Dissecting microgpt
(for high school students and general readers)

Tracing, step by step, how an AI makes names, using only four names

Planned and edited by Yoshio Okamoto / Written by Claude Opus 5.5 (Anthropic)

Ver. 0.1 October 24, 2026

Note: The following content was generated by AI. The editor has checked it, but please be aware that it may still contain errors or mistaken assumptions.

© 2026 Yoshio Okamoto. Free to use for educational purposes (please cite the source). The microgpt program is the work of Andrej Karpathy.

Introduction

AI systems such as ChatGPT and Claude (large language models, or LLMs) repeat a very simple task. As a continuation of the text so far, they predict one next word, add it to the text, and predict the next one again. That is all.

microgpt is a Python program of about 200 lines, released by Andrej Karpathy in February 2026. It runs the same mechanism as a large AI, on a very small scale. Instead of sentences it handles English personal names: it learns to predict the "next letter" and, at the end, invents new names.

In this booklet, we let microgpt learn **only four names, emma, ava, mia and anna**, and dissect how it works from beginning to end. Making it this small lets us follow every number that moves inside the AI, one by one, on paper. All the numbers in the text were confirmed by actually running the program.

Chapter	Title	What we look at
1	Overview and preparation	What the AI practices: the data, letter numbers, practice problems
2	Reading one letter	Following the computation for emma letter by letter, with real numbers
3	Measuring misses and training	Measuring the penalty (loss) and turning the knobs little by little
4	Making names	Drawing lots by probability to invent new names
5	Comparing with real AI	Differences in size and in the way of computing, and what stays the same

The main text is written so that it can be understood without reading any formulas. The formulas are collected in the footnotes and in Appendix A. Reading this booklet together with "A Look Inside a Small AI's Head," which interprets the meaning of the 268 numbers (weights) after training with the same four names, and "Reading microgpt," which reads the 200 lines in order with the original settings, will deepen your understanding.[1]

- Program: https://gist.github.com/karpathy/8627fe009c40f57531cb18360106ce95
- Explanation by the author: https://karpathy.github.io/2026/02/12/microgpt/

[1] Some of the other documents referred to in this booklet are currently available in Japanese only.

Contents

1 Overview and preparation 3

2 Reading one letter 9

3 Measuring misses and training 17

4 Making names 26

5 Comparing with real AI 32

Closing 37

Appendix A Formulas 38

Appendix B Glossary 41

Appendix C How the numbers were obtained, and where to get the program 42

1 Overview and preparation

1.1 Even a large AI just repeats "predict the next one"

A large AI predicts one next word as a continuation of the text so far. After reading "once upon a" the next word is "time"; after reading "once upon a time" the next word is "there"…. Once it predicts a word, it adds it and predicts the next one again.[2] It is the same idea as the "predictive text" on a smartphone; the difference is that it reads much longer text and predicts much better.

microgpt does the same thing. Its material, however, is names instead of sentences, and the unit it predicts is a single letter (Table 1). Only the material and the unit are small; what it does is the same. With names, we can check the whole thing with our own eyes, so it is just the right size for studying the mechanism.

Table 1 A large AI and microgpt

	Large AI (LLM)	microgpt
What it learns from	a huge amount of text	English names (four in this booklet)
One document	one article or book	one name (such as emma)
Unit it predicts	a token (a piece of a word)	one letter
What it makes	a continuation of the text	new names

1.2 microgpt: a small AI made of about 200 lines

microgpt uses no special libraries such as NumPy or PyTorch; it is written only with the features built into Python. At the top of the program, the author writes, in effect, that "this file is the complete algorithm; everything else is just efficiency." It is a claim that the difference from a real AI is mainly size and speed, while the content of the computation is the same. We check this claim in Chapter 5.

The original microgpt learns from about 32,000 names, but this booklet uses a "small example" reduced to four names (Table 2). The program itself is not rewritten. We only changed four numbers and made the list of names four lines long.[3] When it is this small, we can read every number inside the AI on paper.

Table 2 The original microgpt and the small example of this booklet

	Original settings	Small example
Number of names to learn	32,033	4 (emma, ava, mia, anna)
Kinds of letters (+ separator mark)	27	7
Numbers used to represent one letter	16	4
Length it can read at once	16	5
Number of Attention "views" (heads)	4	1
Number of practice steps	1,000	300
Number of knobs (weights)	**4,192**	**268**
Computing time	2–3 minutes	about 2 seconds

[2] The unit that a large AI predicts at a time is not exactly a word but a piece of a word called a "token." In English, a token is often a word or part of a word.

[3] Line 76 `n_embd` 16 → 4, line 77 `block_size` 16 → 5, line 78 `n_head` 4 → 1, line 152 `num_steps` 1000 → 300. The names to learn are written one per line in a file called `input.txt`. These are the same settings as the small example in Section 5 of "Reading microgpt."

1.3 The program consists of six parts

The program consists of the following six parts, in order from the top (Table 3; the line numbers are those of the original 200-line `microgpt.py`).

Table 3 The six parts of the program

	Part	Lines	What it does
1	Data	14–21	reads the names and shuffles their order
2	Tokenizer	23–27	makes the rule that turns letters into numbers
3	Autograd	29–72	the mechanism that works out "which knob to turn, and which way"
4	Knobs and model	74–144	prepares the knobs (weights) and the main body that gives scores to the next letter
5	Training	146–184	measures how far off the prediction is and turns the knobs little by little
6	Generation	186–200	makes new names with the trained model

What knobs (weights) are Inside the AI there are 268 numbers, called **weights** or **parameters**. Like the knobs used to tune a radio, each of them is a number whose value can be changed by turning it.[4] At first they have random values (small random numbers), and each time practice is repeated, they are turned a little in the direction that makes predictions more likely to be right. The positions of the knobs when practice is over are everything the AI has "learned." The 268 knobs are arranged in nine tables (matrices). What is in the tables and how they are used is explained in Chapter 2; how the knobs are turned, in Chapter 3.

The practice loop Practice is easy to understand if we compare it to club practice (Figure 1). Solve a problem (predict the next letter with 4, the model), grade it (measure the miss with the loss), find the cause (with 3, autograd, find out which knob was how much to blame), and fix the form (with 5, Adam, turn the knobs just a little). One round is done with one name, and this is repeated 300 times.

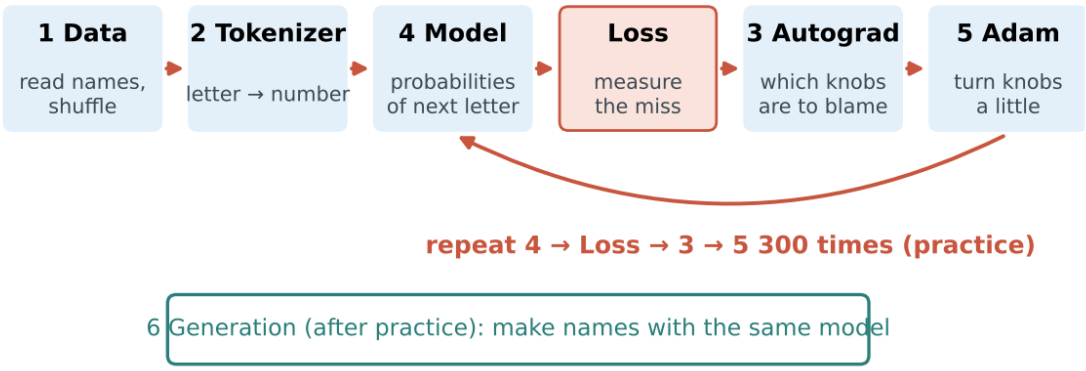


Figure 1 The parts of the program and the practice loop. The numbers are those of Table 3.

[4] "Knobs" is the same comparison as in "Reading microgpt." "A Look Inside a Small AI's Head" calls them "weights" arranged in tables. They are the same thing.

Practice and the real thing The same model is used in two situations. In practice (training), the model predicts the next letter one at a time while looking at the correct name, and the knobs are turned according to the miss. In the real thing (generation), there is no correct answer: starting from the start mark, the model predicts one letter at a time, picks a letter according to the probabilities and appends it. When "end" is picked, the name is complete; the knobs are no longer turned. The computation of the prediction (4, the model) is exactly the same in practice and in the real thing.

1.4 Training data: four names

For training we use the four names emma, ava, mia and anna, and treat one name as one "document." These four were chosen for reasons.

- **Few kinds of letters**: there are only six kinds, a, e, i, m, n and v, so the tables inside the AI are small and can be read in full.
- **There are scenes where it is "correctly unsure"**: after the start (a, e or m) and after a (v or n), the data themselves do not decide one answer.
- **There are letters with the same role**: the i of mia and the v of ava are both always followed by a. We can check whether the AI notices this.
- **All have four letters or fewer**: the length it can read at once is 5, so it can practice all the way to the final "end."

With unrelated names (for example sophia, mary, celine, elena), the next letter would be determined uniquely everywhere except at the first letter, and only memorization would be visible. That is why we chose names that share letters.

Shuffling The program first rearranges the order of the names it has read at random (shuffling), and uses them for practice one at a time in that order. With random seed 42, the order is mia, ava, anna, emma. When the end is reached it goes back to the beginning, so in 300 steps every name is practiced exactly 75 times.[5]

1.5 Collecting the kinds of letters and numbering them

The AI cannot handle letters directly, so the letters are given numbers. We collect the letters that appear in the four names, remove duplicates, sort them alphabetically, and number them from 0. Finally, the separator mark **BOS** (beginning of sequence) gets the next number, 6 (Table 4).[6]

Table 4 Letters and numbers

Letter	a	e	i	m	n	v	BOS
Number	0	1	2	3	4	5	6

If the names to learn are changed, the kinds of letters and their numbers also change. With the original roughly 32,000 names, there are 26 kinds, a–z, plus BOS, 27 in all. Letters that are not in the list (such as b or z) do not exist for this AI.

[5]Random numbers on a computer are computed from a starting number called the "random seed," and the same seed always gives the same sequence. microgpt fixes the seed at 42 (line 12), so whoever runs it, step 1 uses mia. Changing the seed changes the shuffled order, the initial values of the knobs, and the lots drawn when making names.

[6]In the program, `uchars = sorted(set(''.join(docs)))` (line 24) collects and sorts the letters, and `BOS = len(uchars)` (line 25) decides the number of BOS. This rule for going back and forth between letters and numbers is called the tokenizer.

BOS: one mark for both start and end When practicing a name, BOS is attached before and after it. The BOS in front is the signal "a name starts here," and the AI reads BOS to predict the first letter of the name. The BOS behind is the answer "the name ends here"; when the AI predicts BOS, the name is complete. Like the dividing line between names in a list of names, one line marks both the end of the name above and the start of the name below.

Turning the four names into sequences of numbers In this way, the four names become the following sequences of numbers.[7] From here on, all computation is done with these sequences of numbers.

emma	$\rightarrow$	$[6,1,3,3,0,6]$	ava	$\rightarrow$	$[6,0,5,0,6]$
mia	$\rightarrow$	$[6,3,2,0,6]$	anna	$\rightarrow$	$[6,0,4,4,0,6]$

All the AI knows is numbers: the AI knows neither the shape nor the sound of a. Even if the names of all letters were swapped at once (calling a "red," e "blue," …), what it learns would be exactly the same. It is like a soccer record that contains only shirt numbers: from it you can still learn that "passes often go from number 7 to number 10." Using letters as the unit is microgpt's choice; large AIs use "tokens," which group frequent combinations of letters, as the unit.

1.6 Making practice problems

From one name, several problems of the form "having read up to here, what comes next?" can be made. From emma, the five problems of Table 5 are made. What has been read always starts from the beginning of the name. The AI can look back, but it cannot look ahead (at the answer). The final BOS is only an answer; there is no problem that reads it to predict the next one.[8]

Table 5 Practice problems made from emma

Problem	What has been read	Next letter (correct answer)
1	start (BOS)	e
2	e	m
3	em	m
4	emm	a
5	emma	end (BOS)

A four-letter name gives five problems and a three-letter name four, so the four names give $5+4+4+5=18$ problems. Of these, "start" appears in all four names and "a" in both ava and anna, so combining identical "what has been read" gives **14 scenes**.

The tree of names Combining the four names into one tree gives Figure 2. Following it from the BOS on the left to the right gives one of the four names. There are only two places where the branches split: after the start, and after a. In the other twelve scenes, the next letter is decided uniquely.

[7] In the program this is `tokens = [BOS] + [uchars.index(ch) for ch in doc] + [BOS]` (line 157). `uchars.index(ch)` is the command that finds "which number the letter ch is."

[8] In one practice step, all the problems of one name (five for emma) are solved, and the knobs are turned according to the average miss.

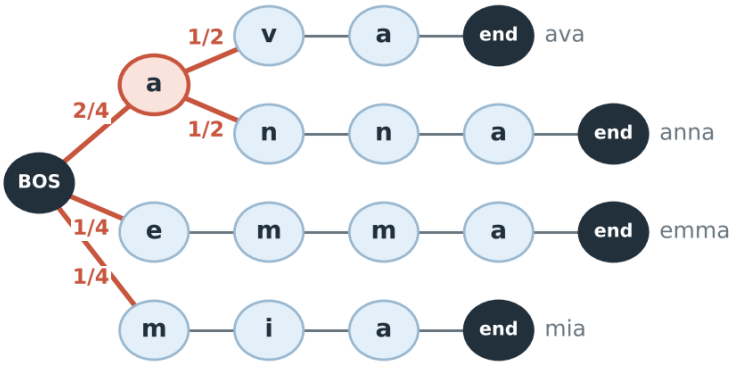


Figure 2 The tree of the four names. "end" is the final BOS. The red numbers are the shares of the data that go along each branch.

1.7 Correctly unsure: scenes where the answer is not unique

After the start, two of the four names begin with a (ava, anna) and one each with e and m. After a, there is one v (ava) and one n (anna) (Figure 3). In such scenes, rather than choosing one, the best answer is to follow the shares in the data: "a 50%, e 25%, m 25%."[9] It is like putting two lots for a and one each for e and m into a lottery box.

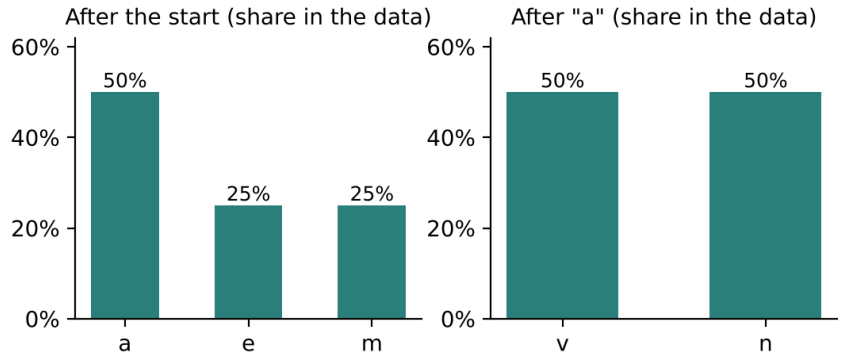


Figure 3 The two scenes where the answer splits (shares in the data).

