LLMs Unplugged

Understand AI by building it yourself

Speaker notes

Open with the promise: understand AI by building one yourself, by hand. No computers, no maths background needed.

Acknowledgement of Country

Speaker notes

Acknowledge the Traditional Custodians of the land you're meeting on, and pay respects to Elders past and present. Keep it brief and sincere; say it in your own words.

activity

everyone stand up

Speaker notes

Quick energiser. Read each line and let people sit; it speeds up, and by "the last 5 minutes" almost everyone's down. Lands the point that this stuff is already everywhere. ~2 min, don't linger.

sit down if you

have **never** used ChatGPT

sit down if you

haven’t used it in the last **month**

sit down if you

haven’t used it in the last **week**

sit down if you

haven’t used it in the last **day**

sit down if you

haven’t used it in the last **hour**

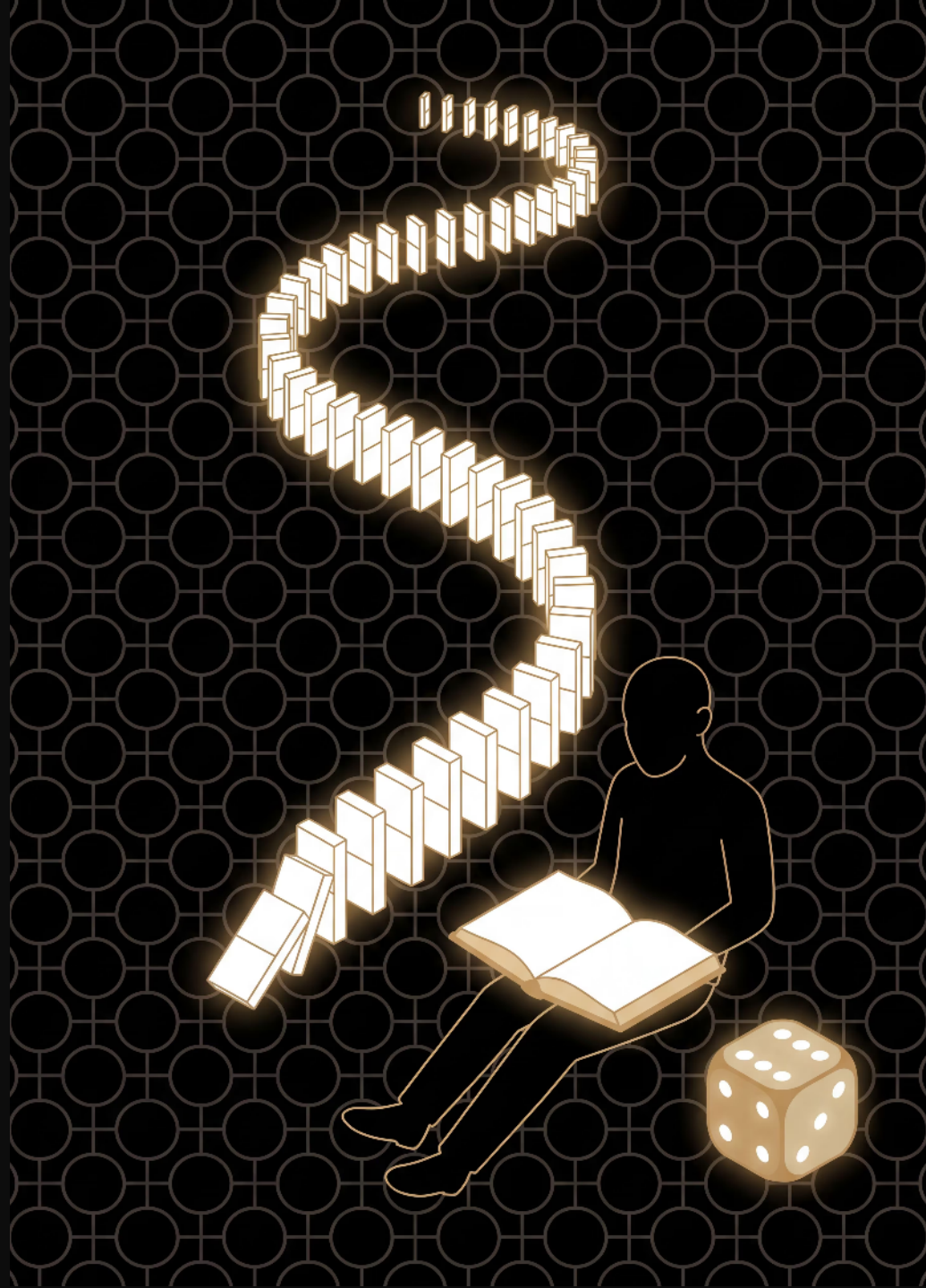
sit down if you

haven’t used it in the last **5 minutes**

What is this about?

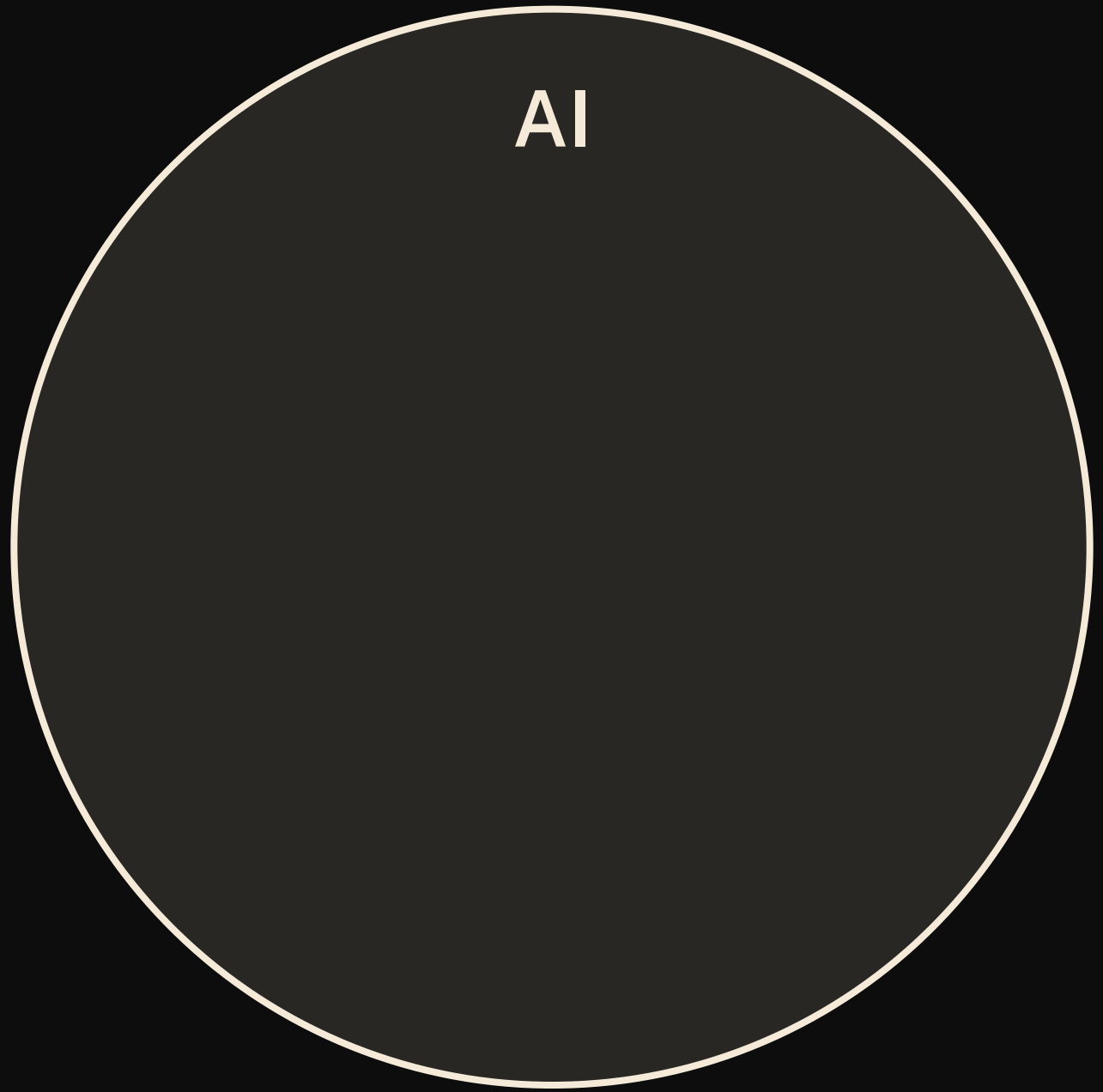
you'll **build** your own language model — from scratch — with just a kids book, pen & paper, and some dice rolling

you'll **learn** how language models work by spotting patterns in text to generate new text



