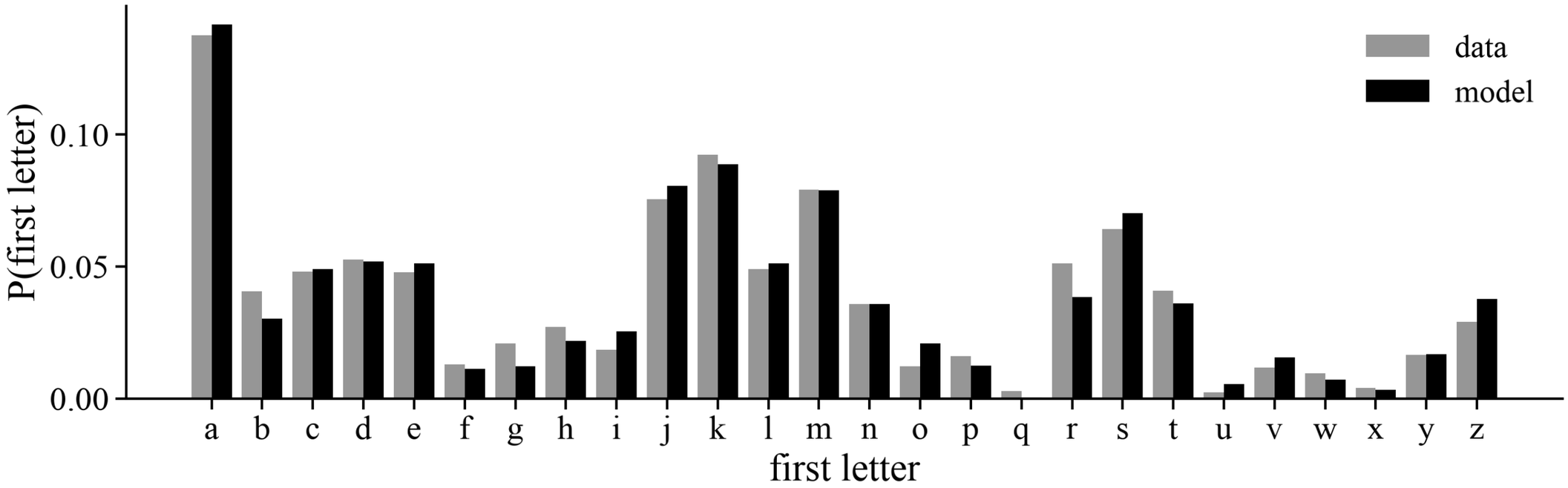
Low T concentrates probability on the most likely letters; high T flattens it. This is the Boltzmann distribution with energies $E_i = -l_i$.

Generated names

kamon	ann	karai	jaire	vialan
karia	yeran	anna	areli	kaina
konna	keylen	liole	alerin	earan
lenne	kana	lara	alela	anton

Eleven of these twenty are real names from the data; nine are inventions that only look like names. The model knows what is probable, not what exists: the same effect as a chatbot stating a plausible but false fact.

What it learned



Given only BOS, the trained model reproduces how often each letter starts a name in the data. The parameters have absorbed the statistics.

Progression

File	What it adds
`train0.py`	bigram count table, no neural network
`train1.py`	MLP, manual and numerical gradients, SGD
`train2.py`	autograd: the `Value` class
`train3.py`	position embedding, single-head attention, rmsnorm, residuals
`train4.py`	multi-head attention and the layer loop: the full GPT
`train5.py`	Adam: this is `microgpt.py`

Karpathy builds the file in six steps, each adding one idea. Reading them in order is a good way to study the code.

microgpt and production LLMs

	microgpt	Production LLM
Data	32,033 names	trillions of tokens of text
Tokenizer	27 characters	about 100,000 subword tokens
Autograd	one scalar at a time, in Python	tensors on GPUs
Architecture	4,192 parameters, 1 layer	10^{11} parameters, 100+ layers, RoPE, MoE, ...
Training	one name per step, 1,000 steps	millions of tokens per step, for months
Post-training	none	fine-tuning, reinforcement learning

Real LLMs are far larger and add many improvements. The core loop is the same: predict the next token, compute the loss, update the parameters.

Next week

- We read `microgpt.py` line by line in the notebook and train it (about two minutes)
- Experiments: temperature, first-letter statistics, attention maps
- Before class: read the post at karpathy.github.io/2026/02/12/microgpt

Today was the map. Next week every box in these diagrams becomes a few lines of Python.

References

- A. Karpathy, microgpt (2026): karpathy.github.io/2026/02/12/microgpt
- J. Alammar, The Illustrated Transformer; The Illustrated GPT-2: jalammar.github.io
- Transformer Explainer: poloclub.github.io/transformer-explainer
- B. Bycroft, LLM Visualization: bbycroft.net/llm
- 3Blue1Brown, Transformers and Attention: 3blue1brown.com/lessons/gpt