However much it practices, the uncertainty never becomes 0: in 12 of the 18 problems the answer is unique, so the uncertainty comes only from the 6 problems after the start and after a. Expressed as "how many choices it is unsure among on average," it is 7 choices before practice (a random guess among 7 kinds), and the limit set by the data is 1.36 choices.[a] However clever the AI becomes, it cannot go below 1.36 choices. After 300 practice steps it is 1.41 choices, quite close to this limit (Chapter 3).

[a] Precisely, it is the reciprocal of the "multiplicative average" (geometric mean) of the probabilities given to the correct letters. The limit 1.36 was computed from the average penalty 0.308 (Appendix A.1).

1.8 Settings decided by people in advance

Like "how many servings to make" and "how many minutes to simmer" in cooking, there are numbers decided before starting (Table 6). Unlike the knobs (weights), they do not change with

[9] "Best" means that the average penalty (loss), explained in Chapter 3, is smallest (Appendix A.1).

practice. The technical term is **hyperparameters**. Because the length it can read at once is 5, names of five or more letters are cut off at the end and cannot be practiced all the way to the final "end." That is why all four names have four letters or fewer.

Table 6 Settings decided by people in advance

Setting	Name in the program	Small ex.	Original	Meaning
Numbers per letter	`n_embd`	4	16	width of the "blackboard" (Ch. 2)
Length read at once	`block_size`	5	16	start + up to 4 letters
Attention views	`n_head`	1	4	explained in Ch. 2
Practice steps	`num_steps`	300	1000	75 per name
How far to turn knobs	`learning_rate`	0.01	0.01	explained in Ch. 3
Random seed	`random.seed`	42	42	same seed, same result

Summary of Chapter 1: the AI repeats "predict the next one." The program consists of six parts; after repeating practice (predict → grade → find the cause → turn the knobs) 300 times, it makes names for real. The names become sequences of numbers and 18 practice problems. The answer splits only after the start and after a, and there, being correctly unsure is the best answer.

2 Reading one letter

In this chapter, we follow, with real numbers, the computation from the moment the AI reads one letter until it gives the probabilities of the next letter. We use the knobs after 300 practice steps (the trained weights). Because the predictions are mostly right, the meaning of the computation is easy to see. How the knobs were turned (training) is the subject of Chapter 3.

2.1 The flow of one prediction: five stages

Each time it reads one letter, the AI computes in the five stages of Figure 4.[10] These five stages are carried out at each of the five positions (BOS, e, m, m, a). The same knobs are used at every position.

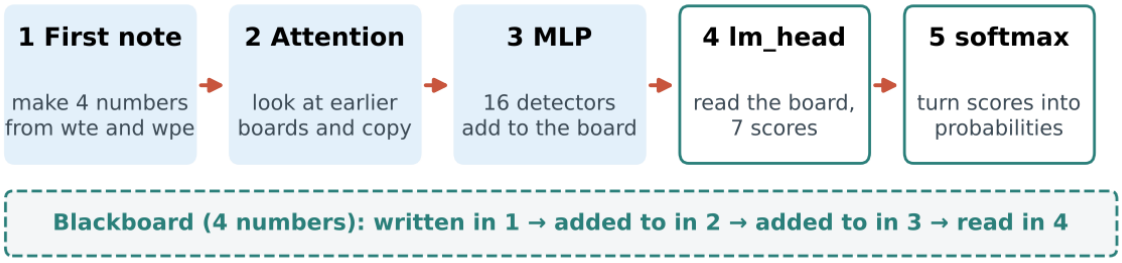


Figure 4 The five stages of one prediction.

2.2 The blackboard inside the AI

For each letter position there is one blackboard on which four numbers can be written; for emma, there are five (Figure 5). First the first note is written, and then Attention and the MLP add their results **by addition**. What was written before is not erased. Finally, lm_head reads the blackboard and turns it into scores for the next letter.[11]

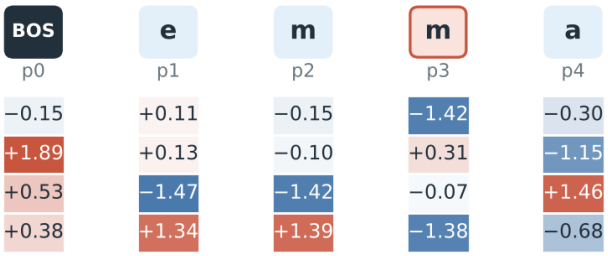


Figure 5 The first notes written on the five blackboards (after training). Red is positive, blue is negative.

The blackboard and the tables of knobs are different things The blackboard is added to within one prediction, and a new blackboard is used to read the next name. It does not remain after practice. On the other hand, the nine tables (knobs) are turned at every practice step and

[10] In the program, stages 1–4 are the function `gpt()` (lines 108–144) and stage 5 is `softmax()` (lines 97–101).

[11] The technical term for this blackboard is the "residual stream." Adding is the addition $\boldsymbol{x} \leftarrow \boldsymbol{x}+$ (result of the computation); in the program it is `x = [a + b for a, b in zip(x, x_residual)]` (lines 134 and 141).

remain after practice as the 268 weights. The blackboard is scratch paper for each problem; the tables are your own notebook, rewritten little by little at each practice. What we follow in this chapter is the scratch paper (blackboard), with the notebook (tables) fixed at its trained state.

The position we follow: the second m of emma We follow the position where "emm" of emma has been read (p3). Although it reads the same m, after the first m (p2, "em" read) the next letter is m, and after the second m (p3) the next letter is a. This problem cannot be decided by the letter alone; it cannot be solved without using "which letter in the name it is" and "what came before." The AI tells the two apart and gives a 98.8% at p3. Watch which stage uses this information.

2.3 Stage 1: the first note

wte: a dictionary with four numbers for each letter wte is a table that assigns four numbers to each of the seven kinds of letters (Figure 6, left). Like looking up a dictionary, the row of the letter being read is taken out. Reading m gives the row of m, $(-0.06, +0.03, -0.12, +0.00)$. The two m's of emma both use the same row of m. With wte alone, the two m's cannot be told apart. The values of the four numbers were decided by practice, and each single number has no fixed meaning.[12]

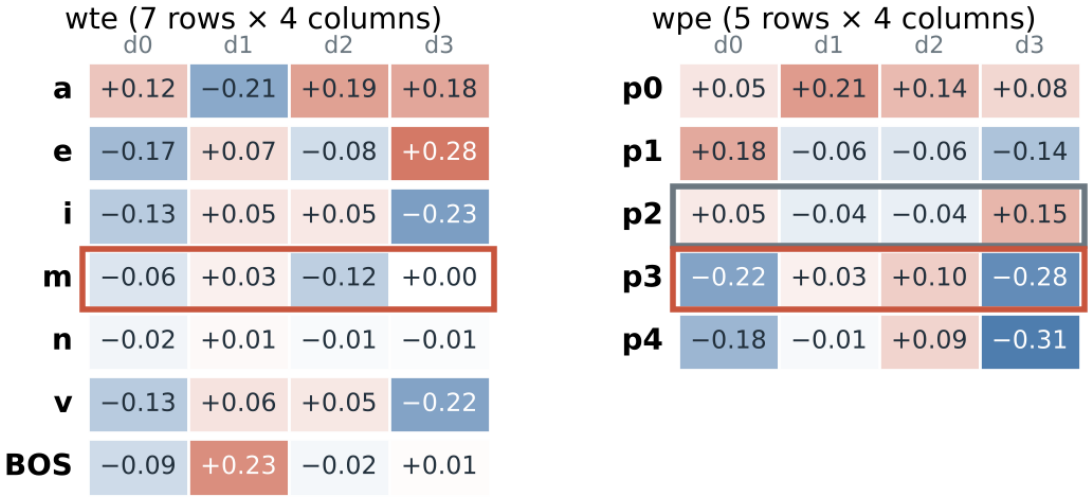


Figure 6 The trained wte (left) and wpe (right). The frames show the row of m and the row of p3 used at p3 (gray: the row of p2).

wpe: a table with four numbers for each position wpe is a table of four numbers according to "which letter in the name it is" (Figure 6, right). p0 is the position of the start (BOS), p1 is the first letter, …, p4 is the fourth letter. The first m uses the row of p2, the second m the row of p3. Here, for the first time, the two m's become different. It is like runners in a relay: the same runner has a different job as the second or third runner. There is a table of positions because Attention has no ability to tell the order of things.[13]

[12] d0–d3 are the four columns of numbers (the four axes of a four-dimensional space). Rotating the directions of the axes does not change how the AI works, so we cannot read things like "because d2 is negative…." For details, see Sections 3–4 of "A Look Inside a Small AI's Head." wte stands for "word token embedding."

[13] Attention is a computation that looks over the earlier positions "put together in a bag," so by itself it cannot hold information about order. wpe stands for "word position embedding."

Adding the numbers of the letter and the numbers of the position The same columns of the two tables (d0 with d0, d1 with d1, …) are added (Figure 7). The sum for p2, $(-0.02, -0.01, -0.15, +0.15)$, and the sum for p3, $(-0.28, +0.06, -0.01, -0.28)$, have become completely different numbers. The information about the letter (m) and the information about the position (third letter) are mixed into one set of four numbers.

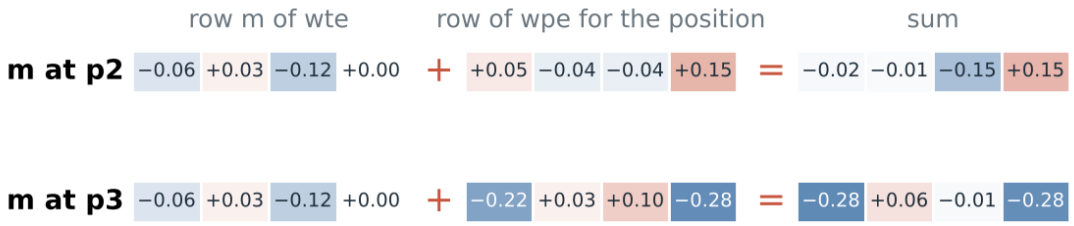


Figure 7 Adding the row of wte and the row of wpe (the m at p2 and p3).

Evening out the size (rmsnorm) The four added numbers are evened out in size before being written on the blackboard. For p3: (1) squaring each of the four numbers and averaging gives $\{(-0.28)^2 + 0.06^2 + (-0.01)^2 + (-0.28)^2\} \div 4 = 0.0402$; (2) its square root is 0.200; (3) dividing each of the four numbers by 0.200 gives $(-1.42, +0.31, -0.07, -1.38)$. This is the first note of p3.

After dividing, the average of the squares of the four numbers is always 1. The direction of the set of numbers does not change; only the size is evened out.[14] With this, those who read the blackboard later can judge by the arrangement (direction) alone, without being misled by the size of the numbers. Like the automatic volume control of a recording, it brings loud and quiet voices to the same volume, so that what is being said is easier to hear.

Comparing first notes Even for the same m, the first notes at p2 and p3 point in completely different directions (alignment of directions, cos, -0.41).[15] Conversely, the different letters e (p1) and m (p2) got notes with almost the same direction (cos 0.98). This is because, for both, "the next one is m." Through practice, the closeness of the notes comes to express not the shape of the letters but the closeness of their roles: "what comes next."

2.4 Stage 2: Attention

A mechanism like doing research in a library Attention is a mechanism that goes to look at the blackboards of earlier positions and copies what it needs. Compared to a library, the blackboard of each position is a book, and each has a "heading" (key, k) and "contents" (value, v). The current position makes a note of "what it is looking for" (query, q) and does the following three things.

1. Compare the query q with the heading k of each position. The more similar the heading, the higher the score.
2. Turn the scores into shares (100% in total). This decides how much of each book to read.

[14] rmsnorm stands for "root mean square normalization" (lines 103–106). In terms of the length of an arrow, with four numbers the length is always 2 ($= \sqrt{4}$). To avoid dividing by 0, in practice 0.00001 is added to the average before taking the square root (Appendix A.2).

[15] cos (cosine) is how well the directions of two sets of numbers line up, viewing each set as an arrow. It is +1 for the same direction, 0 for a right angle and −1 for opposite directions (Appendix A.2).

3. Mix the contents v according to the shares, copy them, and add them to its own blackboard. It can only look at itself and the positions before it. The letters ahead (the answer) are not yet visible. At p3, it can look at the four blackboards of BOS, e, m and m.

Making q, k and v The four numbers on the blackboard are multiplied by each of three tables, wq, wk and wv, to make q, k and v.[16] The query at p3 is q = $(+1.33, +0.38, -1.19, +1.44)$. k and v were made when each position was read, and kept for the later positions.[17]

Measuring how similar the query and the headings are The score is q and the k of each position multiplied column by column and added (the sum of products), divided by 2 (Table 7).[18] The sum of products with the heading of BOS is $(+1.33)(+1.27) + (+0.38)(+0.09) + (-1.19)(-1.19) + (+1.44)(+1.56) = +5.38$. The more the signs of the four numbers agree, the larger the sum of products. q and the heading of BOS had the same sign in all four. The highest score is for BOS (the start): the m at p3 is trying to look at the blackboard of the start of the name.

Table 7 The query at p3 and the heading of each position

Position looked at	Heading k	Sum of products	÷2 = score	e^{score}	Share
BOS (p0)	$(+1.27, +0.09, -1.19, +1.56)$	+5.38	**+2.69**	14.76	**76.2%**
e (p1)	$(-0.09, +1.17, +0.11, -0.29)$	−0.22	−0.11	0.90	4.6%
m (p2)	$(-0.03, +1.26, +0.06, -0.26)$	+0.00	+0.00	1.00	5.2%
m (p3, itself)	$(+0.56, -0.16, -0.55, +0.45)$	+1.99	+1.00	2.71	14.0%
Total				19.36	100%

Turning the scores into shares (softmax) We raise e (about 2.718) to the power of each score and divide by the total (the two right columns of Table 7). A score higher by 1 means looking about 2.7 times as much. The m at p3 directed 76% of its attention to the start of the name (BOS) (Figure 8, left). Like dividing up one's time, it reads the books with higher scores longer, but reads the other books a little too.[19]

Mixing and copying the contents, and adding them to the blackboard The contents v of the four positions are mixed according to the shares: $76.2\% \times \boldsymbol{v}(\text{BOS}) + 4.6\% \times \boldsymbol{v}(\text{e}) + 5.2\% \times \boldsymbol{v}(\text{m}) + 14.0\% \times \boldsymbol{v}(\text{m}) = (-1.11, +0.76, -0.18, -1.18)$. This is turned into the form of the blackboard with the table wo,[20] giving $(-0.97, +2.22, -0.58, -0.13)$, which is added to the first note. The blackboard became $(-2.38, +2.53, -0.65, -1.51)$. Most of what was copied is the contents of the blackboard of the start of the name. The information "an m that is the third

[16]"Multiplying by a table" is the computation that multiplies each row of the table and the four numbers on the blackboard column by column, and adds them up (the product of a matrix and a vector). Example: the first number of q = the product of row 1 of wq and the blackboard $= (-0.41) \times (-1.42) + (+0.64) \times (+0.31) + (+0.12) \times (-0.07) + (-0.40) \times (-1.38) = +1.33$. The table has four rows, so four numbers are made.