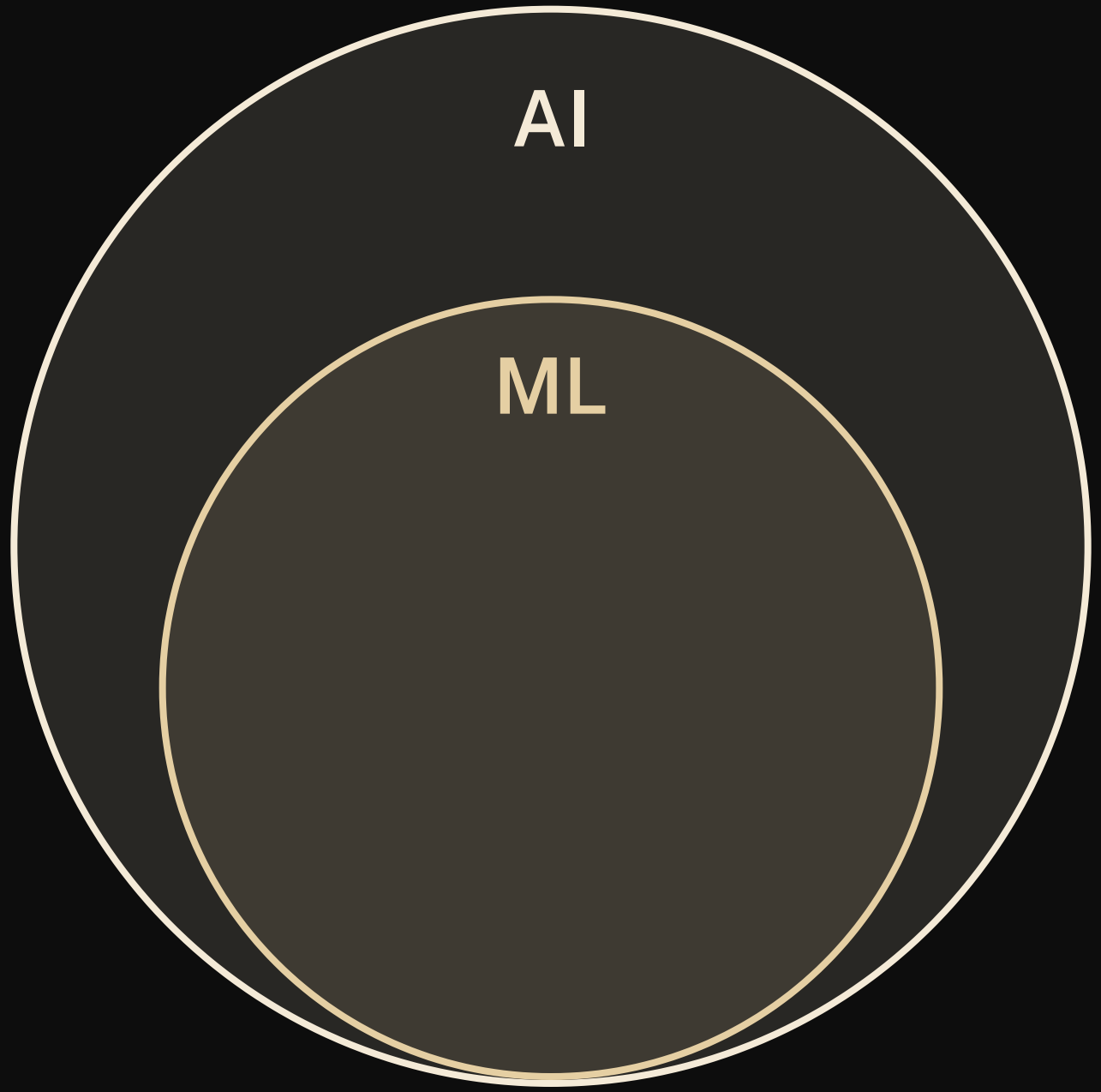
Speaker notes

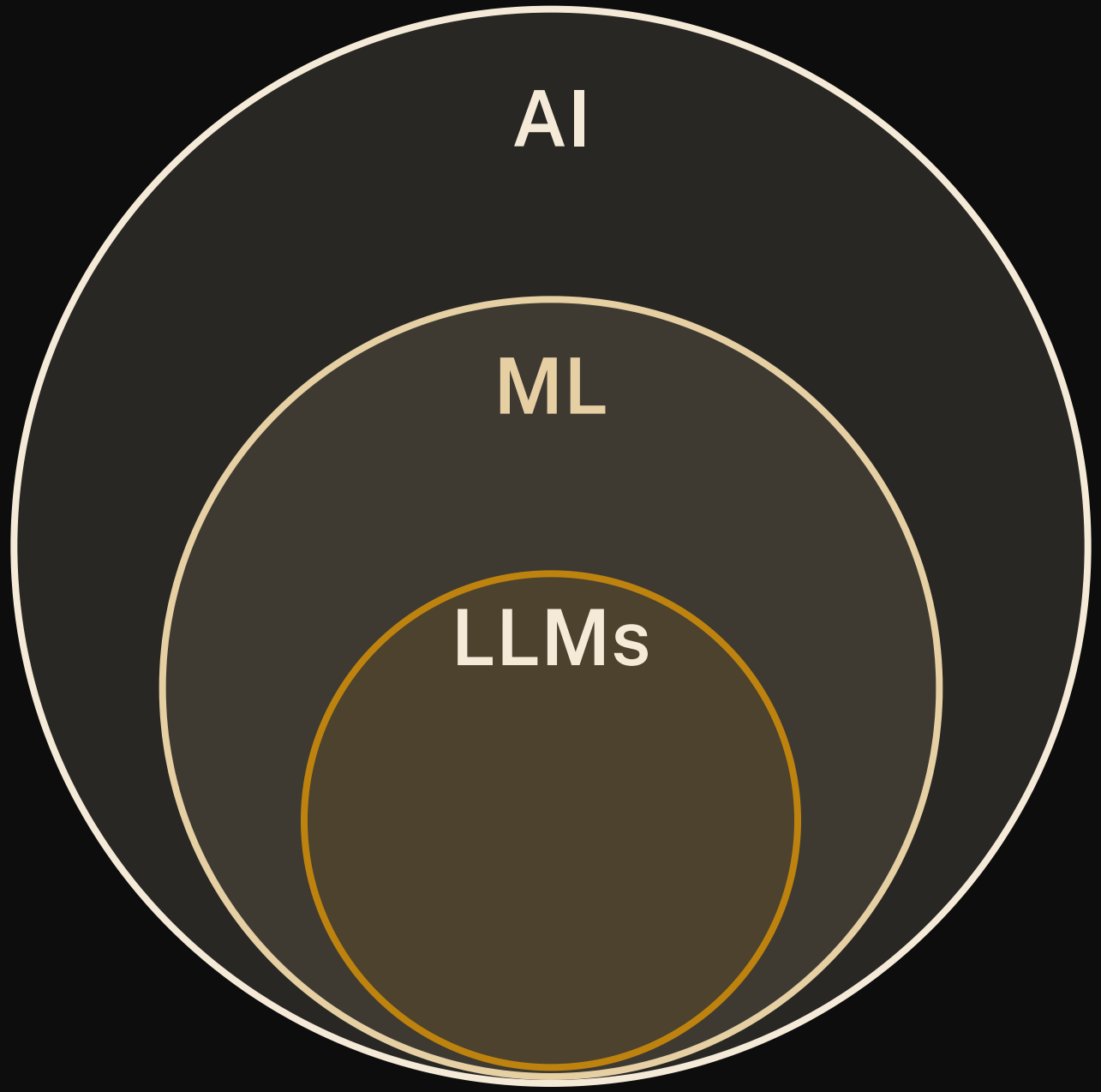
Frame the next stretch: build a model from a kids' book, pen, paper and dice, by spotting patterns. ~1 min, then move.

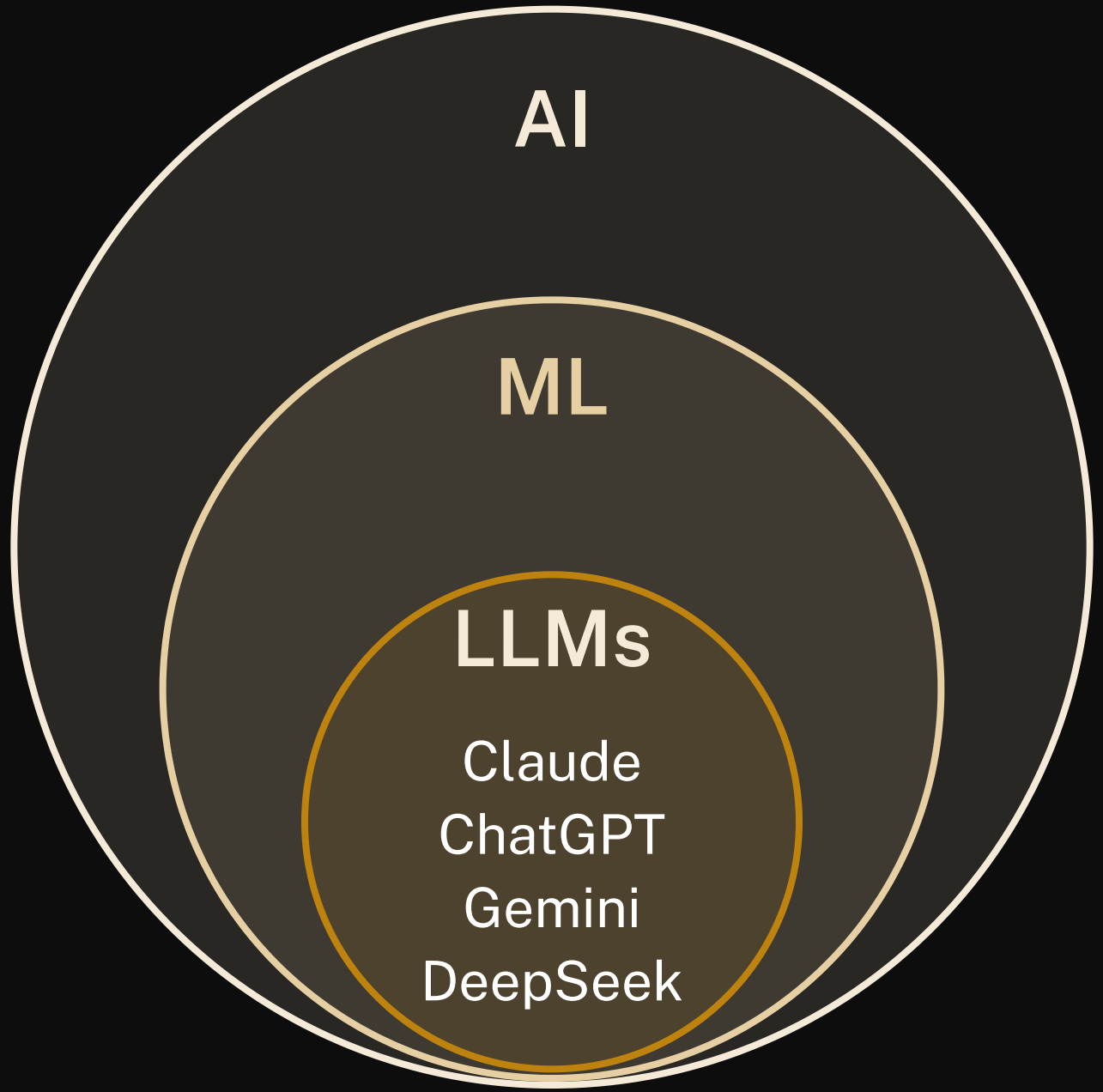

AI

Speaker notes

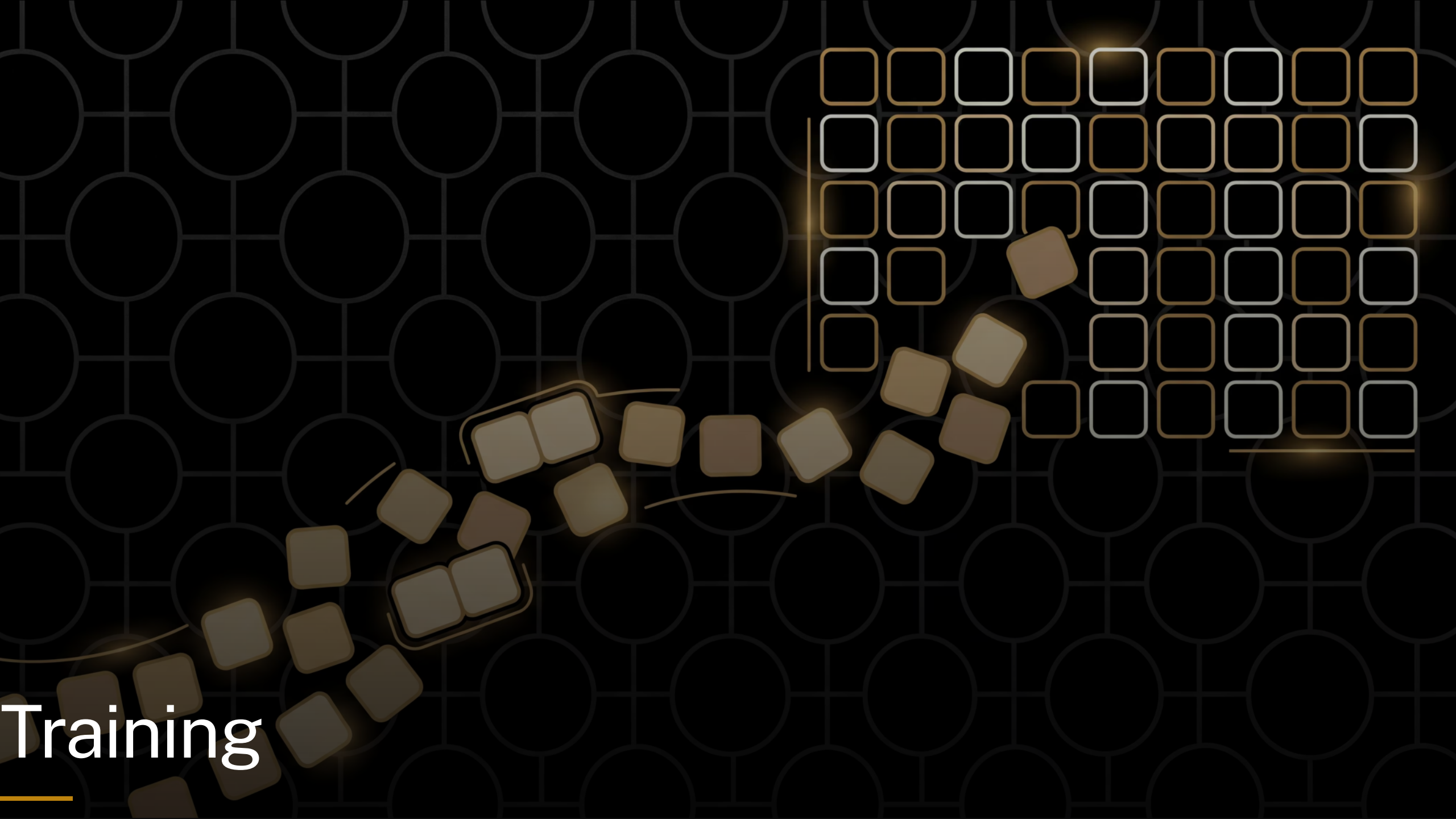
No text on this one — talk over the build. AI: the umbrella term, any system doing things we'd call "smart" (chess engines, spam filters, route planners). Click — ML: the subset that learns patterns from data instead of following hand-written rules; the model you'll build today lives here. Click — LLMs: machine learning models trained on huge amounts of text to predict the next token — exactly what you're about to do by hand, just scaled up. Click — and the chatbots in the news (Claude, ChatGPT, Gemini, DeepSeek) are LLMs wrapped in an app. Same species as the thing you'll build with pen and dice.


AI
ML


AI
ML
LLMs


AI
ML
LLMs
Claude
ChatGPT
Gemini
DeepSeek

llmsunplugged.org



Training

The recipe

walk through your text and tally up which tokens follow which in a grid

Speaker notes

1. **tidy up** your text: lowercase everything, treat words and single punctuation marks (`.` `,` `!` `?` `;` `:`) as separate tokens
2. **set up** your grid: first token goes in both the row & column headers
3. **fill in** the grid: for each consecutive pair of tokens, add a tally mark in that cell (add new rows/columns as needed)

Example

“Run, Spot, run. See Spot run.”

after tidying up:

`run` `,` `spot` `,` `run` `.` `see` `spot` `run` `.`

The empty grid

`run` `,` `spot` `,` `run` `.` `see` `spot` `run` `.`

Training: run → ,

run , spot , run . see spot run .

	run	,			
run					
,					

Training: , → spot

run , spot , run . see spot run .

	run	,	spot		
run					
,					
spot					

Training: spot → ,

run , spot , run . see spot run .

	run	,	spot		
run					
,					
spot					

Speaker notes

`,` is already in our vocabulary — no new column needed!

Training: , → run

run , spot , run . see spot run .

	run	,	spot		
run					
,					
spot					

Training: run → .

run , spot , run . see spot run .

	run	,	spot	.	
run					
,					
spot					
.					