[17]Lines 121–122. The mechanism of keeping the k and v of earlier positions is the same as what large AIs call the "KV cache" (Chapter 4).

[18]Dividing by 2 means dividing by the square root of the number of numbers (4). This keeps the sum of products from becoming too large when there are many numbers (Appendix A.3).

[19]softmax is the computation that turns differences in scores into "how many times as much" and shares them out so that the total is 1. The same computation is used again in the final stage 5 (Appendix A.3).

[20]Multiplying by wo is the computation that rewrites the mixed contents to fit the four columns of the blackboard (line 133). The four tables wq, wk, wv and wo together make up the Attention part ($4 \times 4 = 16$ knobs each, 64 knobs in all).

letter from the start" has been added to the blackboard of p3, and the following MLP can use it as a clue.

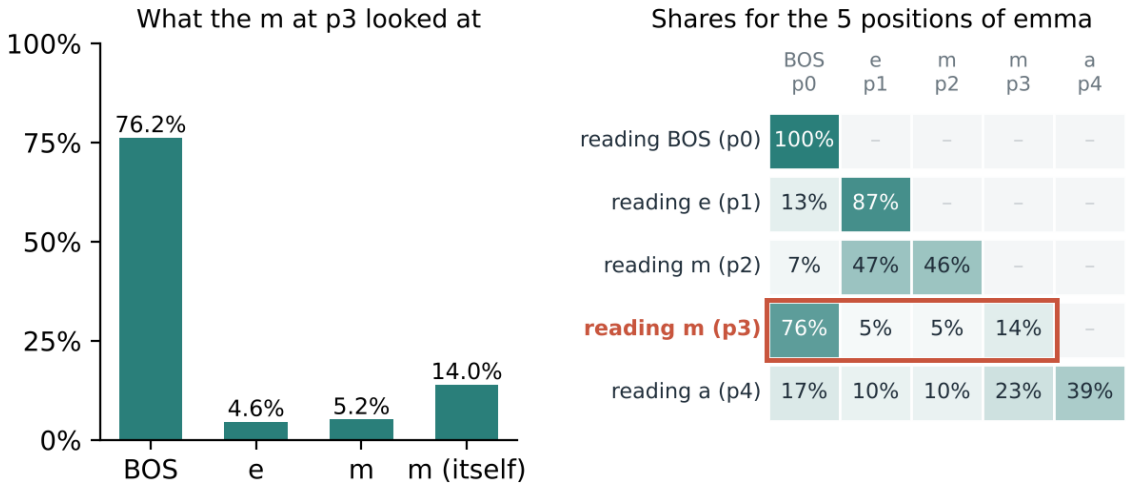


Figure 8 Attention shares. Left: the shares the m at p3 looked at. Right: the shares the five positions of emma looked at (each row totals 100%; "–" is a position ahead that has not been read yet).

Where did the five positions of emma look? In Figure 8, right, the "–" in the upper right are positions ahead that have not been read yet and cannot be looked at. The start (p0) can only look at itself, so it is always 100%. Therefore, the prediction at the start is exactly the same computation for every name. Only the m at p3 looks strongly at the start (BOS). In all four names, in the scenes where the next letter is a (mi, av, ann, emm), 76–84% of the attention went to BOS.

2.5 Stage 3: MLP

A set of detectors that compare with "templates" The MLP (multi-layer perceptron) is the part that first spreads the four numbers on the blackboard out to sixteen and then brings them back to four.[21] Each of the sixteen rows of fc1 is one "template." If the arrangement on the blackboard is similar to the template, the detector responds (lights up); if not, it stays silent. fc2 holds, for each detector that lights up, the content "add this to the blackboard." The MLP looks only at its own blackboard. Information from other positions has already been copied by Attention. It has the same form as the trigger of a seismograph, which compares a waveform with a pattern and reports when a threshold is exceeded.

The responses of the sixteen detectors The sum of products (similarity) of each template and the blackboard is computed, sixteen in all (Figure 9). Those with negative similarity (h12, h14) are set to 0 and silenced. This "0 if negative" is called **ReLU**.[22] At p3, fourteen lit up. In the scenes where the next letter is a (mi, av, ann, emm), 14–15 light up at once. This is the common response for "a is coming soon."

[21] Lines 137–141. Before reading, the size is evened out here too (rmsnorm).

[22] ReLU stands for "rectified linear unit." It imitates the way a nerve cell "fires when the total stimulus exceeds a threshold"; here the threshold is 0 (Appendix A.4).

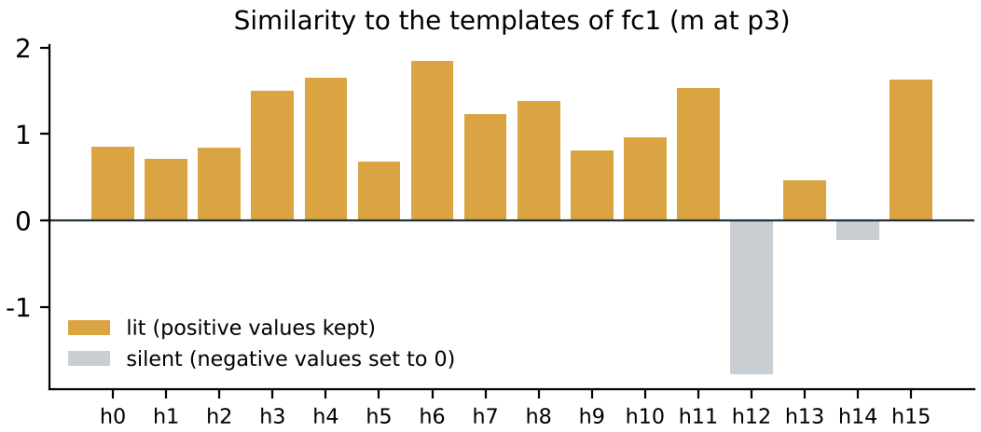


Figure 9 Similarity to the sixteen templates for the m at p3. Negative values (gray) are set to 0.

The detectors that lit up add to the blackboard Adding up the contents of fc2 according to the strength of lighting gives $(-4.74, +7.97, +0.76, -2.59)$. Adding this to the blackboard, the final blackboard is $(-7.12, +10.50, +0.11, -4.10)$. Comparing the sizes (arrow lengths) of what was written on the blackboard, the first note was 2.00, the result of Attention 2.49 and the result of the MLP 9.66.[23] The MLP added the most. About half of the knobs (128) are in the MLP, and it holds many of the rules of the form "in this situation, this comes next."

2.6 Stages 4 and 5: scores and probabilities

lm_head: reading the final blackboard and giving scores Each row of lm_head is the "reading template" of that letter. The sum of products of the final blackboard and the row is the score of that letter.[24] For example, the score of a is

$$(-0.38) \times (-7.12) + (+0.34) \times (+10.50) + (+0.05) \times (+0.11) + (-0.35) \times (-4.10) = +7.70.$$

[25]The scores of the seven letters are a +7.70, e +3.13, i −2.39, m +0.80, n −9.62, v −6.36, end +0.89; a is far higher than the rest (Figure 10, left).

Turning the scores into probabilities (softmax) The scores are turned into probabilities with the same softmax as in Attention. The difference between the scores of a and e is 4.57, and $e^{4.57}$ is about 96, so the answer is that a is about 96 times as likely as e. It gave the correct a 98.8% (Figure 10, right).

[23]The length of an arrow is the square root of the sum of the squares of the four numbers.

[24]lm_head stands for "language model head," the final output part (line 143). The technical term for a score is "logit."

[25]Computing with the numbers of the table rounded to two decimal places can shift the last digit slightly.

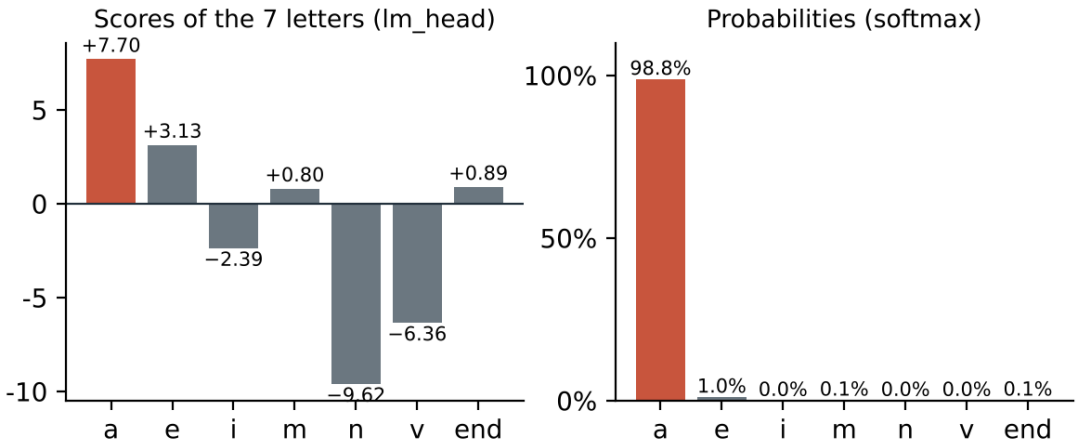


Figure 10 The seven scores (left) and probabilities (right) at p3 (after emm).

A score can be split into the sum of the three writings Writing on the blackboard is always addition, and lm_head only multiplies and adds, so the score also splits exactly into three parts (Figure 11).[26]

$$\text{score of a } +7.70 = \text{first note } +1.12 + \text{Attention } +1.13 + \text{MLP } +5.44$$

At p3, the first note and Attention push a up a little, and the MLP pushes it up a lot. The MLP pushes n and v down strongly.

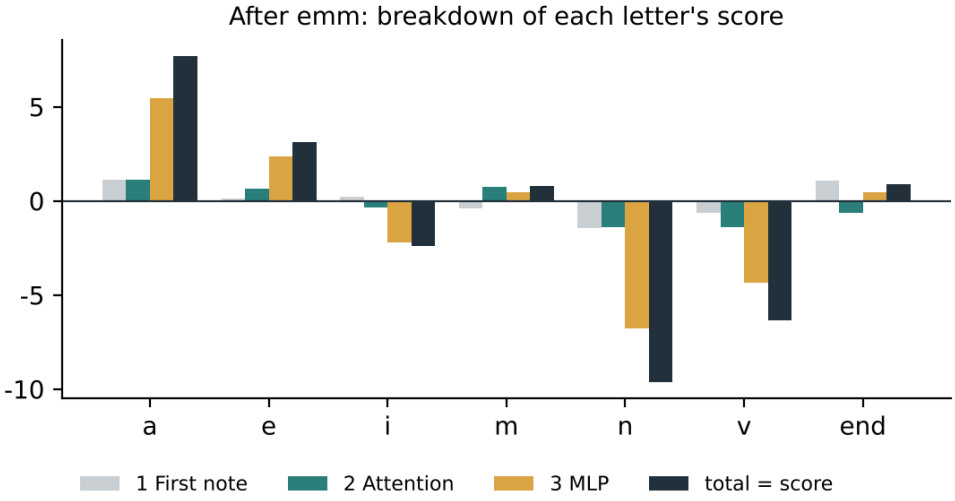


Figure 11 The breakdown of the score of each letter after emm.

At the end of the name, the MLP pushes "end" up At p4, after reading all of emma, the score of "end" is +12.26 = first note +1.57 + Attention +0.54 + MLP +10.15, and its probability was 99.6%. Nine detectors lit up, a different combination from p3: the group that responds to the scene "just finished reading a" is lit. Here too, the MLP is decisive.

[26]There is no size-evening computation (rmsnorm) before lm_head, so the sum of the three parts agrees with the score to the 14th decimal place (Appendix A.5).

2.7 The five problems of emma, and the fourteen scenes

The answers to the five problems of emma are as in Table 8. At p1–p4, it gave the correct answer 94–99.6%. At the start (p0), e gets only 19.3%, but this is not a failure. In the data, after the start comes a 50% and e and m 25% each, and the AI is unsure in nearly those shares (Chapter 1, "Correctly unsure").

Table 8 The answers to the five problems of emma

Position	Read so far	Correct	Prob. of correct	AI's prediction (top 3)
p0	start	e	19.3%	a 50.9%, m 29.7%, e 19.3%
p1	e	m	94.0%	m 94.0%, e 3.9%, a 1.3%
p2	em	m	94.4%	m 94.4%, e 3.7%, a 1.2%
p3	emm	a	98.8%	a 98.8%, e 1.0%, end 0.1%
p4	emma	end	99.6%	end 99.6%, a 0.4%

Looking at all fourteen scenes of the four names (Table 9), in the twelve scenes where the answer is unique, it gives 80–99.8%, and in the two scenes where it is unsure (start, a), it gives probabilities close to the shares in the data. The lowest was i after m (80.2%).

Table 9 Probability of the correct answer in the fourteen scenes (after training)

Read so far	Data	AI
start	a 50%, e 25%, m 25%	a 50.9%, e 19.3%, m 29.7%
m	i 100%	i 80.2%
mi	a 100%	a 99.2%
mia	end 100%	end 99.7%
a	v 50%, n 50%	v 49.9%, n 45.6%
av	a 100%	a 99.2%
ava	end 100%	end 99.8%
an	n 100%	n 84.4%
ann	a 100%	a 98.8%
anna	end 100%	end 99.8%
e	m 100%	m 94.0%
em	m 100%	m 94.4%
emm	a 100%	a 98.8%
emma	end 100%	end 99.6%

Summary of Chapter 2: each time it reads one letter, the AI computes in five stages: the first note (wte + wpe, size evened out) → Attention → MLP → lm_head scores → softmax probabilities. What told the two m's apart was the position table (wpe) and Attention. wpe put in "which letter in the name it is," and Attention copied in "which letter from the start it is." The score is the sum of three writings, and the one that pushed it up most was the MLP.

3 Measuring misses and training

In Chapter 2 we used the trained knobs. In this chapter, we put the knobs back to their random values before training and watch how 300 practice steps lead from there to the knobs of Chapter 2. The flow of one practice step was: predict (the computation of Chapter 2) → grade (loss) → find the cause (gradient) → turn the knobs (Adam). One practice step uses only one name, and the four names are repeated in the order mia, ava, anna, emma, mia, ….[27]

3.1 The AI before training

The 268 knobs all start as small random numbers, scattered roughly in the range −0.16 to +0.16.[28] No row of any letter or any position has a role yet. Like a club member who has just joined, it is eager but does not know anything yet.

Because the knobs are random, in every problem the AI hands out about 12–17% to each of the seven letters (Figure 12). Giving the same probability to seven letters makes each $1/7 \approx 14.3\%$. This is the same as a random guess, and practice starts from here.

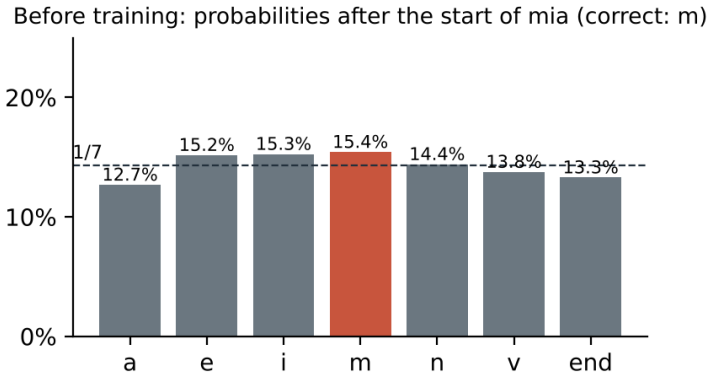


Figure 12 Probabilities of the letter after the start of mia, with the knobs before training. The dotted line is 1/7.

3.2 Measuring the miss (loss)

The AI does not decide on one next letter; it assigns a probability to each of the seven letters. So a penalty is decided from the probability given to the correct letter. This is called the **loss**. The penalty is computed as $-\ln$ (probability of the correct answer) (Figure 13).[29] Probability 1 (100%) gives penalty 0 (a perfect hit); 0.5 gives 0.69 (unsure between two); 1/7 gives 1.95 (a random guess among seven); 0.01 gives 4.61 (said "hardly possible" and missed).

[27] In the program, taking out a name is lines 156–157, prediction and loss are lines 163–169, the gradient is line 172 (`loss.backward()`), and updating the knobs is lines 174–182.

[28] Each is picked from a normal (Gaussian) distribution with mean 0 and standard deviation 0.08 (`random.gauss(0, std)` on line 80). Most values fall within twice the standard deviation (±0.16).

[29] ln is the natural logarithm, which tells "to what power e must be raised to give that number." The closer the probability is to 1, the closer the penalty is to 0; the closer the probability is to 0, the larger the penalty grows without limit.

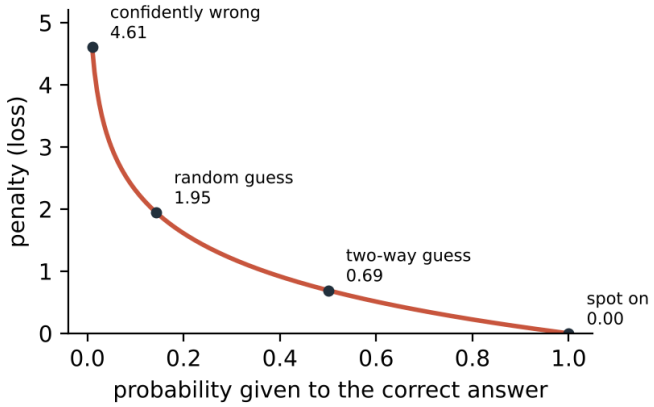


Figure 13 The probability given to the correct answer and the penalty.

> **The same as grading a weather forecast**: if you say "70% chance of rain" and it rains, the penalty is small. If you flatly say "1%" and it rains, the deduction is large. The more confidently you miss, the more sharply the penalty grows. It can also be called "the size of the surprise on learning the correct answer."

The penalty of the first practice step (mia) mia has four problems (Table 10). The average of their penalties is the loss of the first practice step. The average 1.953 is almost the same as the penalty of a random guess ($\ln 7 \approx 1.946$). The goal of training is to lower this average penalty.[30]

Table 10 The penalty of the first practice step (mia)

Problem	Read so far	Correct	Prob. of correct	Penalty	Prob. after step 1
1	start	m	15.4%	1.869	16.5%
2	m	i	11.6%	2.151	12.3%
3	mi	a	17.1%	1.764	18.4%
4	mia	end	13.1%	2.029	14.0%
Average				**1.953**	(penalty 1.889)

Why a penalty and not the "accuracy"? Accuracy (the share of problems where the most probable letter was correct) does not change when the probability is raised a little, from 0.15 to 0.16, unless the top letter changes. Accuracy moves only in jumps, like a staircase, so it cannot measure "how much better things got when a knob was turned a little." The penalty goes down a little even when the probability of the correct answer goes up only a little. Because it changes smoothly, the "direction in which the penalty goes down" can be computed. On a staircase, you cannot tell which way to step to go down; on a smooth slope, the tilt under your feet tells you the way down.

3.3 Which knob to turn, and which way (gradient)

When you want to go down into a valley in a foggy mountain, you cannot see far, but you can feel the tilt under your feet. Taking one step at a time in the downhill direction brings you closer to the valley (Figure 14). The 268 knobs are 268 directions. For each knob, the rate "how much

[30] The technical name for this kind of loss is "cross-entropy loss" (Appendix A.6).

the penalty increases when the knob is increased a little" is called the **gradient**. Decreasing the knobs with a positive gradient and increasing those with a negative gradient lowers the penalty. The gradient is "the tilt at that spot," so the tilt changes once you take a step. That is why we move a little and measure again.

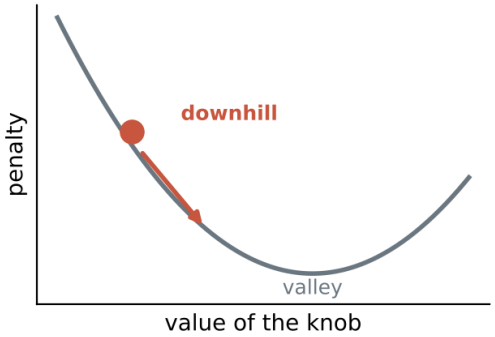


Figure 14 The gradient is the tilt of the slope under your feet.

Checking a gradient for real Let us increase the third knob of the row of i in lm_head (value before training 0.1090) by 0.001. The average penalty of mia increased from 1.953475 to 1.953936, by 0.000461. Change in penalty ÷ change in knob = 0.000461 ÷ 0.001 ≈ 0.461 is the gradient of this knob. The gradient computed automatically by the program is also 0.461; they agree. The gradient is positive, so decreasing this knob lowers the penalty.

However, checking the 268 knobs one by one in this way would require 268 extra prediction computations. Large AIs have hundreds of billions of knobs. So instead, the chain of computations is traced backward and all the gradients are found at once.

3.4 Finding the gradients at once (autograd)

A small example: $f = a \times b + a$ With $a = 2$ and $b = 3$, compute $c = a \times b = 6$ and $f = c + a = 8$ (Figure 15, left). Each time the program computes, it records "from which numbers, and by what computation, this was made." Next, starting from f, it traces the record backward (Figure 15, right). Addition passes the value on as it is; multiplication passes it on multiplied by "the other number." a receives 1 and 3 along two paths, so 4 in total. b gets $1 \times a = 2$.

Checking: increasing a by 0.01 makes f 8.04 (four times 0.01), and increasing b by 0.01 makes it 8.02 (twice 0.01). The 4 and 2 found by tracing backward express "how many times as much f increases when that number is increased a little." In microgpt, f is the penalty and a and b correspond to the 268 knobs.

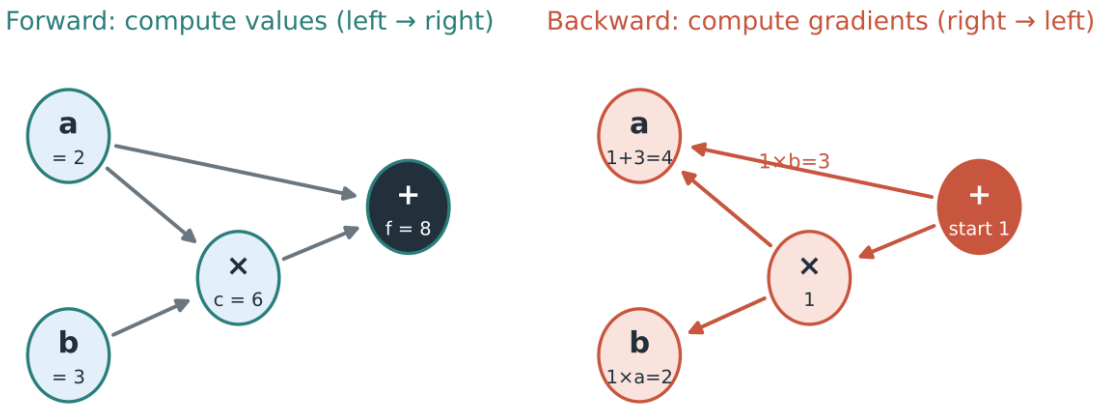


Figure 15 The record of computing $f = a \times b + a$. Left: computing the values. Right: tracing backward to compute the gradients.

The chain rule: multiplying gear ratios Turning a knob changes the numbers on the blackboard; a change in the blackboard changes the scores; a change in the scores changes the probabilities; a change in the probabilities changes the penalty. The effect is passed along in turn, like gears, and reaches the penalty. How many turns the penalty makes for one turn of the knob can be found by multiplying together the ratios of all neighboring gears. This is the **chain rule** of differentiation. The backward computation starts from the penalty and goes back one stage at a time, multiplying the ratios. The intermediate results can be reused, so all 268 gradients are obtained in one backward pass.[31]

The gradient of a score is "probability − correct" The first step of the backward pass takes a very simple form. The gradient of each letter's score is that letter's probability − (1 if correct, 0 if not) (Table 11).[32] The score of the correct m has a negative gradient, so it is turned in the direction of going up; the other six letters have positive gradients, so they are turned in the direction of going down. Since the average over four problems is taken, the result is divided by 4 at the end.

Table 11 The gradients of the seven scores for problem 1 of mia (start → correct m), before training

Letter	Probability	1 if correct	Prob. − correct	÷4 (problems)
a	12.7%	0	+0.127	+0.032
e	15.2%	0	+0.152	+0.038
i	15.3%	0	+0.153	+0.038
m	15.4%	1	−0.846	**−0.211**
n	14.4%	0	+0.144	+0.036
v	13.8%	0	+0.138	+0.034
end	13.3%	0	+0.133	+0.033

[31] This method is called backpropagation. In microgpt, a part called `Value` records, at each computation, "from what and what, by what computation, it was made," together with the gear ratio (the local derivative) (lines 29–72).

[32] The derivatives of the probability and of the penalty ($-\ln$) cancel out just right to give this form (Appendix A.7). The values agree with those computed by the program to the third decimal place.

The first gradients: which tables mattered most Comparing the size of the gradients table by table (Figure 16), the largest is lm_head (the table that gives the scores). It is the gear closest to the penalty. The gradients of the Attention tables (attn_wq and so on) were almost 0. Before training, the values of wv and wo are small, so whichever positions are looked at, the amount added is tiny; this is probably why changing the positions looked at hardly changes the penalty. Of the 268, 16 had a gradient of exactly 0: the rows of wte for e, n and v, letters that do not appear in mia (12), and the row of wpe for the unused fifth position p4 (4). Knobs that were not used have no effect on the penalty.

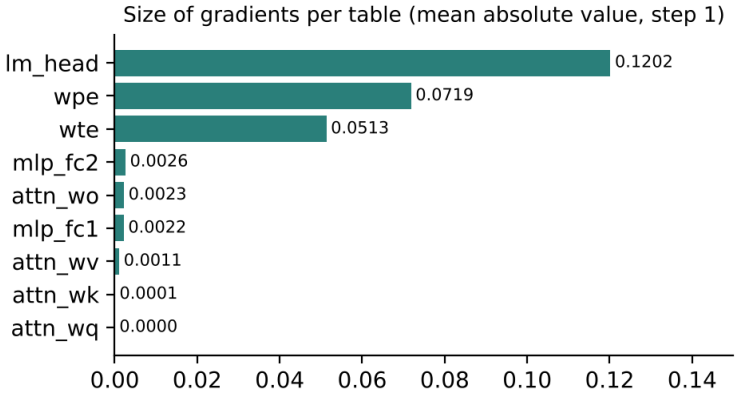


Figure 16 Size of the gradients by table (step 1).

3.5 Turning the knobs (Adam)

The basics: turn a little in the direction opposite to the gradient If the gradient is positive, decrease; if negative, increase. This is called **gradient descent**. Turning too far at once jumps over the valley. So we turn a little at a time and measure again. The guide to how far to turn at one time is called the **learning rate**. In microgpt, it starts at 0.01 and is made smaller little by little so that it becomes 0 at step 300. Long strides at first, short steps as the valley gets near.

Two ideas of Adam microgpt uses an improved version of gradient descent called Adam.[33]

1. **Using the average of the gradients (momentum)**: one practice step uses only one name, so the gradients vary from name to name. The average of recent gradients is used to smooth out this variation. Like the momentum of a ball rolling down a slope, it does not change direction at every small bump.
2. **Evening out the size of the turns**: the size of the gradient differs by several orders of magnitude from knob to knob. Each gradient is divided by its own size, so that every knob moves by about the same amount. It is like walking with the same stride whether the slope is steep or gentle.

Step 1: every knob moved by almost exactly 0.01 In Table 12, the knobs with positive gradients decreased and those with negative gradients increased, all in the direction that lowers the penalty. Whether the gradient was 0.46 or 0.28, the amount moved was almost the same,

[33]The name comes from "adaptive moment estimation"; it was published by Kingma and Ba in 2014. "Moment" is a term from statistics: the first moment is the average, the second moment the average of the squares. Adam estimates the average of the gradients and the average of the squared gradients to decide the direction and the size of a turn (Appendix A.8).

0.01. Because of Adam's "evening out the size," at step 1 every knob moves by the learning rate (0.01).[34] Knobs with a gradient of 0 (the 16, such as the wte of e) do not move.

Table 12 How the knobs moved at step 1 (the same as Section 5.3 of "Reading microgpt")

Knob	Before training	Gradient	After step 1	Amount moved
lm_head (row i, 3rd)	0.1090	+0.461	0.0990	−0.0100
lm_head (row a, 2nd)	0.0940	−0.438	0.1040	+0.0100
lm_head (row m, 4th)	−0.0270	−0.390	−0.0170	+0.0100
wte (row a, 1st)	0.0014	+0.282	−0.0086	−0.0100
wte (row e, 1st)	−0.1026	0	−0.1026	0

After step 1: mia got a little better. What about the other names? For all four problems of mia, the probability of the correct answer went up a little, and the penalty went down from 1.953 to 1.889 (Table 10, right). Checking the penalties of the other names too, ava, which like mia "ends in a," went down from 1.971 to 1.938, but emma went up from 1.963 to 1.979, and anna was almost unchanged, 2.001 to 2.002.