Speaker notes

. is a new token — punctuation gets its own row and column too

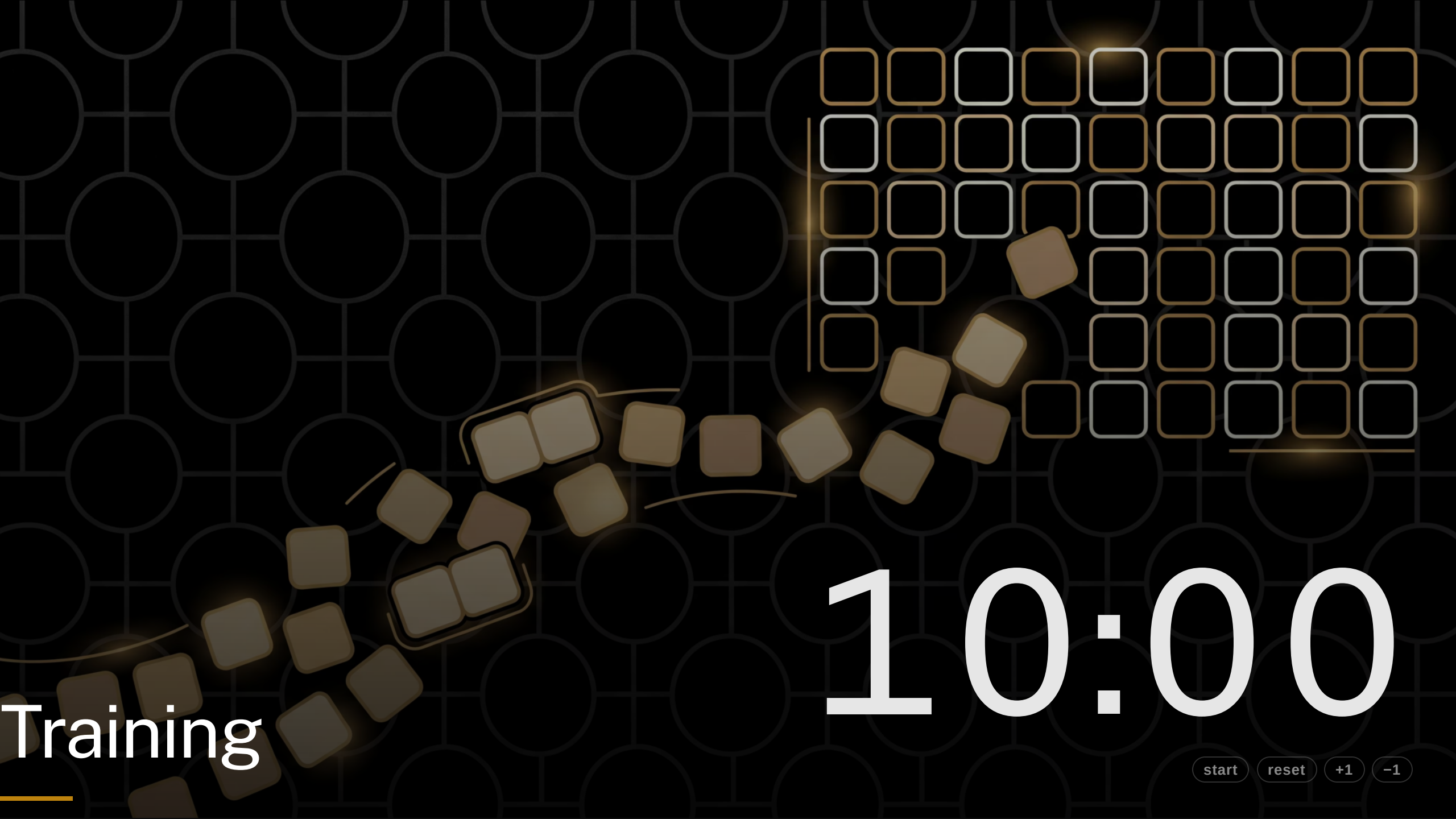
Complete model

`run` `,` `spot` `,` `run` `.` `see` `spot` `run` `.`

	`run`	`,`	`spot`	`.`	`see`
`run`		|		||	
`,`	|		|		
`spot`	|	|			
`.`					|
`see`			|		

Speaker notes

with the rest of the text tallied the same way, the grid is complete. `run` → `.` came up twice, so its cell has two marks — those counts are what make some next-words more likely than others.


10:00
Training
start
reset
+1
−1

Speaker notes

Hand out grids; groups tally their own text. 10 min, circulate --- the usual snag is that a token only gets a new row and column the first time it appears.

The *language* of language models

- model
- token
- vocabulary
- training

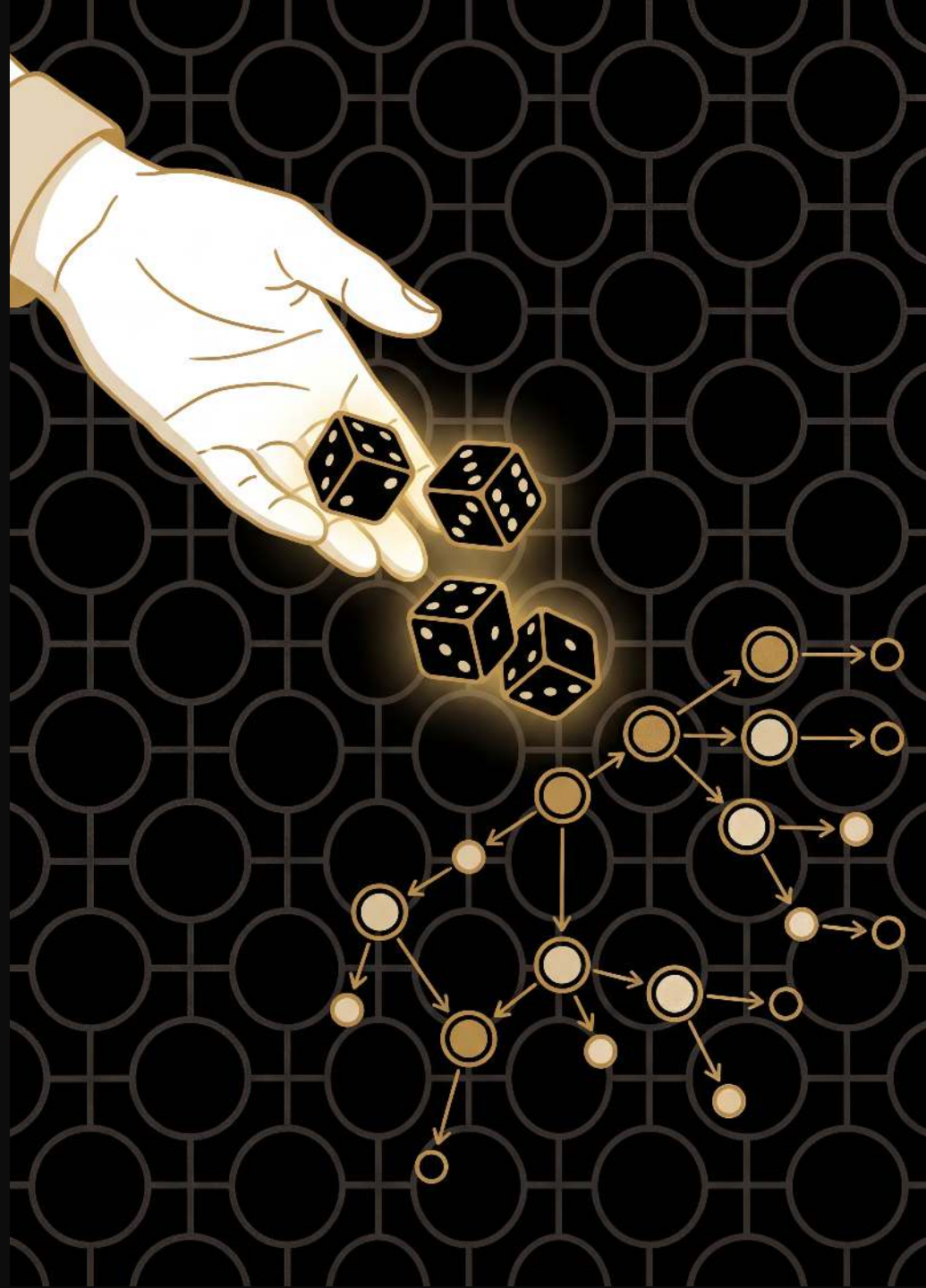
Speaker notes

Gloss each: model = the thing that captures the patterns (your grid); token = one word or punctuation mark; vocabulary = all the distinct tokens it has seen; training = counting which token follows which.

Generation

The recipe

use your grid to generate new text, rolling dice to choose each next word



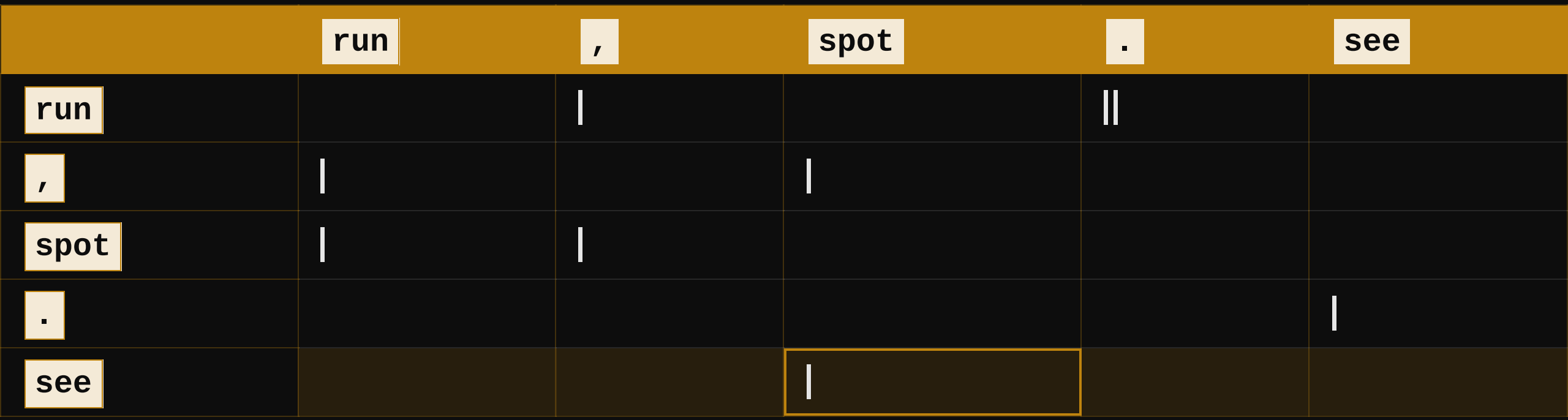
Speaker notes

1. **choose** a starting word from your grid (any word---not just the first one)
2. **look** at that word's row to find all possible next words and their counts
3. **roll dice** weighted by the counts to pick the next word
4. **write down** the chosen word and make it your new starting word
5. **repeat** from step 2

Generation: start with `see`

`see` `spot`

one option — no roll needed



Speaker notes

any word will do as a seed---we're starting from `see`, which is the last row, to show you don't have to begin at the top. Only one option here, so no roll.

Generation: from `spot`

`see` `spot`

2 options — roll the die!

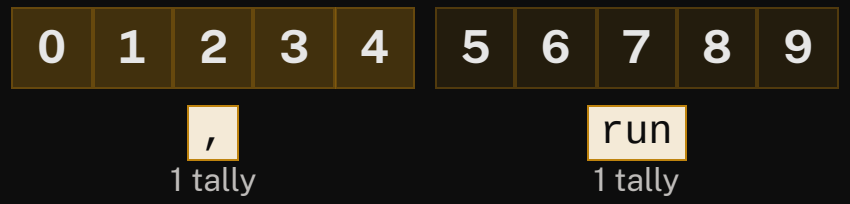
	`run`	`,`	`spot`	`.`	`see`
`run`		|		||	
`,`	|		|		
`spot`	|	|			
`.`					|
`see`			|		

Speaker notes

two options in this row (`,` and `run`) with equal tallies---highlight both, then roll

How the die chooses: `spot`

`spot` → ? roll a d10



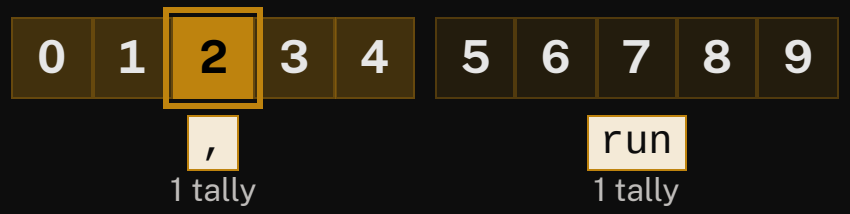
equal tallies → equal chances

Speaker notes

equal tallies, so each option gets an equal share of the ten d10 faces---0-4 for one, 5-9 for the other. Pause here before rolling.

How the die chooses: `spot`

`spot` → ? roll a d10



equal tallies → equal chances

rolled **2** → `,`

Speaker notes

now roll --- a 2 lands in the `,` band

Generation: spot → ,

see spot ,

rolled 2 → ,

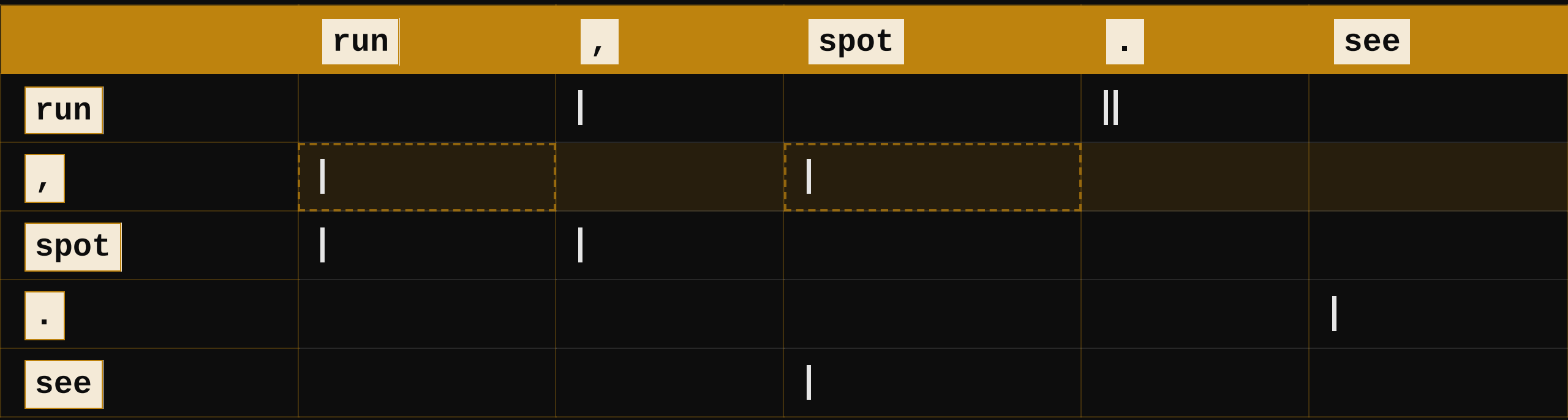
Speaker notes

the 2 landed in the `,` band, so `,` is our next word

Generation: from `,`

`see` `spot` `,`

2 options — roll the die!



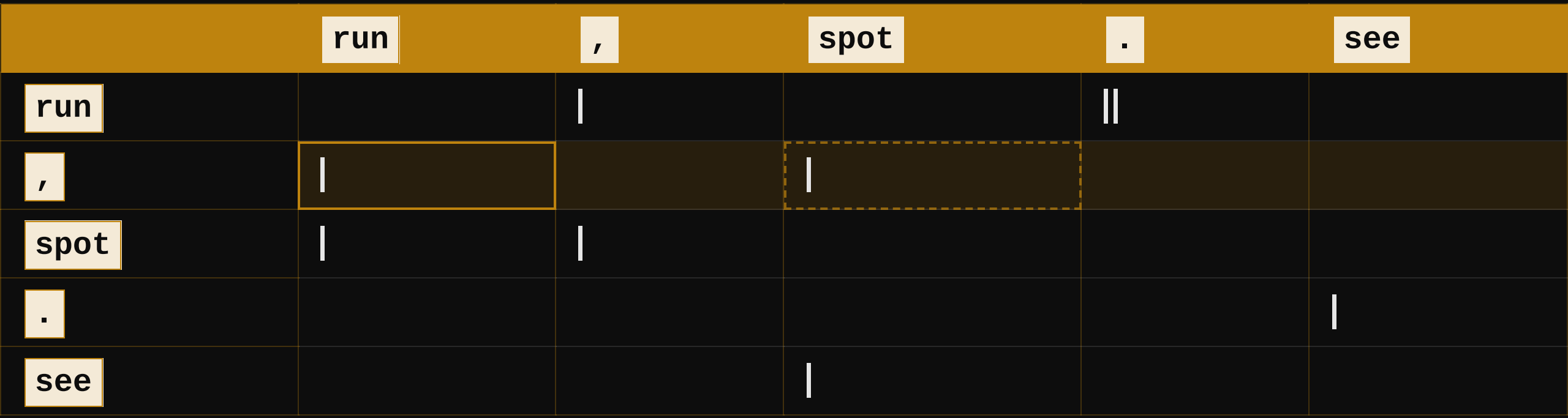
Speaker notes

another two-option row (`spot` and `run`), still equal --- both highlighted, then roll

Generation: , → run

see spot , run

rolled 7 → run



Speaker notes

rolled a 7, which lands in the `run` band

Generation: from `run`

`see` `spot` `,` `run`

2 options — roll the die!

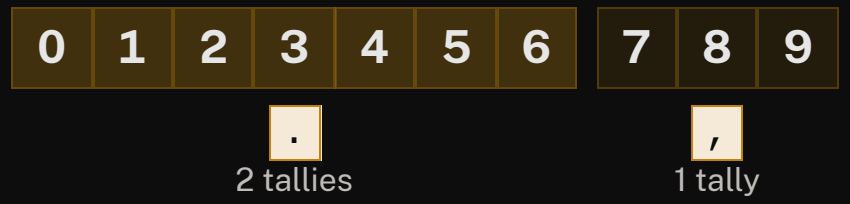
	`run`	`,`	`spot`	`.`	`see`
`run`		|		||	
`,`	|		|		
`spot`	|	|			
`.`					|
`see`			|		

Speaker notes

two options again --- but `.` has two tallies to `,`'s one, so this isn't a 50/50 roll

How the die chooses: `run`

`run` → ? roll a d10



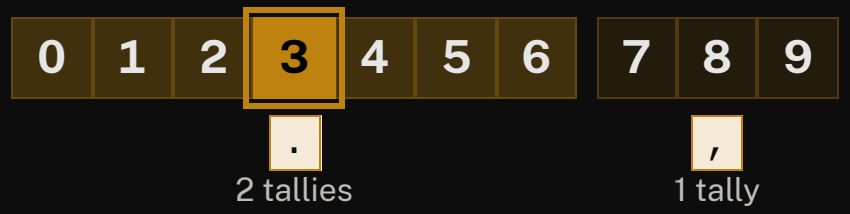
more tallies → more faces → more likely

Speaker notes

`.` has twice the tallies, so it gets more of the faces (0-6) than `,` (7-9). With a ten-sided die that's the closest split rather than exactly 2:1 --- the point is more tallies → more faces → more likely. Pause here before rolling.

How the die chooses: `run`

`run` → ? roll a d10



more tallies → more faces → more likely

rolled **3** → `.`

Speaker notes

now roll --- a 3 sits in the `.` band

Generation: run → .

see spot , run .

rolled 3 → .

	run	,	spot	.	see
run		|		||	
,	|		|		
spot	|	|			
.					|
see			|		

Speaker notes

the 3 sits in the `.` band, so we pick `.`

Generation: from `.`

`see` `spot` `,` `run` `.` `see`

one option — no roll needed

	`run`	`,`	`spot`	`.`	`see`
`run`		|		||	
`,`	|		|		
`spot`	|	|			
`.`					|
`see`			|		

Speaker notes

only one option (`see`) --- no roll. Note we keep rolling straight past the full stop; the model has no idea a sentence just ended.

Generation: back to `see`

`see` `spot` `,` `run` `.` `see`

one option — no roll needed

	`run`	`,`	`spot`	`.`	`see`
`run`		|		||	
`,`	|		|		
`spot`	|	|			
`.`					|
`see`			|		

Speaker notes

and so on---we've looped back to where we started. Keep going for as long as you like; you decide when to stop.

Generated text

“see spot, run. see”

a new sentence — not in the training data!

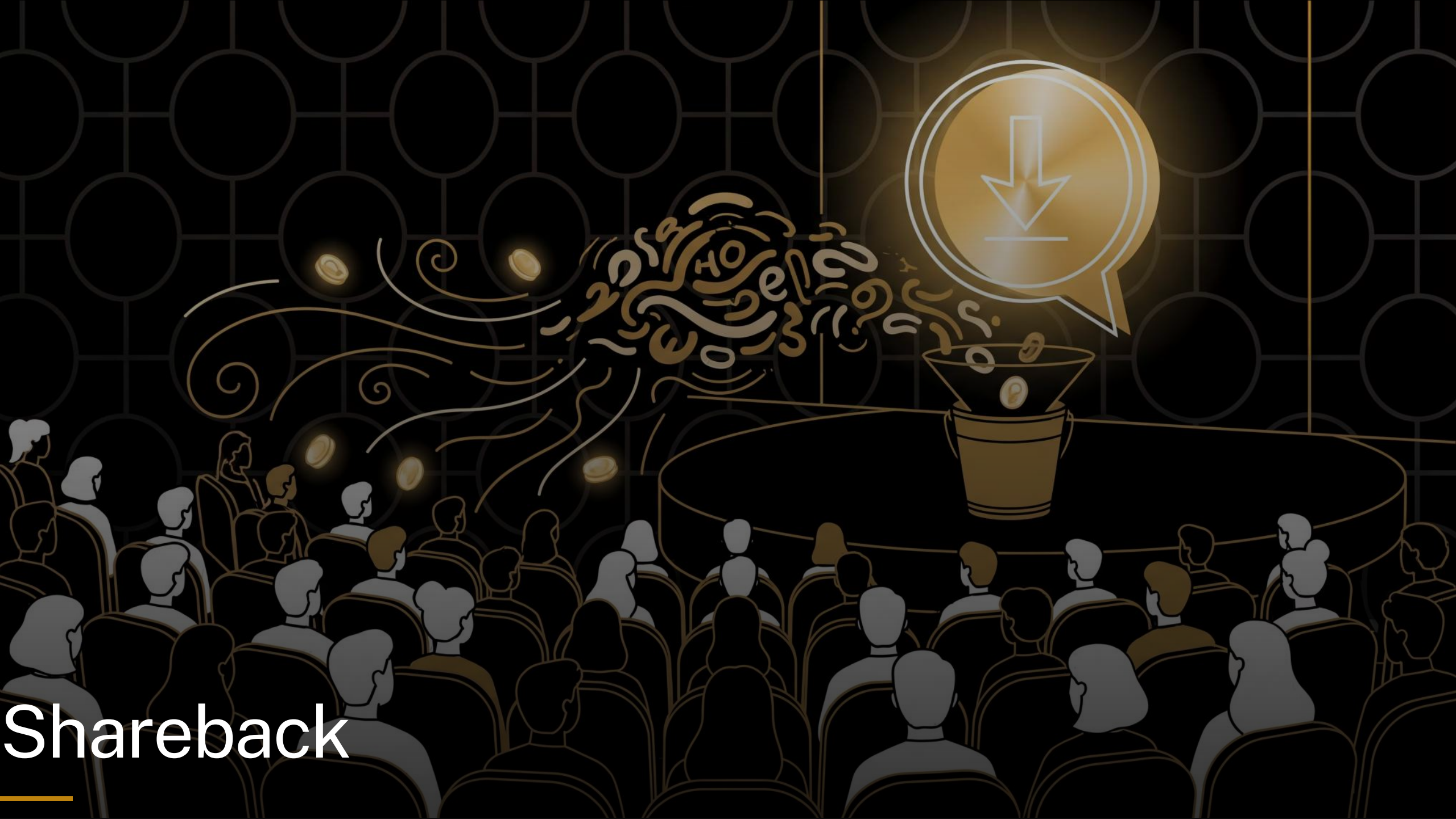
Generation

10:00

start reset +1 −1

Speaker notes

Groups generate from their grid, rolling dice on each choice. 10 min, circulate. Where a row has only one option, no roll needed.



Shareback

Speaker notes

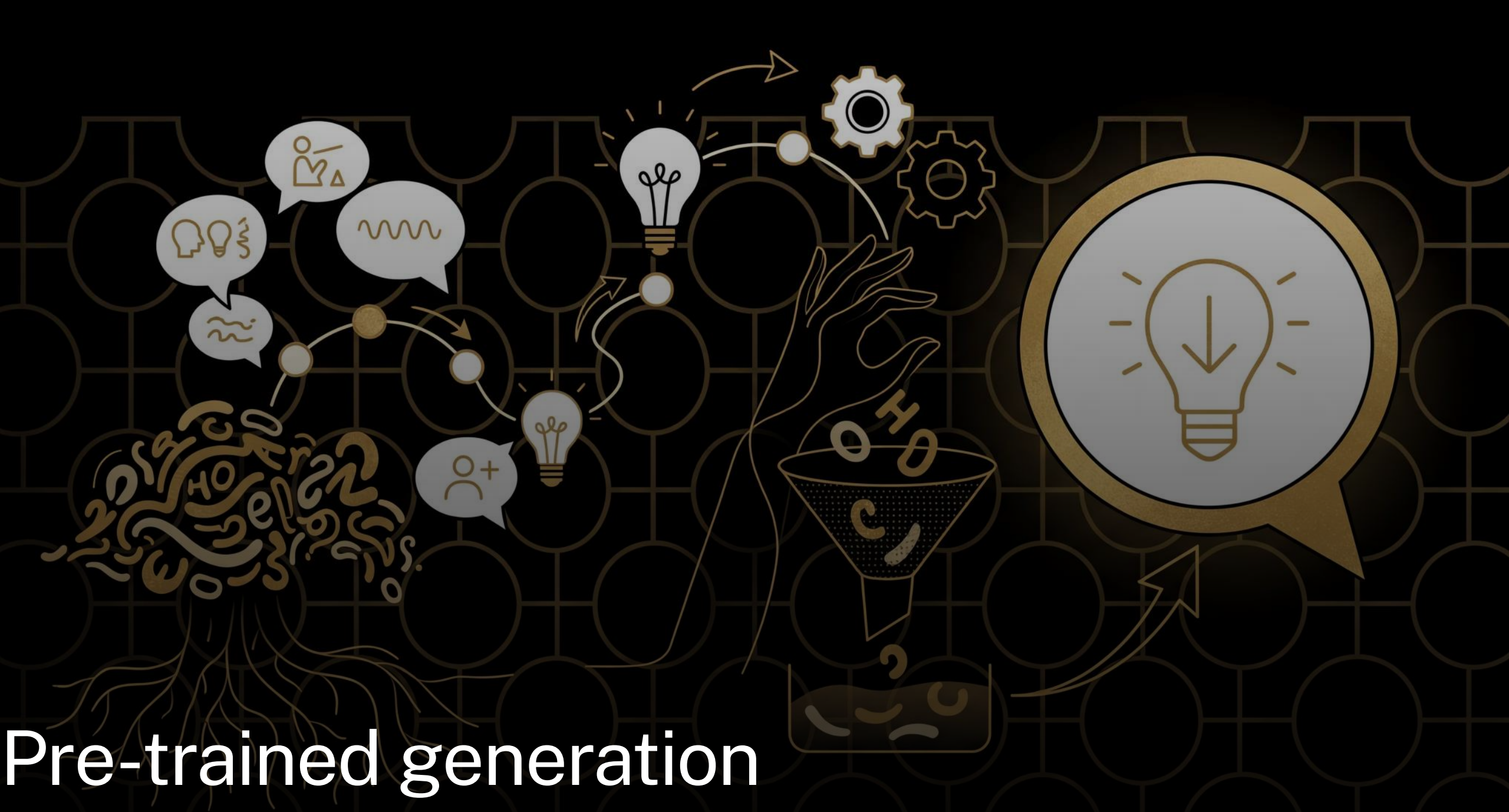
Take a couple of groups' generated sentences; celebrate the weird ones. ~5 min.