> An adjustment made for one name is not necessarily good for the other names. That is why the names are switched and the practice is repeated many times, searching for a place where the penalty goes down overall.

3.6 300 practice steps

How the penalty changed: jagged and smooth The gray line in Figure 17 is the penalty of the one name used at that step. Each name has a different difficulty, so it is jagged. The green line is the average of solving all four names (18 problems) with the knobs at that time. It went down smoothly from 1.97 to 0.347. It falls sharply up to about step 50, and then slowly approaches the line at 0.308.[35] In terms of "how many choices it is unsure among," it went from 7.2 choices to 1.41 choices.

[34] At step 1, both the average of the gradients and the average of their squares are made from that single gradient alone, so "gradient ÷ size of the gradient" = ±1. Knobs with very small gradients move about 0.0099, because of the small number added to avoid dividing by 0.

[35] 0.308 is the limit that comes from being "correctly unsure" after the start and after a (Chapter 1, Appendix A.1).

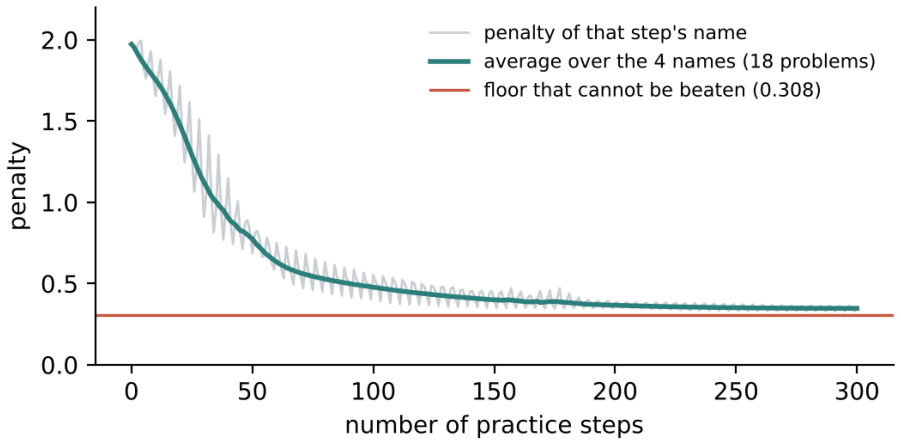


Figure 17 Number of practice steps and the penalty.

The AI learns the easiest things first Let us look at how the probability of the correct answer changed in the scenes where the answer is unique (Figure 18, left). The end of a name (mia → end) was learned early: 86.7% at step 60. m → i is slow to be learned: 21.0% at step 60, and still 80.2% at step 300. "After the final a of a name comes the end" is common to all four names, but "i after m" is a habit of mia alone. The more names a rule is common to, the more chances there are to practice it, and the earlier it is learned.

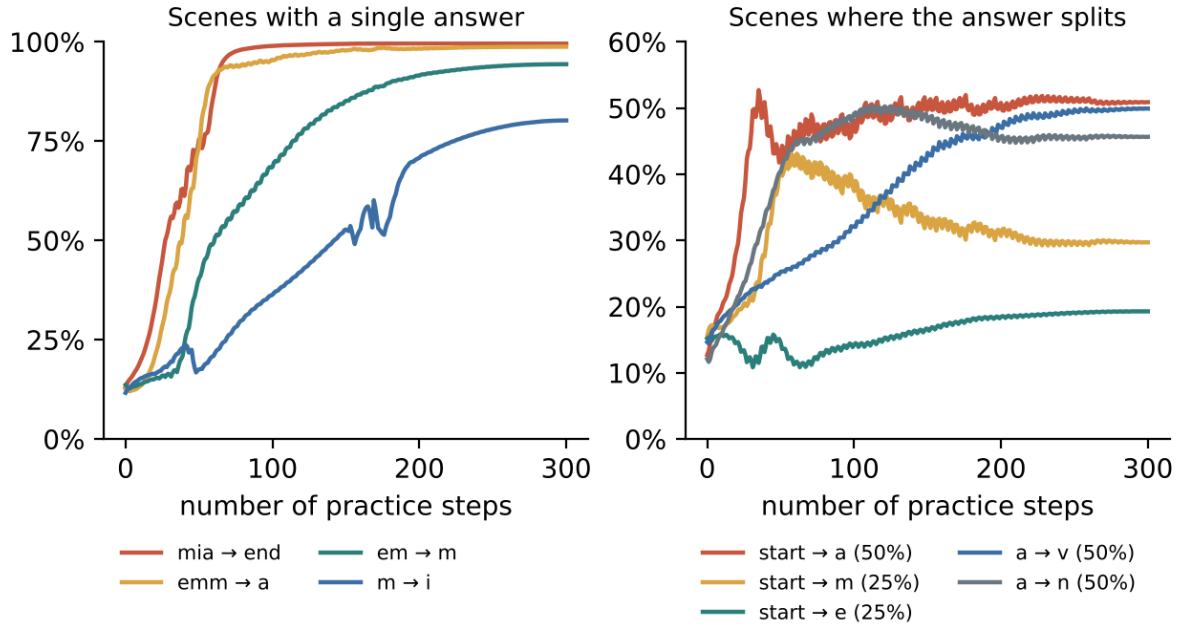


Figure 18 Changes in the probability of the correct answer. Left: scenes where the answer is unique. Right: scenes where the answer splits (shares in the data in parentheses).

The unsure scenes settle at the shares in the data In the scenes where the answer splits (Figure 18, right), m after the start once rose too high, to 41.3% at step 60, and then came back to around 30%. After a, n grew first and v caught up later; at step 300 it was nearly half and half, v 49.9% and n 45.6%. In both cases, they finally settle close to the shares in the data.

i and v came to point in the same direction during practice The alignment (cos) of the rows of i and v in wte was 0.64 before training (a chance direction). It became 0.91 at step 10, 1.00 at step 50, and stayed there (Figure 19). i (mia) and v (ava) have the same role: "the next one is always a." Placing them in the same direction lets them share the same rule. Nobody taught it to "make i and v similar." It happened naturally while the knobs were being turned in the direction that lowers the penalty.

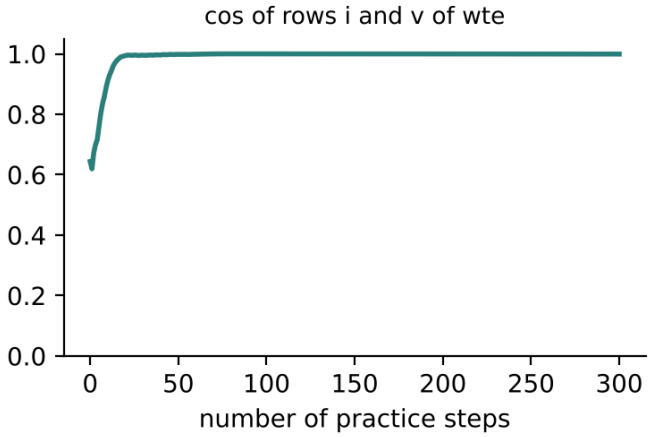


Figure 19 Change in the alignment of the rows of i and v in wte.

How far did the knobs move in 300 steps? Comparing the change from before to after training table by table (Figure 20), even the Attention tables, which had almost no gradient at first, moved a lot during the 300 steps. The lengths of the MLP templates (the sixteen rows of fc1) grew from 0.03–0.21 before training to 0.74–1.27. The smallest movement was in wte and wpe.[36]

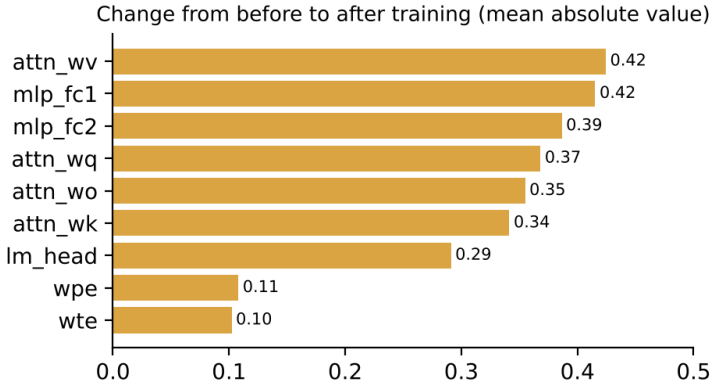


Figure 20 Change from before to after training, by table (average of absolute values).

3.7 Training is the same idea as the least squares method

In a science experiment where we hang weights on a spring, measure the extension and draw a straight line on the graph, we choose the slope and intercept of the line so that "the gaps between the data points and the line" are as small as possible (the least squares method). What microgpt does is essentially the same (Table 13).

[36]The values before training are about ±0.16, so an average change of 0.3–0.4 is much larger than the initial values.

Table 13 The least squares method and training microgpt

	Spring experiment (least squares)	microgpt
Knobs (parameters)	slope and intercept of the line (2)	numbers in the nine tables (268)
Miss (loss)	sum of the squared gaps between points and line	average penalty of the probability given to the correct answer
Training	find the slope and intercept that minimize the gaps	search for knob positions that make the penalty small
How to find it	can be solved at once with a formula	cannot be solved with a formula, so little by little, using the gradient

"Learning" for an AI means something different from learning for humans. It is adjusting the knobs so that the loss becomes small, nothing more and nothing less. Even so, an "organization" such as i and v pointing in the same direction emerges as a result.

Summary of Chapter 3: the miss is measured by the average of "$-\ln$ (probability of the correct answer)." Before training it was 1.953, the same as a random guess, and 300 practice steps brought it down to 0.347 (limit 0.308). The gradients are found all at once by tracing the record of the computation backward. The gradient of a score takes the easy form "probability − correct." Turning the knobs little by little with Adam, the AI learned the easiest things first, and i and v naturally came to point in the same direction.

4 Making names

When training is over, new names are made using the same prediction computation (Chapter 2). This is called **generation**. The knobs are no longer turned. The only new thing is how the probabilities are used to choose a letter.

4.1 The procedure for making a name

1. Have it read the start mark (BOS).
2. Compute the probabilities of the next letter (the computation of Chapter 2).
3. Draw lots whose chances of winning are the probabilities, and choose one letter.
4. If the chosen letter is the end (BOS), the name is complete. If not, append the chosen letter, have it read that letter, and go back to 2.

In practice, the AI read the correct name; here, it reads the letters it chose itself. The length it can read at once is 5, so when it reaches the fifth letter, it stops there even if the end has not come out.[37]

4.2 Turning probabilities into lots

After training, the AI gives the letters after the start a 50.9%, m 29.7% and e 19.3%. This is turned into a lottery like drawing one ticket from a box of 1000 tickets in which about 509 are winning tickets for a, about 297 for m and about 193 for e. It is certain that a comes out more easily, but m or e may also come out. The result changes with each draw, so the same AI makes a different name each time.

How the lots are drawn The computer picks one number at random from 0 up to (but not including) 1 (a random number). The bands of the probabilities are lined up from the left, and the letter of the section that contains the chosen number wins (Figure 21). The width of each section equals the probability, so the more probable a letter, the more easily it wins. It is the same as throwing darts with your eyes closed: the wider sections are hit more often.[38]

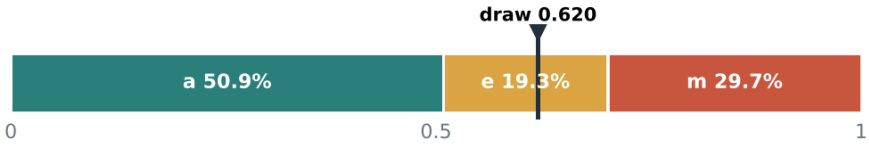


Figure 21 The band of probabilities and the lot. 0.62 falls in the section of m, so m is chosen.

Why not choose "the most probable letter"? If the top letter is always chosen (greedy decoding), the same AI always produces only the same name. For this AI, it goes start → a (50.9%), a → v (49.9%), …, and every time it is just ava. After a, v and n have almost the same probability, yet v is always chosen. Conversely, if the probabilities are ignored and letters are chosen equally, the result is a random string of letters like before training. Drawing lots with the probabilities as the chances of winning makes a variety of names while keeping them name-like.

[37] In the program, this is lines 186–200. Line 195 computes the probabilities, and `random.choices` on line 196 draws the lots.

[38] The random numbers are computed from the random seed 42 seen in Chapter 1, so with the same program, whoever runs it gets the same sequence of names.

4.3 Temperature

Before turning them into probabilities (before the softmax of Chapter 2), the seven scores are divided by a number called the **temperature** (Figure 22). When the temperature is smaller than 1, the differences between scores widen, and the more probable letters become even more likely to be chosen. At temperature 0.5, a after the start increases from 50.9% to 67.4%, and e decreases from 19.3% to 9.7%. When the temperature is larger than 1, the differences shrink, and rare letters also become more likely to be chosen. microgpt uses temperature 0.5.[39]

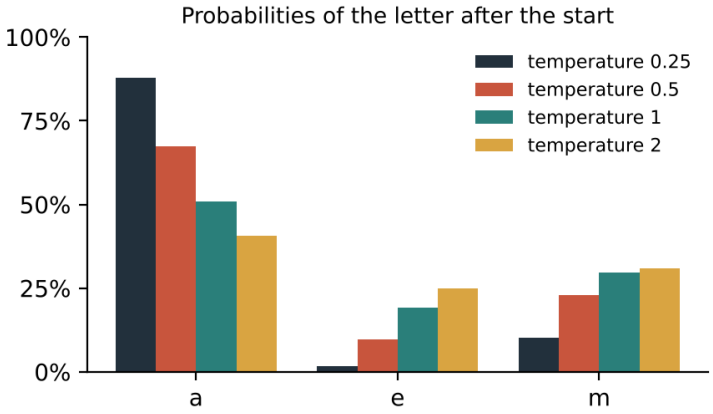


Figure 22 Differences in the probabilities of the letter after the start, by temperature.

The temperature is a knob for "caution" and "adventurousness." At a low temperature it chooses only the probable letters, and safe, familiar names become more common. As it approaches 0, it becomes the same as greedy decoding, which always chooses the top letter. At a high temperature it often chooses improbable letters too, and unusual or broken names increase. Making it large brings it close to random.