The *language* of language models

- prompt
- response/completion
- context window

Speaker notes

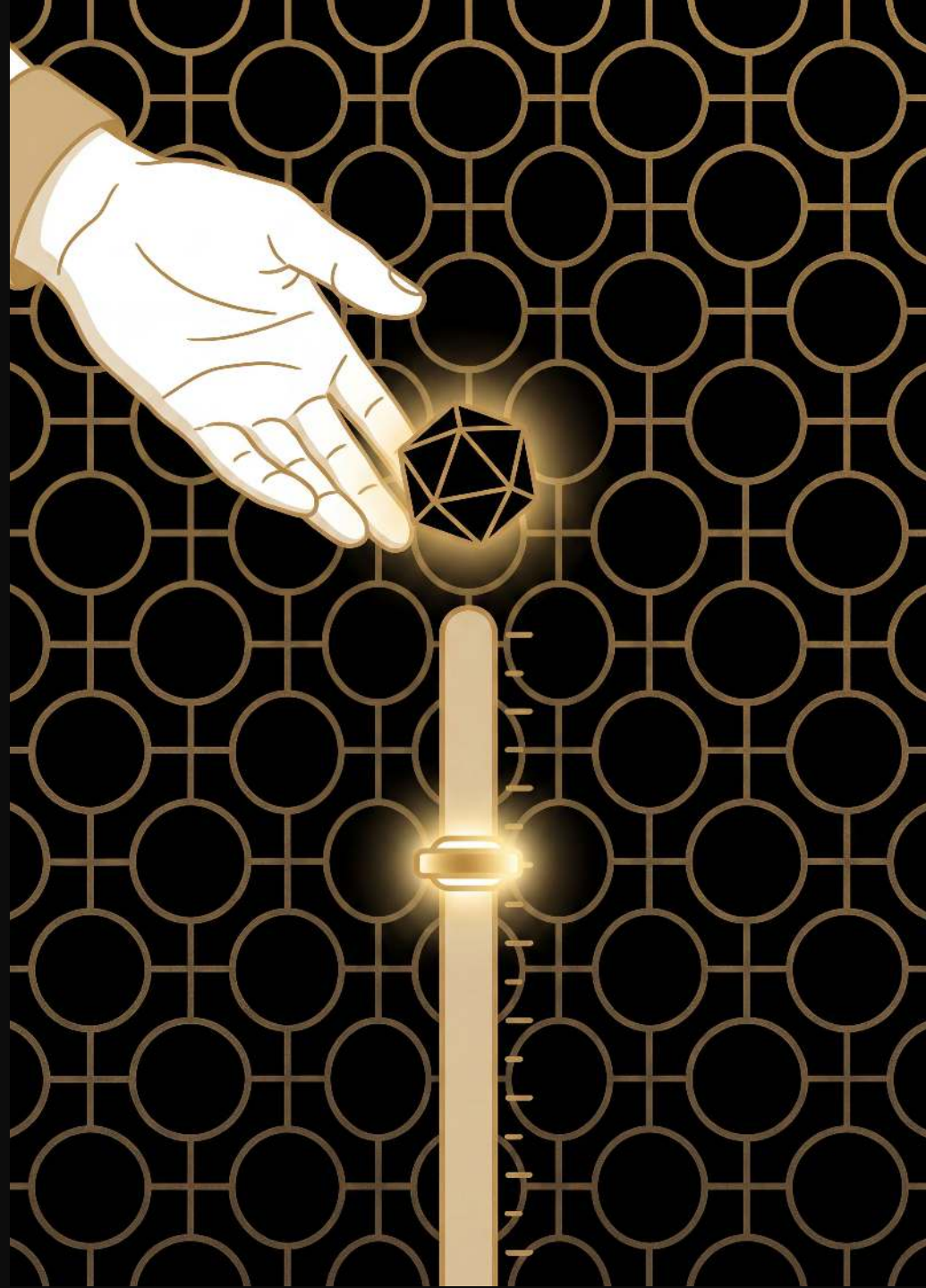
Gloss each: prompt = the starting text you give it; response/completion = the text it generates back; context window = how much it can "see" at once (here, just the current word).



Pre-trained generation

The recipe

use the booklet to generate new text, rolling one or two d10s to choose each next word



Speaker notes

1. **choose** a starting word --- any bold word in the booklet
2. **look up** that word's entry to see possible next words and their thresholds
3. **roll** your d10(s) and scan the thresholds to find the next word
4. **write down** the chosen word and make it your new starting word
5. **repeat** from step 2

Pre-trained: start with `the`

`the` `cat`

the ♦♦ 49|cat 79|dog 99|hat **rolled 27**

cat ♦ 5|sat 9|ran

sat .

. the

ran .

dog ♦ 6|sat 9|ran

hat ♦ 4|sat 9|ran

Speaker notes

a bigger model — `the` is so common it needs **two** dice (roll two d10s for 00 to 99)

Pre-trained: from `cat`

`the` `cat` `sat`

the ♦♦ **49**|cat **79**|dog **99**|hat

cat ♦ **5**|sat **9**|ran **rolled 4**

sat .

. the

ran .

dog ♦ **6**|sat **9**|ran

hat ♦ **4**|sat **9**|ran

Speaker notes

`cat` is rarer than `the` — just **one** die, roll a d10

Pre-trained: from `sat`

`the` `cat` `sat` `.`

the ♦♦ 49|cat 79|dog 99|hat

cat ♦ 5|sat 9|ran

sat .

. the

ran .

dog ♦ 6|sat 9|ran

hat ♦ 4|sat 9|ran

Speaker notes

only one option — no dice roll needed

Pre-trained: from .

the cat sat . the

the ♦♦ 49|cat 79|dog 99|hat

cat ♦ 5|sat 9|ran

sat .

. the

ran .

dog ♦ 6|sat 9|ran

hat ♦ 4|sat 9|ran

Speaker notes

only one option — no dice roll needed

Pre-trained: from `the` again

`the` `cat` `sat` `.` `the` `dog`

the ♦♦ 49|cat 79|dog 99|hat **rolled 63**

cat ♦ 5|sat 9|ran

sat .

. the

ran .

dog ♦ 6|sat 9|ran

hat ♦ 4|sat 9|ran

Speaker notes

two dice again — 63 is past `cat`'s 49, so we land on `dog`

Pre-trained: from dog

`the` `cat` `sat` `.` `the` `dog`

the ♦♦ **49**|cat **79**|dog **99**|hat

cat ♦ **5**|sat **9**|ran

sat .

. the

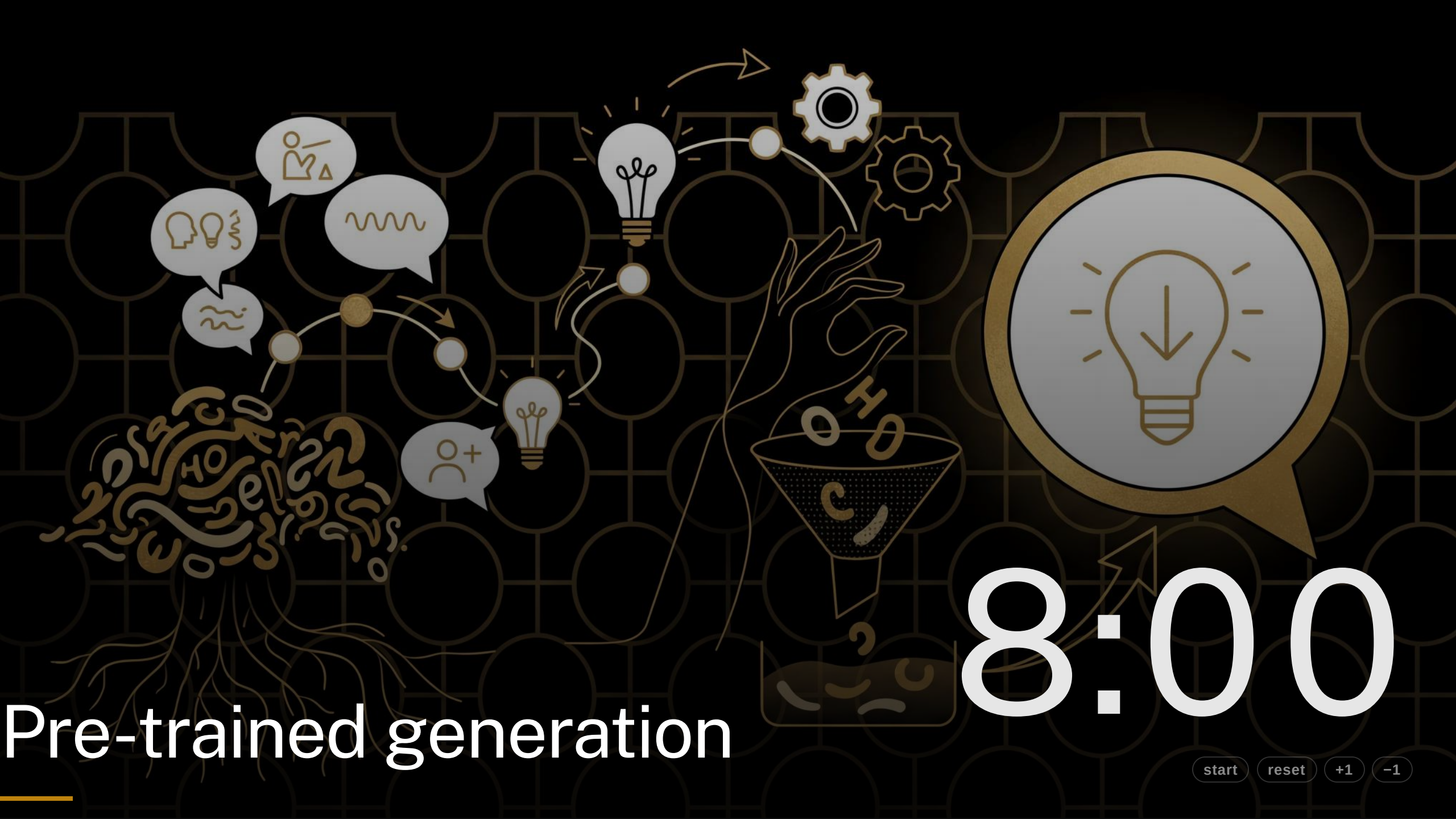
ran .

dog ♦ **6**|sat **9**|ran

hat ♦ **4**|sat **9**|ran

Speaker notes

and so on...



Pre-trained generation

8:00

start reset +1 −1

Speaker notes

Same as before but from the booklet — a bigger model, one or two d10s. 8 min, circulate.

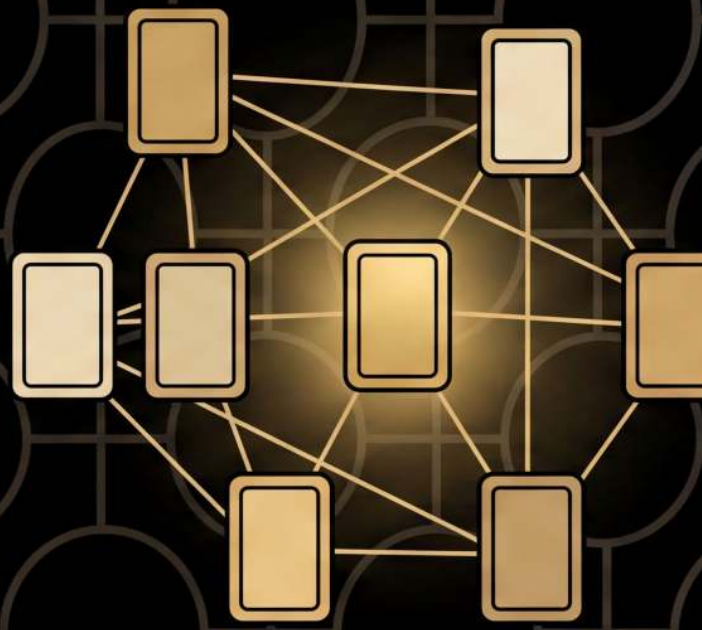
Shareback

Speaker notes

Compare with the hand-built model: a bigger training text gives more varied, more natural output. Invite ~5 groups to share back rather than the whole room. ~5 min.

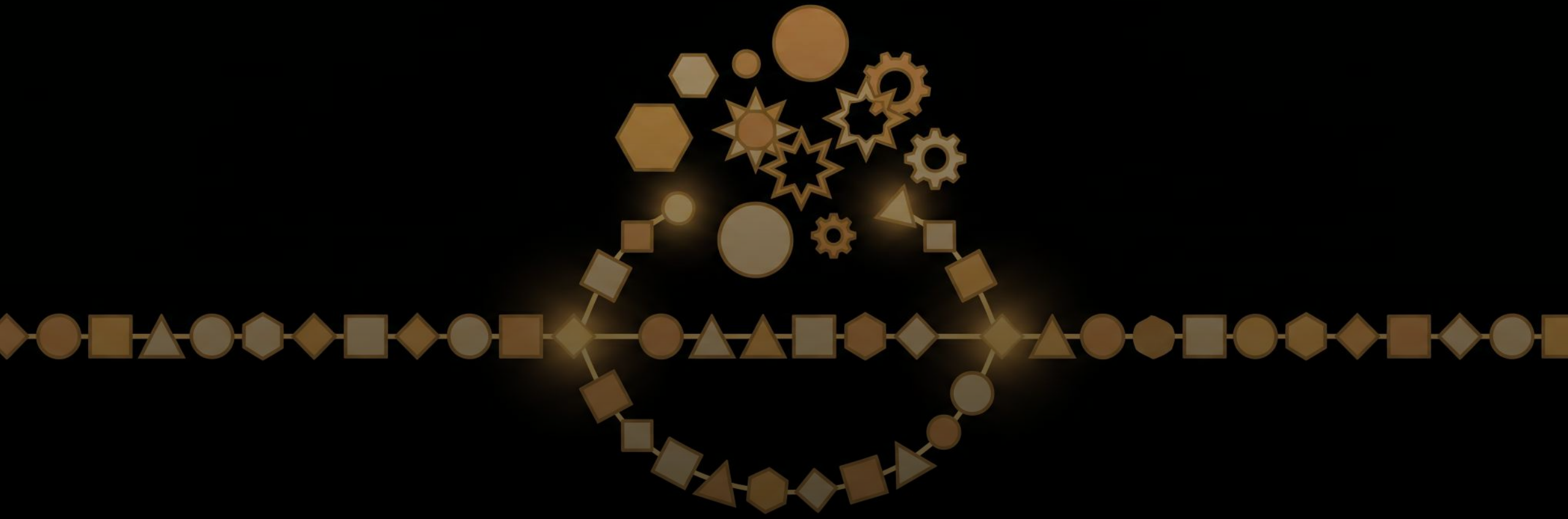
The *language* of language models

- pre-training
- foundation model



Speaker notes

Gloss each: pre-training = the big training run done ahead of time on huge amounts of text; foundation model = the general-purpose model that run produces, which other things are built on.



Agentic AI

Speaker notes

Section beat: an agent is a model that can pause, call a tool, and use the result. We bolt that onto your model.

What makes a model an “agent”?

an agent is a model that can **call tools**: pause generation, get information from “outside” the model, and continue

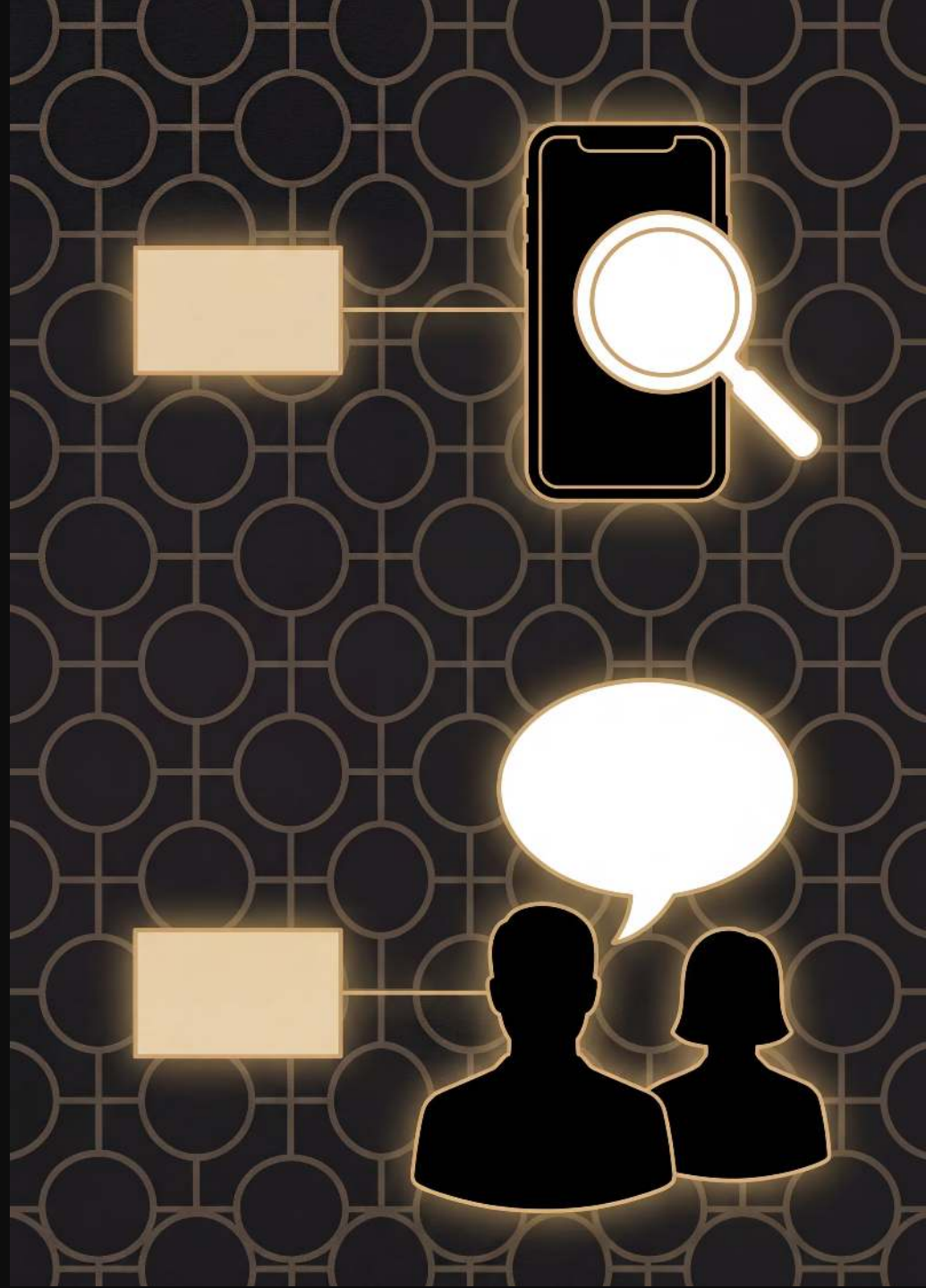
the loop: generate → trigger token → call tool → write the result → keep going

Speaker notes

The beat: the model pauses, gets information from outside itself, then continues. The loop is the thing.

The recipe

generate from your model as before, but every punctuation token triggers a **tool call**: text the sentence so far to 3 friends, and the whole first reply goes into your text — followed by the punctuation you rolled



Speaker notes

1. **generate** from your model as before, rolling for each next word
2. when you land on a **punctuation token** (like `.` or `,`), pause---that's a tool call
3. **text the sentence so far** (everything since the last full stop) to 3 friends
4. **write down** the whole first reply, then the punctuation token you rolled
5. **continue** generating from the punctuation token

Worked example

your text so far is “the cat sat”, and the next dice roll gives you `.` as the next token — pause, that’s a tool call

text “the cat sat” (the whole sentence so far) to 3 friends; the first reply back might be “down by the river”

write `down by the river`, then the `.` you rolled anyway, and continue generating from `.`

Speaker notes

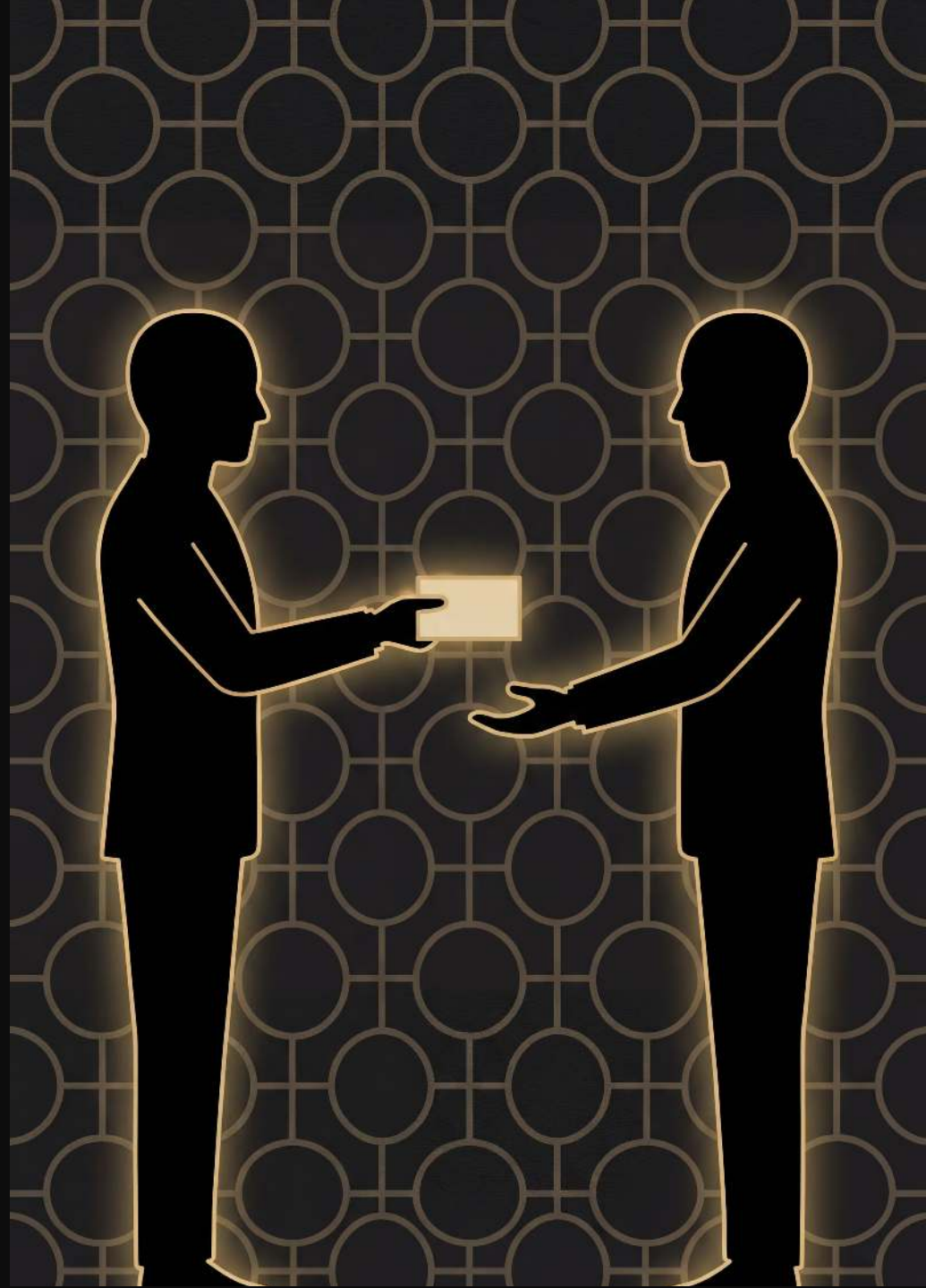
Walk it slowly: the punctuation is the trigger, the sentence so far is the tool input, the whole first reply is the tool result. The punctuation still gets written — the tool result slots in just before it. If asked why we don't continue from the friend's reply: those words probably aren't in your model's vocabulary, so resuming from the punctuation token is the cleanest way to keep generating.

You will need

your model, dice, pen and paper (as before)

a phone

3 friends who text back quickly



Speaker notes

Texting three friends is the retry logic --- first reply wins. If no-one replies before the next punctuation token, queue it up and back-fill. ~2 min set-up.

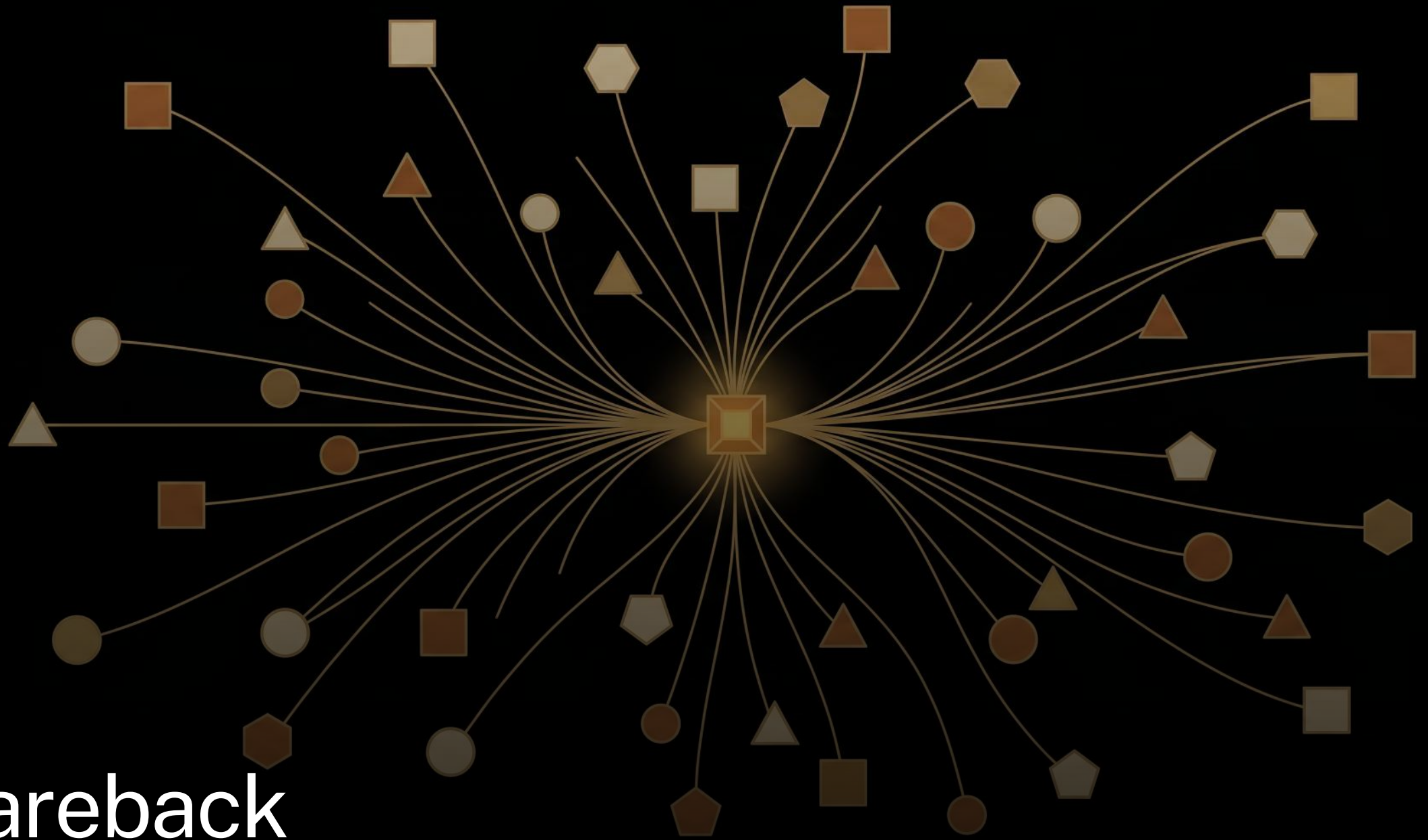
10:00

start reset +1 −1

Agentic AI

Speaker notes

Generate from your model --- on punctuation, text the sentence so far to 3 friends and splice in the whole first reply. 10 min --- texting latency is real, so keep groups moving; if a reply hasn't landed by the next punctuation token, back-fill it later. Circulate.