> The temperature is not used in practice (training). It is a setting chosen by the user when making names. Without retraining the knobs, names can be made again just by changing the temperature. The same holds for large AIs such as ChatGPT and Claude: in developer use, the temperature can be specified. The name "temperature" comes from physics. The formula that divides the scores by the temperature to get probabilities has the same form as the Boltzmann distribution of statistical mechanics (Appendix A.9).

4.4 How one name is made

Let us follow, one lot at a time, the first name the trained AI made, "ava" (Figure 23). The bands are the probabilities adjusted with temperature 0.5. After the start, the lot 0.555 fell in the section of a (0–0.674), and a was chosen. After a, the lot 0.932 fell in the section of v, and v was chosen. After av and after ava, only one letter has a probability of almost 100%, so whatever number is drawn, the same letter (a, end) results.

[39] At temperature 1, the probabilities are exactly those computed by the AI. In the program, line 187 sets `temperature = 0.5`, and line 195 divides the scores by the temperature.

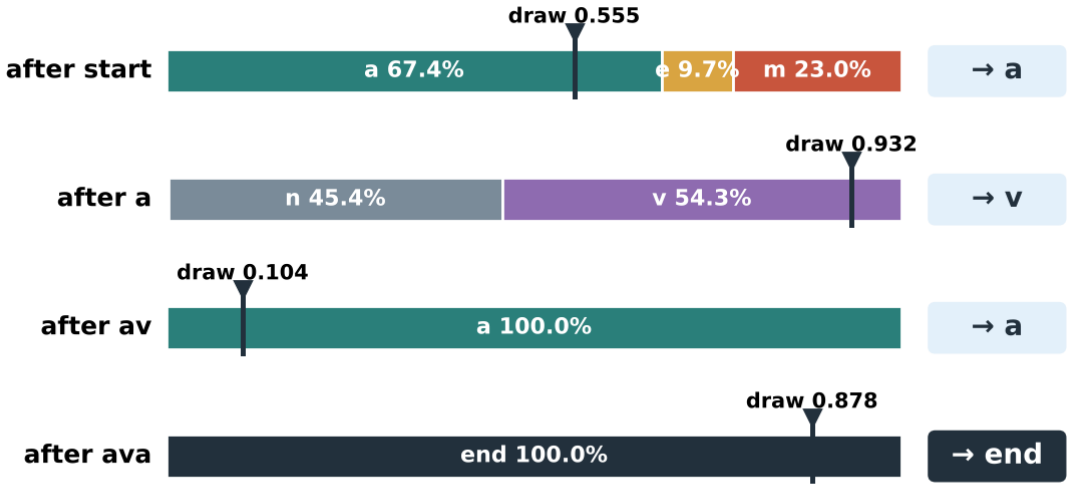


Figure 23 How the first name, "ava," is made (temperature 0.5). The black line is the number chosen by the lot.

The chosen letter becomes the next letter read In practice, the whole correct name was known from the beginning, so the predictions at all positions could be computed together. When making names, the next-but-one cannot be computed until the next letter is decided by lot. So it can only proceed one letter at a time. This is also why ChatGPT and Claude display their answers little by little. The headings (k) and contents (v) of the earlier positions that Attention looks at are kept in the order they were made and reused.[40]

4.5 The names it made

When the trained AI was made to produce twenty names, the result was as follows (temperature 0.5).[41]

ava, ava, anna, anna, mia, mia, anna, anna, ava, ava, ava, mia, anna, ava, ava, anna, mia, anna, emma, ava

There were eight ava, seven anna, four mia and one emma, and not a single name other than the four. With so little data, the AI has memorized the names whole. The four names it learned each appeared exactly once, yet ava and anna are common and emma appears only once. Why?

Why emma is rare: temperature 0.5 reduces "e" even further The probability of a name is the product of the probabilities of its letters. For emma to be chosen, e must first come out after the start. e after the start is 19.3% at temperature 1, and only 9.7% at temperature 0.5. a increases from 50.9% to 67.4%. Temperature 0.5 makes "the majority even larger and the minority even smaller." That is why ava and anna, which start with a, increased, and emma decreased (Table 14).

[40] The mechanism of keeping the k and v of earlier positions is the same as what large AIs call the "KV cache" (lines 121–122).

[41] This is exactly the result of running the original `microgpt.py` with only its four numbers and `input.txt` changed (the same as Section 5 of "Reading microgpt").

Table 14 The probability that each name is made (after training)

Name	Data	Prob. at temp. 1	Prob. at temp. 0.5	Count out of 20
ava	25%	25.1%	**36.6%**	8
anna	25%	19.3%	**30.0%**	7
mia	25%	23.6%	**22.6%**	4
emma	25%	16.8%	**9.6%**	1
Other names	0%	15.1%	1.2%	0

Making names at different temperatures Making twenty names each while changing only the temperature gave the following.[42] Underlined names are not in the data.

- **Temperature 0.5** (0 names not in the data): the same as above.
- **Temperature 1** (2): emma, mia, anna, ava, mia, mia, emma, anna, ava, ava, emma, anna, ava, emma, anna, ava, emma, ava, aia, emea
- **Temperature 2** (12): emeea, mnia, ava, ava, meia, anna, mnia, ema, m, anna, emna, anva, ava, iana, aia, mia, ava, ava, mvaa, av

Making 1000 names each and counting them (Table 15, Figure 24), at temperature 0.25 only the four names come out, and emma becomes very rare. At temperature 1 the four are about equally common, and names not in the data make up about 15%. At temperature 2, more than half are names not in the data (aia, mva, anva, emea, a and so on).

Table 15 Counts when making 1000 names each (random seed 7)

Temperature	ava	anna	mia	emma	Names not in the data
0.25	501	368	113	18	0
0.5	350	335	216	88	11
1	261	193	235	163	148
2	136	64	117	80	603

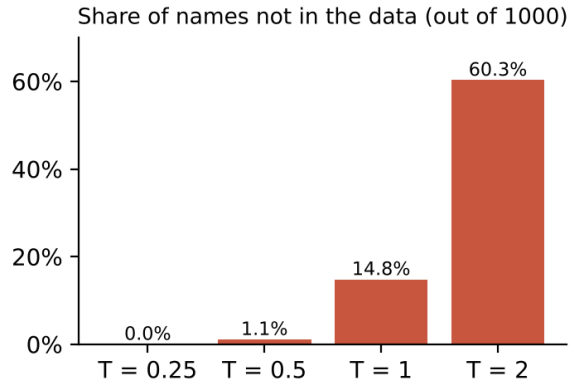


Figure 24 Share of names not in the data, out of 1000.

[42] All three use the continuation of the same random numbers right after training, with only the temperature changed. The temperature-0.5 list is the same as the result of the original program.

4.6 Names not in the data are combinations of learned parts

How were the names not in the data made? Lining up the probabilities (temperature 1) of the letters chosen at each position gives the following.

Name	Probability of each letter
ava	start → a 50.9%, a → v 49.9%, av → a 99.2%, ava → end 99.8%
aia	start → a 50.9%, a → **i 3.5%**, ai → a 99.2%, aia → end 99.8%
anva	start → a 50.9%, a → n 45.6%, an → **v 10.8%**, anv → a 98.8%, anva → end 99.8%

aia is a name where a was chosen after the start, and then the rare i (3.5%) was drawn. After that, the rules learned from mia, "after i comes a" and "after the final a of a name comes the end," were used as they are. anva too, after drawing the rare v (10.8%) after an, continues with "after v comes a" (learned from ava). New names are not random; they are learned parts (rules) joined in new ways.

4.7 Names made before and during practice

The names made with the knobs before training are random strings of letters (temperature 0.5, random seed 7).

ianam, iamai, aaiva, en, mi, aviea, ivenn, imaae, nminm, ivnim, m, ne, eivem, anvnv, innnm, v, mnann, (empty), veina, meaav

Because it gives almost the same probability to the seven letters, "end" is chosen only about as often as the other letters, so some names end after one letter (m, v) and some end without making anything (empty). The five-letter ones are names stopped at the length it can read at once (5).

Making ten names each during practice, the names changed as in Table 16.[43] Because it learns the easiest rules first (Chapter 3), the names it made changed from "all a's" → "name-like endings" → "the four names."

Table 16 Names made during practice (ten each)

Practice	Names made	What we saw in Chapter 3
Before training	ianam, iamai, aaiva, en, mi, aviea, ivenn, imaae, nminm, ivnim	almost the same probability for the seven letters
Step 25	aeea, aeaa, aea, aia, an, avaaa, ava, eeaaa, annnv, aim	first learns that "a is common"
Step 50	aiaa, aiaa, aia, ann, mnva, aia, mia, ania, mnn, anva	learning the rule for endings
Step 100	anna, aia, anna, anna, ava, anna, anna, aia, anna, mnna	has also learned emm → a, em → m
Step 300	anna, anna, anna, ava, ava, anna, ava, anna, anna, mia	has learned m → i up to 80%

4.8 With more data

The twenty names made with the original settings (32,000 names, 1000 practice steps, temperature 0.5) are as follows (Section 7.2 of "Reading microgpt").

[43] Made at temperature 0.5, each time with random seed 7. This does not affect the random numbers of training (the same method as Appendix D of "Reading microgpt").

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

Some are real names, such as ann, anna, lara and anton, but many are name-like ones not in the data, such as kamon and vialan. 32,000 names cannot all be memorized, so the AI learns the statistics of "how letters are often arranged in names." With four names, memorization; with 32,000, the statistics of parts. Even with the same program, the AI's "way of learning" changes with the amount of data.

"New names" and the made-up stories of large AIs When microgpt prints the names it made, it writes "new, hallucinated names." "Hallucination" is also the word used when a large AI produces something untrue that sounds plausible. The mechanism is the same. A large AI, too, is only choosing the next token by lot according to the probabilities of the word sequences it has learned. If a sequence is plausible as a sequence, it is produced regardless of whether it is true.[44] kamon is "name-like," but it is not necessarily a real name. With the answers of large AIs too, being "plausible as text" and being "correct" are different things. The habit of checking is important.

Summary of Chapter 4: names are made by drawing probability lots one at a time. Starting from the start, it computes the probabilities of the next letter, draws a lot, reads the chosen letter, and when the end comes out, the name is complete. The temperature adjusts the chances of the lots. Temperature 0.5 makes the majority even larger, so ava and anna, which start with a, were common and emma was rare. New names are combinations of learned parts. With four names it is close to memorization; with more data, new names are made from the statistics of "name-likeness."

[44] In large AIs, efforts are made to reduce made-up stories, for example by additional training to choose answers that match the facts, or by having them answer based on materials they have looked up (see "LLM Basics (3): Toward Dialogue Models").

5 Comparing with real AI

At the top of the microgpt program, it was written, in effect, that "this file is the complete algorithm; everything else is just efficiency." In this chapter, we check this claim.

5.1 Comparing sizes

Table 17 lines up the small example, the original microgpt, GPT-2 (the smallest one) and GPT-3.[45] GPT-2 (2019) and GPT-3 (2020) are language models announced by OpenAI, early members of the GPT series on which ChatGPT is based. The microgpt model has almost the same structure as GPT-2. The difference is mainly in the size of the numbers in this table.

Table 17 The sizes of four models

	Small ex.	microgpt	GPT-2 (smallest)	GPT-3
Blackboard width (numbers per token)	4	16	768	12,288
Layers (Attention + MLP pairs)	1	1	12	96
Attention heads	1	4	12	96
Length it can read at once	5	16	1,024	2,048
Vocabulary (kinds of tokens)	7	27	50,257	50,257
Number of knobs (approx.)	**268**	**4,192**	**124 million**	**175 billion**

Lining up the knobs at "1 knob = 1 mm," the small example (268) is 26.8 cm, about the length of a ruler; microgpt (4,192) is about 4.2 m, about the width of a classroom; the smallest GPT-2 (about 124 million) is about 124 km, about three full marathons; and GPT-3 (about 175 billion) is about 175,000 km, a length that goes around the earth about 4.4 times.[46] GPT-3 has about 40 million times as many knobs as microgpt, and about 650 million times as many as the small example (Figure 25, left).

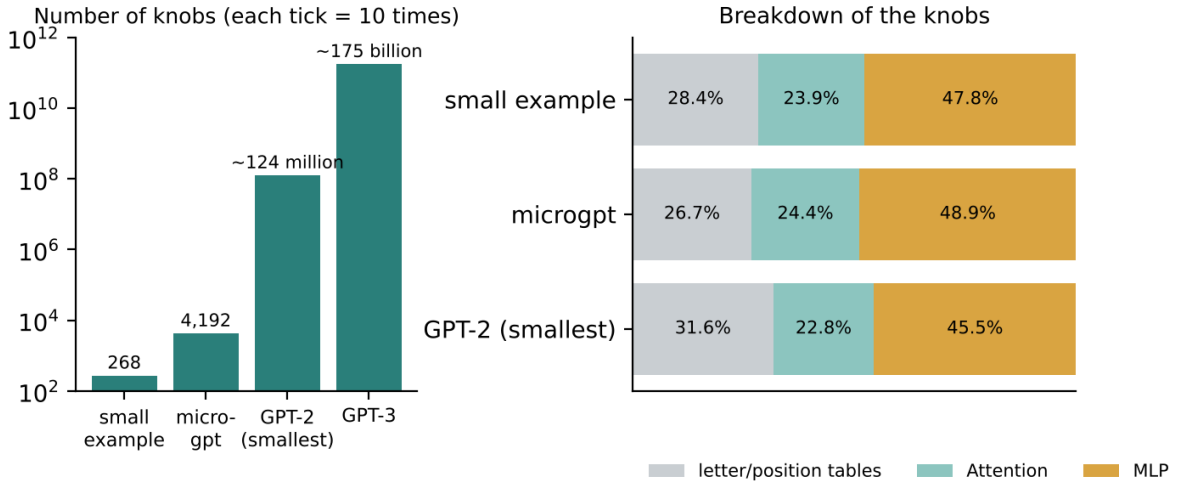


Figure 25 Left: number of knobs (each step on the vertical axis is a factor of 10). Right: breakdown of the knobs.

[45] The numbers for GPT-2 and GPT-3 are the same as in the table of Section 8.1 of "Reading microgpt." For many recent large AIs such as ChatGPT and Claude, the structure and size have not been published in detail, so here we compare with GPT-2 and GPT-3, which have been published.

[46] Computed with the earth's circumference as about 40,000 km and a full marathon as 42.195 km.

The distribution of knobs is much the same, whatever the size Dividing the knobs into "tables of letters and positions (wte, wpe, lm_head)," "Attention" and "MLP" (Figure 25, right), in every model the MLP is about half (48%, 49%, 46%) and Attention about a quarter.[47] As we saw in Chapter 2, the largest number of knobs goes to the MLP, which holds many of the rules of the form "in this situation, this comes next." Even though the sizes differ by a factor of about 460,000, the idea behind the distribution is the same.

Stacking layers microgpt passes through the pair of Attention and MLP only once (one layer). The smallest GPT-2 passes through it 12 times, GPT-3 96 times. The blackboard stays a single one, and the Attention–MLP pairs add to it again and again. The next layer can read what the previous layer added, so it can "combine what was found and find even more." Like an assembly line in a factory, each process does only a simple job, but stacking processes makes a complex product.

Heads and the length it can read at once The number of heads is how many groups Attention is split into. The width of the blackboard is divided, and each group goes to look at different positions. Several heads are like sharing the reading of the same sentence among, say, a "grammar person" and a "meaning person" (although the actual division of labor is decided by training and is not necessarily clear to humans). The length it can read at once is the upper limit on the number of positions Attention can look at. Every position looks at all the positions before it, so doubling the length makes the number of pairs looked at about four times as large (Table 18, Appendix A.11).

Table 18 The length it can read at once and the number of pairs Attention looks at

Length read at once n	Number of pairs looked at $n(n+1)/2$
5 (small example)	15
16 (microgpt)	136
1,024 (GPT-2)	524,800
2,048 (GPT-3)	2,098,176

Vocabulary In microgpt, one token is one letter, but GPT-2 and GPT-3 group combinations of letters that appear often in the data (words or parts of words) into one token.[48] A frequent word is one token; a rare word is split into several pieces. A larger vocabulary makes the wte and lm_head tables larger: in GPT-2 it is 50,257 rows × 768 columns, and about 30% of the knobs are in this table. The point that all the AI knows is numbers is the same.

[47] The numbers for GPT-2 were counted from its published structure (blackboard width 768, 12 layers, vocabulary 50,257, length 1,024), for a total of 124,439,808 (Appendix A.10). The Attention and MLP numbers for GPT-2 include the constant terms (biases). The normalization knobs (0.03%) are omitted from the figure. GPT-2 shares lm_head with wte.

[48] One method of grouping frequent combinations of letters is BPE (byte pair encoding). For details, see "LLM Basics (1): Tokens and Language Models."

5.2 The amount of data and practice

Table 19 The amount of data used for practice

	Small example	microgpt	GPT-3
Data	4 names	32,033 names	a huge amount of text
Practice problems (tokens)	18	228,146	about 300 billion
Practice steps	300	1,000	–
Data used in one step	1 name	1 name	many documents together

The data of the original microgpt are 32,033 names, 196,113 letters, which make 228,146 practice problems (Table 19).[49] Yet the original microgpt uses only 1000 names in its 1000 practice steps, about 3% of all the data. Even so, it became able to make new, name-like names (Chapter 4). A large AI reads many documents together in one practice step and turns the knobs with the average gradient.

More data changes the way of learning With four names it was almost memorization. Names not in the data came out only when the temperature was raised, by joining parts in new ways. When there are too many names to memorize, the AI has no choice but to learn the statistics of "how letters are often arranged in names." For large AIs too, it is known that the more knobs and the more data, the lower the penalty (loss) on text seen for the first time tends to be (scaling laws[50]).

5.3 The way of computing

microgpt puts each number into a part called `Value` and computes while recording each addition and multiplication one at a time. It runs with only the standard features of Python, and the whole computation is visible, but in exchange it is slow: about 2 seconds for the small example and 2–3 minutes with the original settings. Large AIs multiply tables of numbers (matrices) all at once. They use many GPUs, devices that can carry out a huge number of computations of the same kind at the same time.[51] The formulas are the same; the way of computing is fast. microgpt is like one person computing one step at a time with a calculator; a large AI is like a gymnasium full of people doing computations of the same form all at once. The answer is the same, but the time it takes is completely different.

For example, "multiplying a table by the blackboard" in microgpt repeats a sum of products as many times as there are rows, but with matrices it can be written all at once as $\boldsymbol{y} = W\boldsymbol{x}$. The Attention scores too: instead of finding the sum of products of q and k for each earlier position, the q's and k's of all positions are put together into matrices Q and K, and $QK^{\mathsf{T}}/\sqrt{d}$ is computed at once. In practice, the whole answer text is known from the beginning, so the predictions at all positions can be computed together at once. Large AIs, too, can only proceed one letter (one token) at a time when making names or text.

[49] This is 196,113 letters plus one "end" for each name. The GPT-3 number is from the published paper (Brown et al., 2020).

[50] See "LLM Basics (2): Pretraining and Scaling Laws."

[51] In developing large AIs, libraries (such as PyTorch) are used to carry out matrix computations quickly on GPUs. microgpt does not use them and writes everything itself.

5.4 Small differences in structure

A comment on line 93 of the program lists only three differences from GPT-2 (Table 20). All are changes made "to simplify the computation." Of these, rmsnorm and no biases are choices also used in recent large AIs (such as LLaMA). The assembly of the parts, blackboard (residual stream), Attention, MLP, lm_head and softmax, is the same as GPT-2.

Table 20 Differences from GPT-2

	GPT-2	microgpt	Difference
Size-evening computation	layernorm	rmsnorm	layernorm subtracts the average before evening out; rmsnorm does not subtract the average (simpler)
Constant terms (biases)	yes	no	a fixed number added after multiplication; microgpt omits it
Bend in the MLP	GELU	ReLU	GELU bends smoothly near 0; ReLU bends sharply at 0

Adjustment after training: making an AI that can hold a dialogue All microgpt does is the practice of predicting the "next token" from a large amount of data (pretraining). Dialogue AIs such as ChatGPT and Claude, after pretraining, receive additional training with examples of questions and answers so that they follow instructions (instruction tuning), and learn preferred ways of answering from human ratings of good and bad answers (RLHF and others).[52] At every stage, the principle of turning the knobs in the direction that lowers the penalty is the same. What differs is what counts as a "good answer" (how the loss is defined).

5.5 What stays the same

The skeleton is the same (Figure 26). Prediction turns tokens into numbers, looks up the tables to make the first note, repeats Attention and MLP for the number of layers, gives scores, and turns them into probabilities with softmax. Training measures the penalty ($-\ln$ of the probability of the correct answer), finds the gradients by tracing backward, and turns the knobs. Generation repeats drawing lots with probabilities adjusted by temperature, appending the chosen token, and reading it.

[52] RLHF stands for "reinforcement learning from human feedback." For details, see "LLM Basics (3): Toward Dialogue Models."

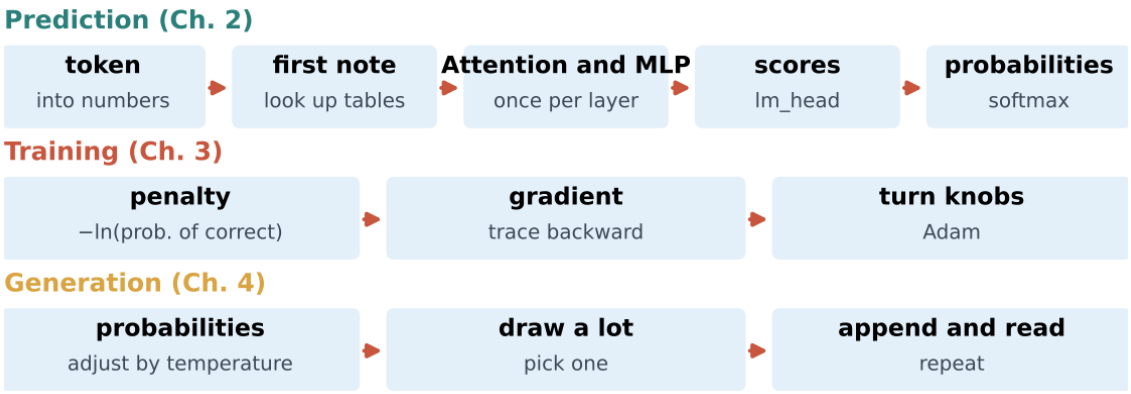


Figure 26 The skeleton common to microgpt and large AIs.

How do the things we saw in Chapters 1–4 look in large AIs (Table 21)?[53]

Table 21 What we saw in microgpt, and large AIs

What we saw in microgpt	In large AIs
All the AI knows is numbers (Ch. 1)	The same: it handles only token numbers
Correctly unsure; the penalty does not become 0 (Ch. 1)	The same: the continuation of a text is often not unique
Letters with the same role are placed in the same direction (i and v)	Words used in similar ways are known to point in nearby directions
Adding to the blackboard (Ch. 2)	The same mechanism; the blackboard is wide, 768–12,288
Learns the easiest things first (Ch. 3)	More frequent patterns tend to be learned earlier
Temperature and lots (Ch. 4)	The same: the user can choose the temperature
Plausible new names (Ch. 4)	Plausible made-up stories (hallucinations)

What differs: reading the inside is much harder For the 268 knobs of the small example, "A Look Inside a Small AI's Head" was able to work out the roles of almost all of them (i and v, the sixteen detectors and so on). Even so, the value of each single cell had no meaning, and all we found was that it remembers things as "a collection of rules." With 175 billion knobs, it becomes very hard for humans to work out what is going on inside. Research into this (interpretability research) is very active today. One study showed that large AIs solve arithmetic problems not with a clean procedure but with "a collection of many small rules."[54] This closely resembles the picture we saw in the small AI with 268 knobs.

Summary of Chapter 5: the sizes differ by orders of magnitude; GPT-3 has about 175 billion knobs (about 650 million times the small example). What differs is mainly the way of computing (putting things together into matrices and computing all at once on GPUs) and three small points of structure (rmsnorm, no biases, ReLU). The skeleton (numbers → note

[53] The right column briefly summarizes tendencies generally known about large AIs. They do not necessarily hold in every case.

[54] Nikankin et al., "Arithmetic Without Algorithms: Language Models Solve Math With a Bag of Heuristics."

→ Attention and MLP → probabilities) and the principle of training (lowering the penalty using gradients) are the same.
The answer to the first question: microgpt's claim that "this file is the complete algorithm; everything else is just efficiency" was true as far as the skeleton of the computation goes. However, the additional training for dialogue, and the abilities that arise from size, are not in the 200 lines.

Closing: what dissecting microgpt showed us

1. **Overview and preparation**: the AI repeats "predict the next one." The names become sequences of numbers and 18 practice problems. In scenes where the answer splits, being correctly unsure is the best answer.
2. **Reading one letter**: first note → Attention → MLP → scores → probabilities. The score is the sum of three writings, and the one that matters most is the MLP.
3. **Measuring misses and training**: the penalty is measured with $-\ln$, the gradients are found by tracing backward, and the knobs are turned with Adam. It learned the easiest things first, and i and v naturally came to point in the same direction.
4. **Making names**: probability lots are drawn one at a time. The temperature changes the lineup. New names are combinations of learned parts.
5. **Comparing with real AI**: size and the way of computing differ, but the skeleton and the principle of training are the same.

For further reading, there are "Reading microgpt," which reads the 200 lines of microgpt in order, including the original settings (32,000 names); "A Look Inside a Small AI's Head," which works out the meaning of the 268 trained knobs with the same four names; "LLM Basics (1)–(3)," on tokens and language models, pretraining and scaling laws, and dialogue models; and "Deep Learning (1)–(4)," from neural networks to the Transformer.

Appendix A Formulas

Here the computations of the main text are written as formulas, at about the level of first- or second-year university science students. The symbols are as follows: vocabulary size $V = 7$, blackboard width $d = 4$ (`n_embd`), length read at once $B = 5$ (`block_size`). The tables are E (wte, $V \times d$), P (wpe, $B \times d$), W_Q, W_K, W_V, W_O ($d \times d$), W_1 (fc1, $4d \times d$), W_2 (fc2, $d \times 4d$) and U (lm_head, $V \times d$). One name is written as the sequence of numbers $x_0, x_1, \ldots, x_n$ ($x_0 = x_n = 6$, BOS).

A.1 Scenes where the answer splits, and the lower limit of the loss (Ch. 1)

What the AI computes is the probability of the next letter conditioned on what has been read, $x_0 \ldots x_{t-1}$: $P(x_t \mid x_0, \ldots, x_{t-1})$. By the multiplication rule of probability, the probability of a whole name is

$$P(\text{name}) = P(x_1 \mid x_0)\, P(x_2 \mid x_0, x_1) \cdots P(x_n \mid x_0, \ldots, x_{n-1}). \tag{1}$$

Computed with the shares in the data, $P(\text{emma}) = \frac{1}{4} \times 1 \times 1 \times 1 \times 1 = \frac{1}{4}$ and $P(\text{ava}) = \frac{2}{4} \times \frac{1}{2} \times 1 \times 1 = \frac{1}{4}$; all four are $\frac{1}{4}$.

When answering exactly with the shares in the data, the total penalty of the 18 problems is: in the four problems at the start, $-\ln \frac{2}{4}$ twice and $-\ln \frac{1}{4}$ twice; in the two problems after a, $-\ln \frac{1}{2}$ twice; and in the other twelve problems, $-\ln 1 = 0$. So

$$2(0.693) + 2(1.386) + 2(0.693) = 5.545, \qquad 5.545 \div 18 = 0.308. \tag{2}$$

This is the ambiguity that remains in the data (conditional entropy); whatever the prediction, the average penalty cannot be smaller than this (Gibbs' inequality). $e^{\text{average penalty}}$ is called the perplexity and expresses "how many choices it is unsure among on average." The limit is $e^{0.308} = 1.36$ choices; before training it is $e^{1.973} = 7.2$ choices, and after training $e^{0.347} = 1.41$ choices.

A.2 The first note, rmsnorm, cos (Ch. 2)

$$\text{rms}(\boldsymbol{z}) = \frac{\boldsymbol{z}}{\sqrt{\frac{1}{d}\sum_i z_i^2 + \varepsilon}}, \quad \varepsilon = 10^{-5}, \qquad \boldsymbol{x}_t^{(0)} = \text{rms}\big(E_{x_t} + P_t\big) \tag{3}$$

Ignoring ε, the length of $\text{rms}(\boldsymbol{z})$ is $\sqrt{d} = 2$. microgpt also applies rmsnorm just before Attention and the MLP, but the first note already has length 2, so the rmsnorm just before Attention hardly changes it. The alignment of the directions of two vectors is

$$\cos\theta = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{|\boldsymbol{a}|\,|\boldsymbol{b}|}, \qquad \boldsymbol{a} \cdot \boldsymbol{b} = \sum_i a_i b_i. \tag{4}$$

The e at p1 and the m at p2 had $\cos\theta = 0.98$; the m at p2 and the m at p3 had -0.41.