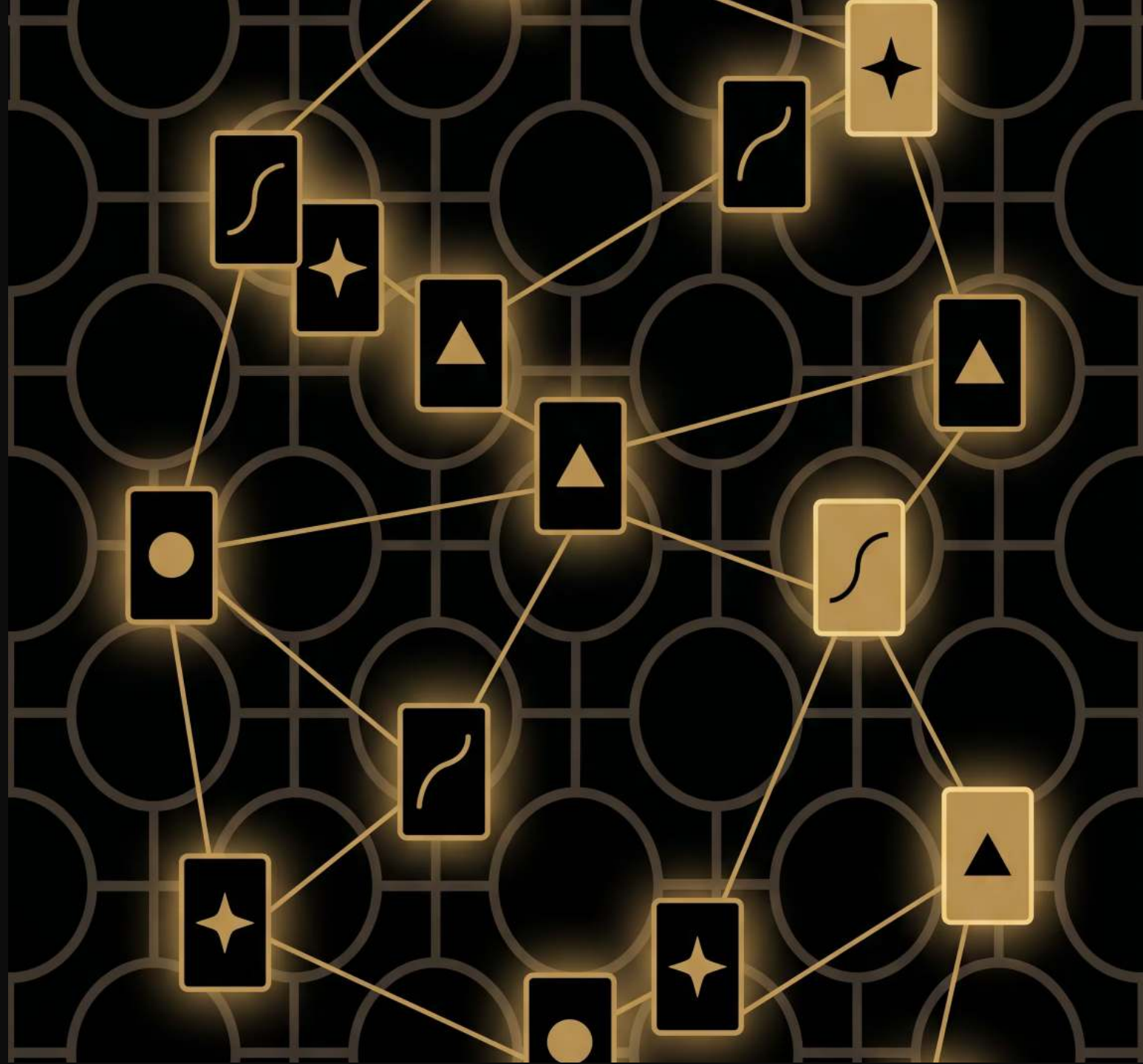
Shareback

Speaker notes

What did the tool calls add that the bare model couldn't --- fresh, outside information? Did the friends' words fit the text, or derail it? Invite ~5 groups to share back rather than the whole room. ~5 min.

The *language* of language models

- agent
- tool call
- agentic loop
- agentic AI



Speaker notes

Gloss each: agent = a model that can act, not just talk; tool call = it asks for something outside itself; agentic loop = generate, call, read the result, repeat; agentic AI = the umbrella buzzword for all of this---and now they know what's underneath it. If anyone asks what runs the loop: *you* did. The harness is the scaffolding around the model that pauses it, makes the call, and splices the result back---here, that was the students themselves.

Scaling up

Speaker notes

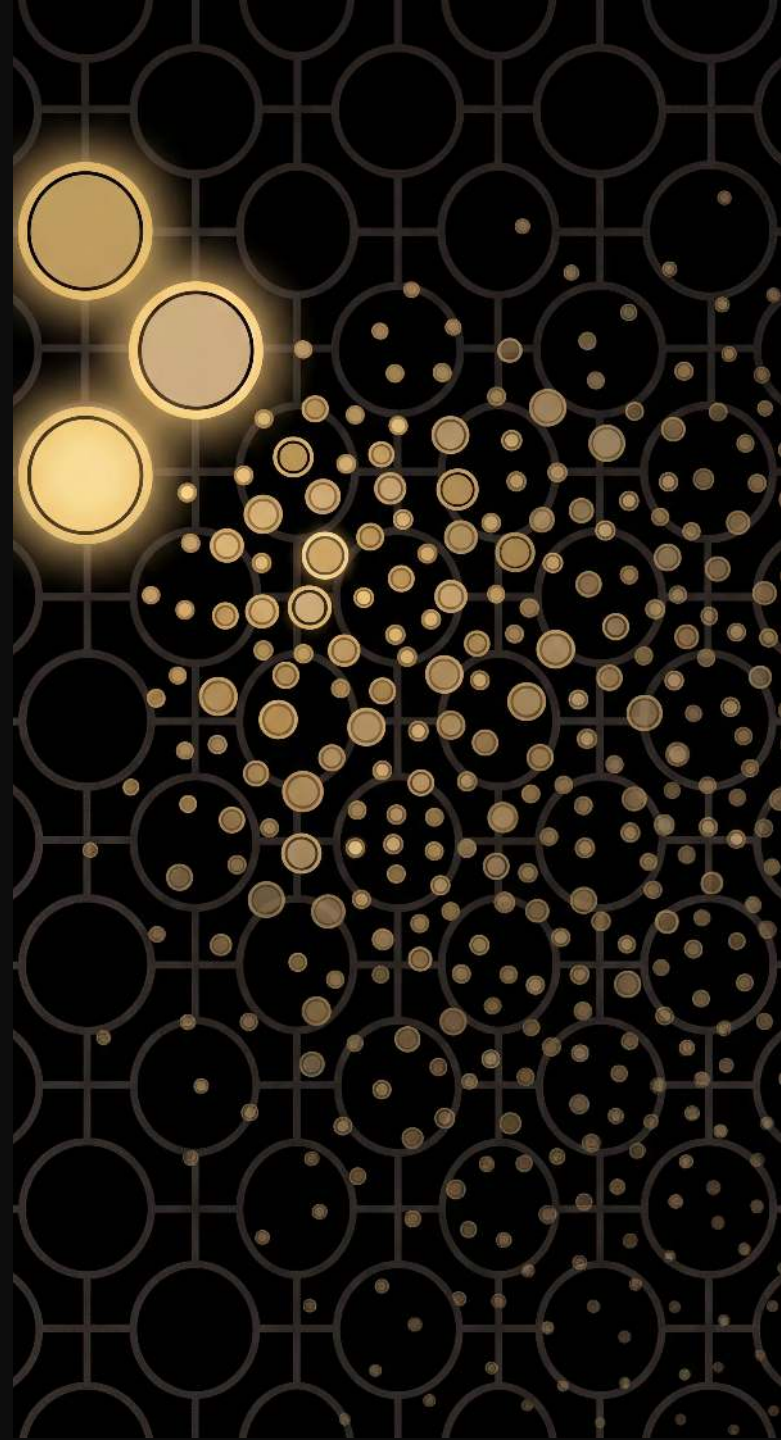
The reassurance beat: what they built isn't a cartoon of an LLM, it's the real mechanism at human scale---look at the context, weigh every possible next word, roll the dice, write it down, repeat. So what's actually different? More context, attention, billions of numbers instead of tallies, and a second round of training. Each is a diagram coming up. ~8 min, keep it moving.

More context

your grid `run` `,` `spot` `,` `run` `?`

Commercial LLMs *··· hundreds of pages before this ···* `run` `,` `spot` `,` `run` `?`

your grid sees **one word**; Claude sees **hundreds of thousands**



Speaker notes

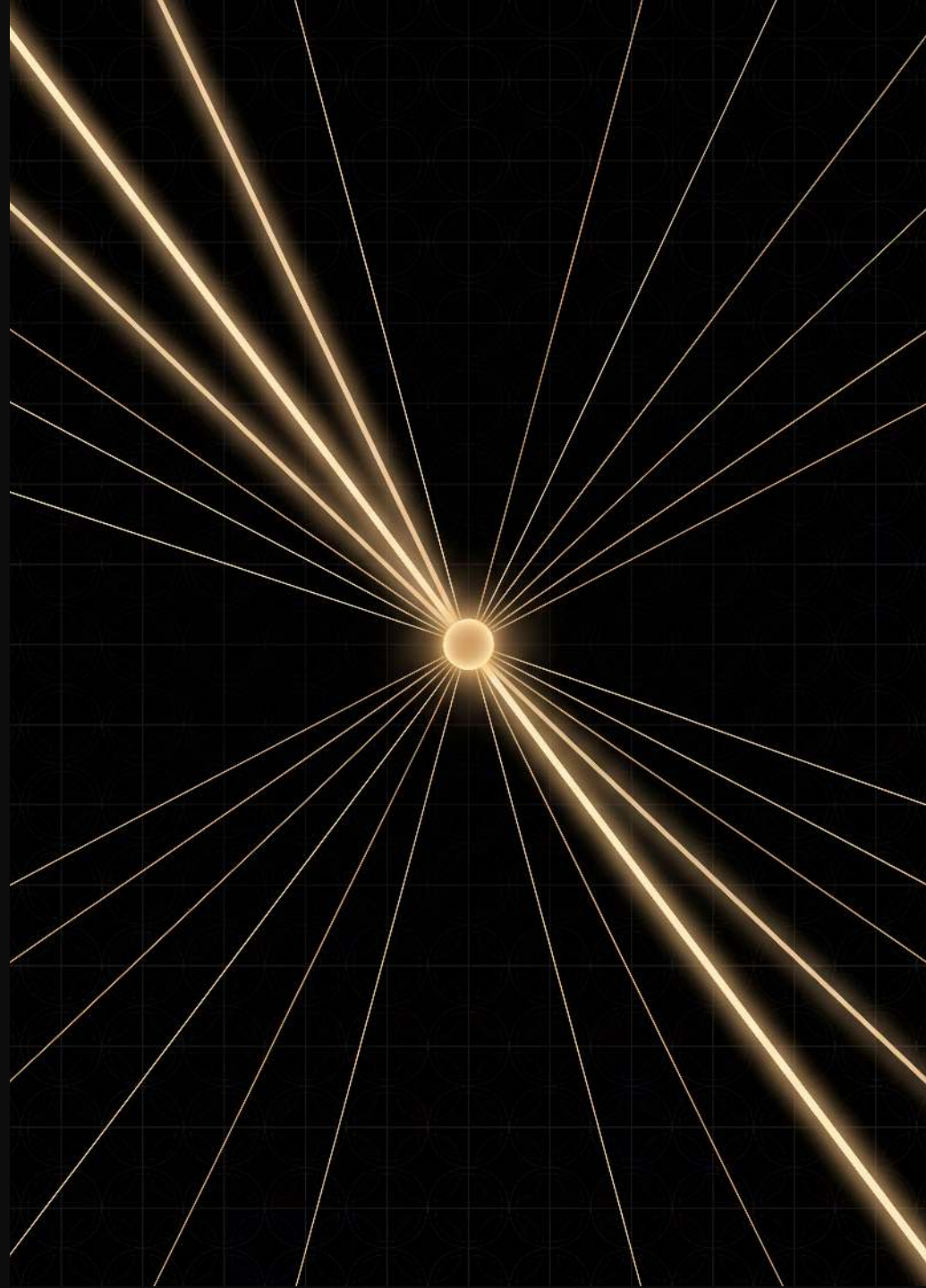
This is the "context window". One word of context is why their generated text wanders---it forgets where it was going one word ago. Real models hold a few novels' worth in view for every single prediction.