A.3 Attention and softmax (Ch. 2)

With $\hat{\boldsymbol{x}}_t = \text{rms}(\boldsymbol{x}_t^{(0)})$,

$$\boldsymbol{q} = W_Q \hat{\boldsymbol{x}}_t, \quad \boldsymbol{k}_s = W_K \hat{\boldsymbol{x}}_s, \quad \boldsymbol{v}_s = W_V \hat{\boldsymbol{x}}_s, \qquad s_s = \frac{\boldsymbol{q} \cdot \boldsymbol{k}_s}{\sqrt{d}}, \qquad a_s = \frac{e^{s_s}}{\sum_{s'=0}^{t} e^{s_{s'}}} \tag{5}$$

$$\boldsymbol{x}_t^{(1)} = \boldsymbol{x}_t^{(0)} + W_O \sum_{s=0}^{t} a_s \boldsymbol{v}_s \tag{6}$$

Since $s \le t$, positions ahead cannot be looked at (causal Attention). In the p3 example, the scores are $(+2.69, -0.11, +0.00, +1.00)$ and the shares $(0.762, 0.046, 0.052, 0.140)$. The original microgpt has `n_head` $= 4$: the 16 numbers are split into groups of four, and four sets of Attention are carried out separately (multi-head).

A.4 MLP (Ch. 2)

$$\boldsymbol{u} = W_1 \,\mathrm{rms}(\boldsymbol{x}_t^{(1)}), \qquad h_i = \mathrm{ReLU}(u_i) = \max(0, u_i), \qquad \boldsymbol{x}_t^{(2)} = \boldsymbol{x}_t^{(1)} + W_2 \boldsymbol{h} \tag{7}$$

u_i is the inner product of row i of fc1 (template $\boldsymbol{\kappa}_i$) and the size-evened blackboard $\hat{\boldsymbol{x}}$. Since $|\hat{\boldsymbol{x}}| = 2$, $u_i = 2|\boldsymbol{\kappa}_i| \cos\theta_i$: the closer the direction is to that of the template, the larger it is, and beyond 90° it is negative (silent). The amount added, $W_2\boldsymbol{h} = \sum_i h_i \boldsymbol{w}_i$ ($\boldsymbol{w}_i$ is column i of fc2), is the sum of the "contents to add" of the detectors that lit up, weighted by the strength of lighting. Without ReLU, $W_2 W_1$ would be the same as a single 4×4 matrix, and spreading out to sixteen would be meaningless.

A.5 Splitting a score into three parts (Ch. 2)

Since $\boldsymbol{x}_t^{(2)} = \boldsymbol{x}_t^{(0)} + \boldsymbol{A} + \boldsymbol{M}$ ($\boldsymbol{A}$: what Attention added, $\boldsymbol{M}$: what the MLP added),

$$\boldsymbol{\ell} = U\boldsymbol{x}_t^{(2)} = U\boldsymbol{x}_t^{(0)} + U\boldsymbol{A} + U\boldsymbol{M}, \qquad p_j = \frac{e^{\ell_j}}{\sum_k e^{\ell_k}} \tag{8}$$

holds exactly. The condition is that there is no rmsnorm before lm_head. softmax does not change when the same number is added to all seven scores (numerator and denominator change by the same factor), so what matters is the differences between scores. $p_a/p_e = e^{\ell_a - \ell_e}$, which at p3 was $e^{4.57} \approx 96$. In the program, to keep the numbers from getting too large, the largest score is subtracted before taking powers of e (lines 97–101).

A.6 Loss (cross-entropy) (Ch. 3)

$$\mathrm{Loss} = \frac{1}{n} \sum_{t=1}^{n} \big(-\ln p_t(\mathrm{correct})\big) \tag{9}$$

For mia ($n = 4$) at step 1, it is $(1.869 + 2.151 + 1.764 + 2.029)/4 = 1.953$. Giving the same probability $1/7$ to seven letters gives $\mathrm{Loss} = \ln 7 \approx 1.946$; for the original microgpt (27 letters), $\ln 27 \approx 3.30$. The cross-entropy $H(\boldsymbol{y}, \boldsymbol{p}) = -\sum_j y_j \ln p_j$ of the distribution $\boldsymbol{y}$, with 1 for the correct letter and 0 for the others, and the AI's probabilities $\boldsymbol{p}$ is exactly $-\ln p(\mathrm{correct})$.

A.7 Gradient, chain rule, gradient of the scores (Ch. 3)

The gradient is the rate of change of the penalty when a knob w is moved by a small number Δ:

$$\frac{\partial\,\mathrm{Loss}}{\partial w} \approx \frac{\mathrm{Loss}(w + \Delta) - \mathrm{Loss}(w)}{\Delta}. \tag{10}$$

In the example of the main text ($\Delta = 0.001$), $(1.953936 - 1.953475)/0.001 = 0.461$, which agreed with the autograd value 0.4608. When Loss depends on w through y, $\dfrac{\partial\,\mathrm{Loss}}{\partial w} = \dfrac{\partial\,\mathrm{Loss}}{\partial y}\dfrac{\partial y}{\partial w}$ (chain rule); when there are several paths, the products along each path are added. For $f = ab + a$, $\partial f/\partial a = b + 1 = 4$ and $\partial f/\partial b = a = 2$. The local derivative of addition $z = x + y$ is $\partial z/\partial x = 1$, and of multiplication $z = xy$ it is $\partial z/\partial x = y$ (the other number); microgpt's `Value` records these at each computation.

When probabilities and the loss are made from the scores $\boldsymbol{z}$, $\ell = -\ln p_c = -z_c + \ln \sum_k e^{z_k}$ (c: the correct letter), so

$$\frac{\partial \ell}{\partial z_j} = -y_j + \frac{e^{z_j}}{\sum_k e^{z_k}} = p_j - y_j. \tag{11}$$

This is because differentiating $\ln \sum e^z$ by z_j gives exactly the softmax p_j. The sum of the gradients is $\sum_j (p_j - y_j) = 1 - 1 = 0$: the others are pushed down by just as much as the correct score is pushed up.

A.8 Adam (Ch. 3)

With gradient g_t ($m_0 = v_0 = 0$),

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{12}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{13}$$

$$w \leftarrow w - \eta_t \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \qquad \eta_t = 0.01\Big(1 - \frac{t-1}{300}\Big) \tag{14}$$

with $\beta_1 = 0.85$, $\beta_2 = 0.99$ and $\epsilon = 10^{-8}$ (lines 174–182). At step 1 ($t = 1$), $\hat{m}_1 = g_1$ and $\hat{v}_1 = g_1^2$, so $\hat{m}_1/\sqrt{\hat{v}_1} = \pm 1$, and every knob moves by $\eta = 0.01$ (it does not move if $g = 0$). $\beta_1 = 0.85$ corresponds to an average over roughly the last $1/(1-0.85) \approx 7$ gradients.

A.9 Lots, temperature and the probabilities of names (Ch. 4)

Make the cumulative probabilities $C_j = p_1 + \cdots + p_j$ ($C_7 = 1$), and for a uniform random number $0 \le r < 1$, choose the letter j with $C_{j-1} \le r < C_j$. The probability that r falls in this section equals the width of the section, p_j, so letter j is chosen with probability p_j. The probability at temperature T is

$$p_j(T) = \frac{e^{z_j/T}}{\sum_k e^{z_k/T}}. \tag{15}$$

Viewing the score z as the energy with a minus sign, this has the same form as the Boltzmann distribution of statistical mechanics, $p_j = e^{-E_j/kT} / \sum_i e^{-E_i/kT}$ (E: energy, k: Boltzmann constant). The ratio of the probabilities of two letters is $p_j/p_k = e^{(z_j - z_k)/T}$; at $T = 0.5$ the ratio is squared. As $T \to 0$, only the letter with the highest score is chosen (greedy decoding); as $T \to \infty$, all letters approach 1/7.

The probability of a name is the product of the (temperature-adjusted) probabilities at each position, and the expected count when making N names is $N \times P(\text{name})$. At temperature 1, $P(\text{aia}) = 0.509 \times 0.035 \times 0.992 \times 0.998 = 0.0175$, so about 17 are expected in 1000 (actually 22). At temperature 0.5, $P(\text{emma}) = 0.096$, so about 1.9 are expected in 20 (actually 1), and about 96 in 1000 (actually 88).

A.10 Number of knobs (Ch. 5)

One layer contains $4d^2$ in the four Attention tables and $4d \times d + d \times 4d = 8d^2$ in the two MLP tables, $12d^2$ in all.

$$\text{microgpt: } 2Vd + Bd + 12d^2 L \tag{16}$$

$$\text{GPT-2, GPT-3: } Vd + Bd + L(12d^2 + 13d) + 2d \tag{17}$$

GPT-2 and GPT-3 share lm_head with wte, and include the knobs of biases and normalization ($13dL + 2d$). The small example ($V = 7, d = 4, B = 5, L = 1$) gives $56 + 20 + 192 = 268$;

microgpt $(27, 16, 16, 1)$ gives 4,192; the smallest GPT-2 $(50{,}257, 768, 1{,}024, 12)$ gives 124,439,808; and GPT-3 $(50{,}257, 12{,}288, 2{,}048, 96)$ gives about 174.6 billion (about 175 billion). Most of it is the $12d^2L$ part.

A.11 Amount of computation in Attention (Ch. 5)

In a text of length n, position t looks at t positions, so the number of pairs is $1 + 2 + \cdots + n = n(n+1)/2$. Each pair needs an inner product of q and k (d multiplications), so the amount of computation is roughly proportional to $n^2 d$. In practice, the q's and k's of all positions are put together into matrices Q and K, $QK^{\mathsf{T}}/\sqrt{d}$ is computed at once, and the upper-right part is hidden so that positions ahead are not looked at before softmax is taken. When making names or text, the k and v of earlier positions are kept (KV cache), so only the computation for the one new position is needed.

A.12 layernorm and GELU (Ch. 5)

$$\text{layernorm}(\boldsymbol{x}) = \boldsymbol{\gamma} \odot \frac{\boldsymbol{x} - \mu}{\sqrt{\sigma^2 + \varepsilon}} + \boldsymbol{\beta}, \qquad \text{GELU}(u) = u\,\Phi(u) \tag{18}$$

μ is the average, σ^2 the variance, $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are knobs that are trained, and Φ is the cumulative distribution function of the standard normal distribution. rmsnorm does not subtract the average; it only divides by the square root of the average of the squares (microgpt also omits $\boldsymbol{\gamma}$). GELU is almost u when u is a large positive number and almost 0 when it is a large negative number, and it bends smoothly near 0. ReLU makes that bend a corner. GPT-2's multiplication is $\boldsymbol{y} = W\boldsymbol{x} + \boldsymbol{b}$ ($\boldsymbol{b}$ is the bias), while microgpt has only $\boldsymbol{y} = W\boldsymbol{x}$.

Appendix B Glossary

Term	Meaning in this booklet
LLM	large language model, such as ChatGPT or Claude
Token	the unit the AI predicts at one time; one letter in microgpt
Tokenizer	the rule for going back and forth between letters and numbers
Vocabulary	the kinds of tokens handled; seven in the small example
BOS	beginning of sequence; the mark for the start and end of a name
Weight (parameter)	a number adjusted by practice; called a "knob" in this booklet
Hyperparameter	a setting decided by people before practice
Blackboard (residual stream)	four numbers per position, added to as the computation goes
wte / wpe	token / position embedding; the table of letters / table of positions (first note)
rmsnorm	root mean square normalization; evens out the size of the four numbers
Attention	the mechanism that goes to look at the blackboards of earlier positions
q / k / v	query / key / value: the request / heading / contents
softmax	turns scores into shares (probabilities) totaling 1
MLP	multi-layer perceptron; sixteen detectors and their additions (fc1, fc2)
ReLU	rectified linear unit; sets negatives to 0 (whether a detector lights up)
lm_head	language model head; the table that reads the blackboard and gives seven scores
Logit	the score before softmax
Loss	the average penalty $(-\ln p)$ of the probability given to the correct answer
Cross-entropy	the mathematical name of this loss

Term	Meaning in this booklet
Perplexity	how many choices it is unsure among on average
Gradient	the rate at which the loss increases when a knob is increased a little
Gradient descent	the method of turning little by little in the direction opposite to the gradient
Automatic differentiation (autograd)	the mechanism that finds gradients by tracing the record of the computation backward
Backpropagation	the method of passing gradients from the output side using the chain rule
Chain rule	the rule of multiplying together how effects are passed on (gear ratios)
Learning rate	the guide to how far to turn at one time ($0.01 \to 0$)
Adam	adaptive moment estimation; a way of turning that uses momentum and evened-out sizes
Generation	having the trained AI make new names
Temperature	a setting that divides the scores to sharpen or flatten the probabilities
Greedy decoding	the method of always choosing the most probable letter
KV cache	the mechanism of keeping and reusing the k and v of earlier positions
Hallucination	producing something plausible but untrue
Layer	one pair of Attention and MLP; large AIs stack it many times
Head	a group into which Attention is divided
GPU	graphics processing unit; a device that can do a huge number of computations of the same kind at once
Pretraining	practice predicting the next token on a large amount of text
RLHF	reinforcement learning from human feedback; additional training based on human ratings
Scaling law	the tendency of the loss to go down regularly as knobs and data are increased
Interpretability	research into working out what is happening inside an AI

Appendix C How the numbers were obtained, and where to get the program

- **Training settings**: the training data are the four names emma, ava, mia and anna. Blackboard width 4, length read at once 5, number of heads 1, 300 practice steps, random seed 42. These are the same settings as the small example in Section 5 of "Reading microgpt" and as "A Look Inside a Small AI's Head." We confirmed that the training is the same by checking that the loss at step 1 (1.9535), the loss at step 300 (0.3563) and the trained row of a in wte, $(+0.119, -0.207, +0.194, +0.183)$, agree.
- **Numbers in Chapter 2**: using the trained weights, all intermediate computations at each position were found with a program that rewrites the same computation as `gpt()` with ordinary decimal numbers (`compute_part2.py`).
- **Numbers in Chapter 3**: while carrying out the same training, the average penalty of the 18 problems and the probabilities in each scene were found with the knobs after each step (`compute_part3.py`). The numerical check of the step-1 gradient was done by moving a knob by 0.001.
- **Numbers in Chapter 4**: after training, twenty names were made in the same way as the original program (`random.choices`, temperature 0.5) (`compute_part4.py`). The twenty names at other temperatures were made again from the state of the random numbers right after training. Random seed 7 was used for the 1000-name counts and for generation during training (without affecting the random numbers of training).

- **Numbers in Chapter 5**: the numbers of knobs were counted from the published structures, and the size of the original data was counted from `names.txt` (`compute_part5.py`). The sizes of GPT-2 and GPT-3 are from Section 8.1 of "Reading microgpt," and the amount of GPT-3's training data is from Brown et al. (2020).
- **Figures**: made with `mkfigs_anatomy_en.py`. Slides with the same content (PowerPoint, Parts 1–5, in Japanese) have also been made.
- **The microgpt program**: https://gist.github.com/karpathy/8627fe009c40f57531cb18360106ce95 (version retrieved on September 30, 2026). Because no license is stated, the program is not included in this booklet.