Attention: learning where to look

the wifi `password` is `swordfish` … the `password` is `?`

the next word is *swordfish* — attention finds where `password` last appeared and copies what came after

attention lets every earlier word vote on what comes next, and the model *learns* the weights — no grid could ever be that big



Speaker notes

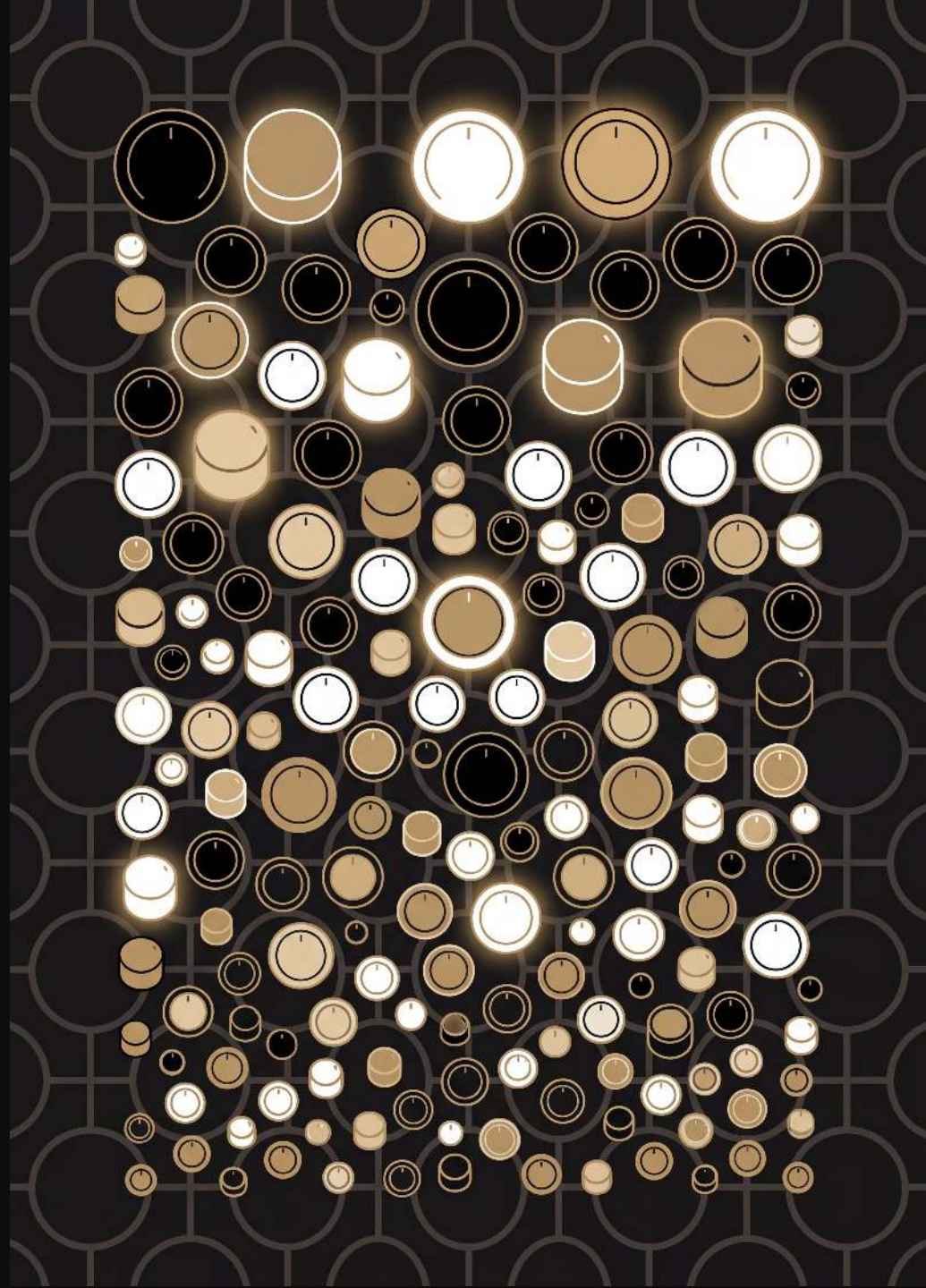
The transformer's central trick, and the answer to the blown-up grid: instead of a row for every possible context, each prediction looks back over the whole context and weighs which words matter. Here it's copying outright --- the model spots the earlier "the password is swordfish" and repeats what followed. That look-back-and-copy move is an induction head, and the Induction Heads lesson on the website is exactly this pattern --- "find the last time I saw this word, copy what came next" --- run with pen and paper.

Tallies become knobs

your grid `run` `.` `||||`

a transformer `0.31` `−1.20` `0.08` `0.94` `−0.55` `1.30` *× billions*

your grid stores each pattern as **tally marks you can count**; a transformer spreads the same patterns across **billions of tunable numbers** — knobs nudged during training, so similar words like `dog` and `cat` come to share them



Speaker notes

Your grid stores patterns as tally marks you can point to. A transformer spreads them across billions of numbers (its parameters), each nudged a tiny amount after every wrong guess in training. That spreading is why it generalises --- "dog" and "cat" end up sharing patterns. The cost: you can no longer point to where any one pattern lives.

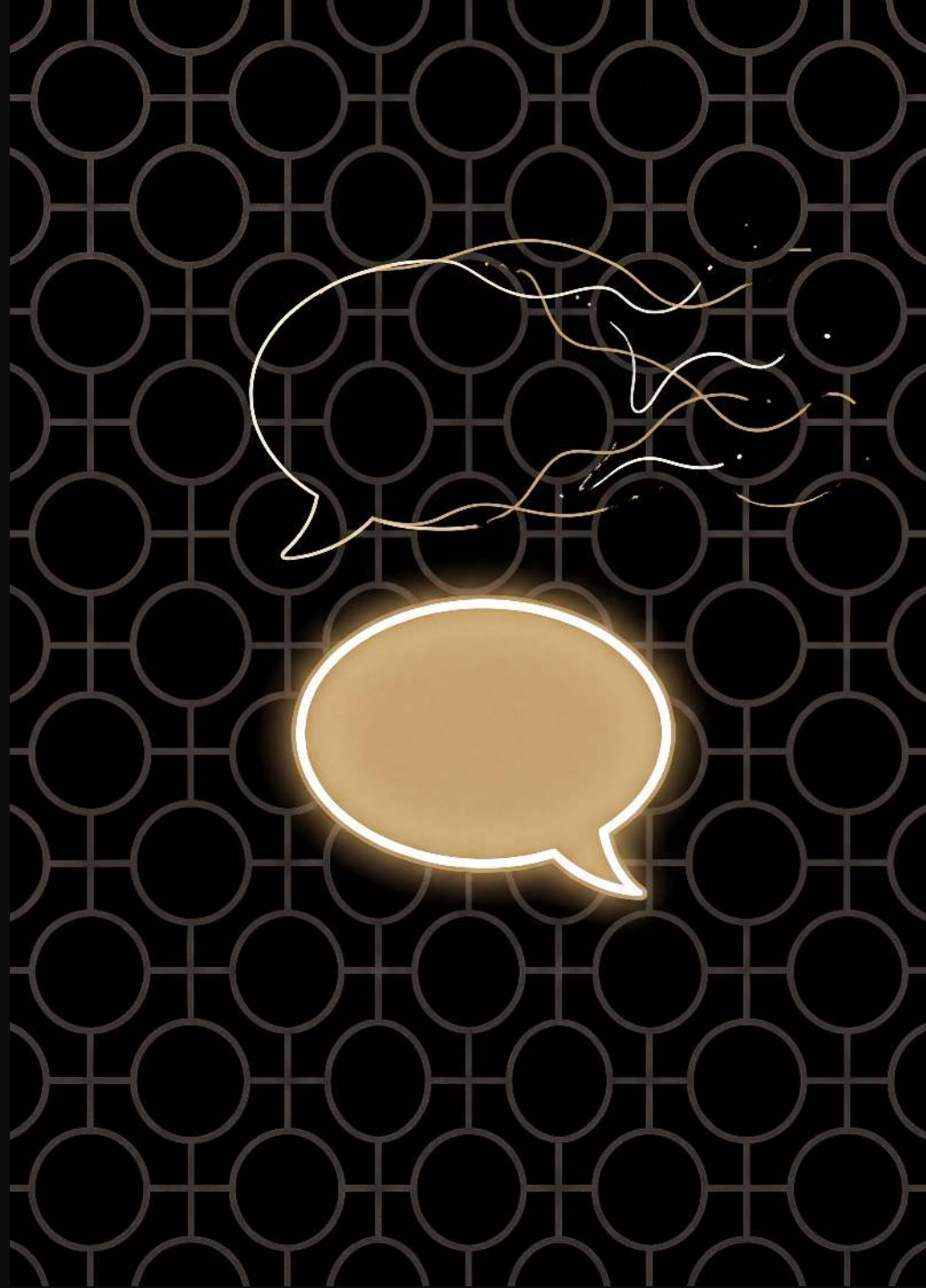
From continuing to answering

prompt: “what is the capital of France?”

a base model `“What is the capital of Germany? What is the largest…”`

+ post-training `“Paris.”`

everything you built only **continues** text — **post-training** on example conversations is what turns a *continuer* into an *assistant*



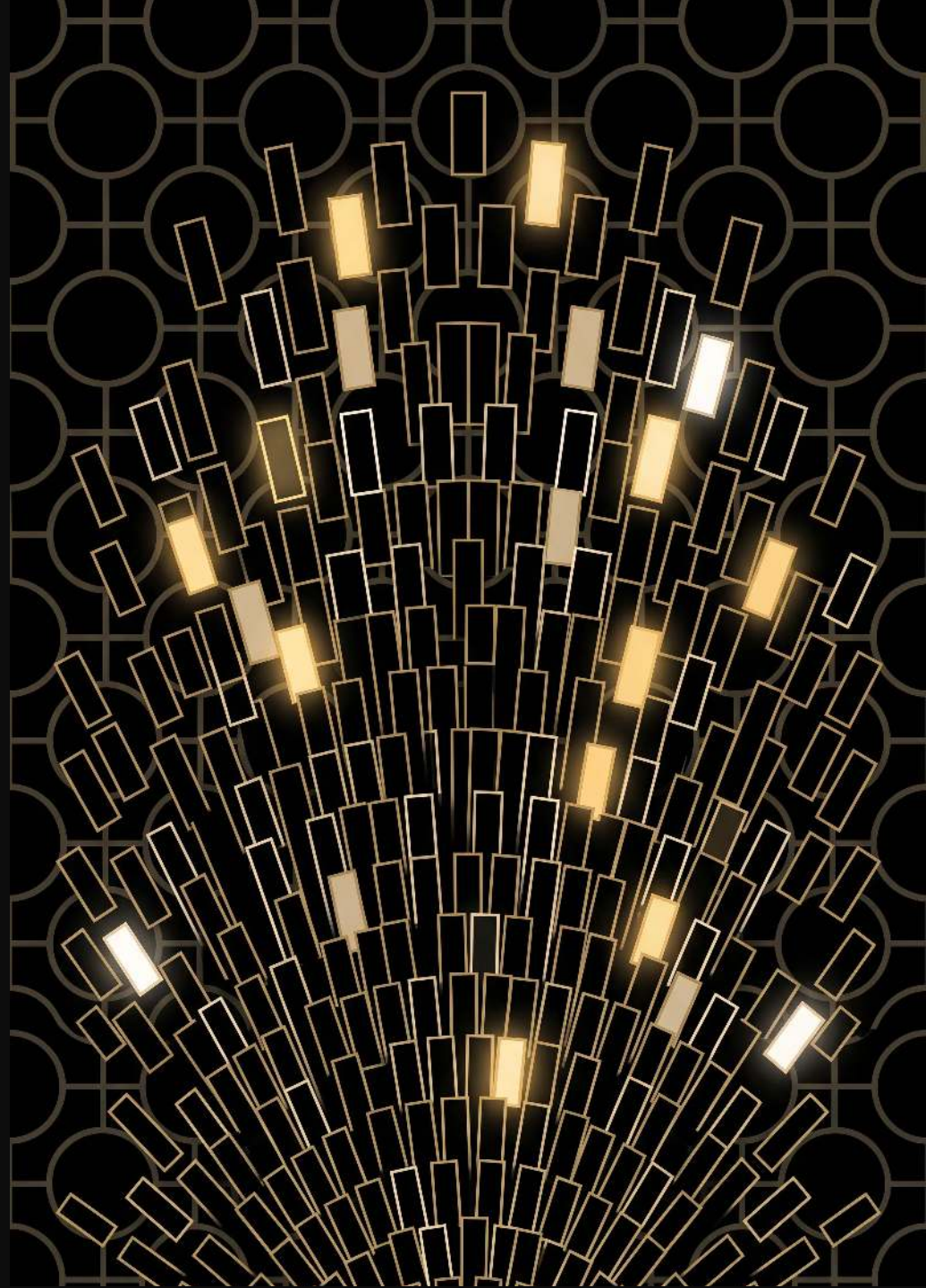
Speaker notes

A pretrained model is a text-continuer, not an assistant --- prompt it with a question and a plausible continuation is more questions. The chat behaviour comes from a second, much smaller phase of training on example conversations plus feedback on which answers people preferred. Same predict-the-next-word mechanism underneath; different data, different habits on top.

Scale is a hell of a drug

your model	trained on a few hundred words — a few dozen tallies
frontier LLM	trained on tens of trillions of words — hundreds of billions of numbers

the same loop you ran by hand — now with **billions of times** the text and **billions of times** the numbers



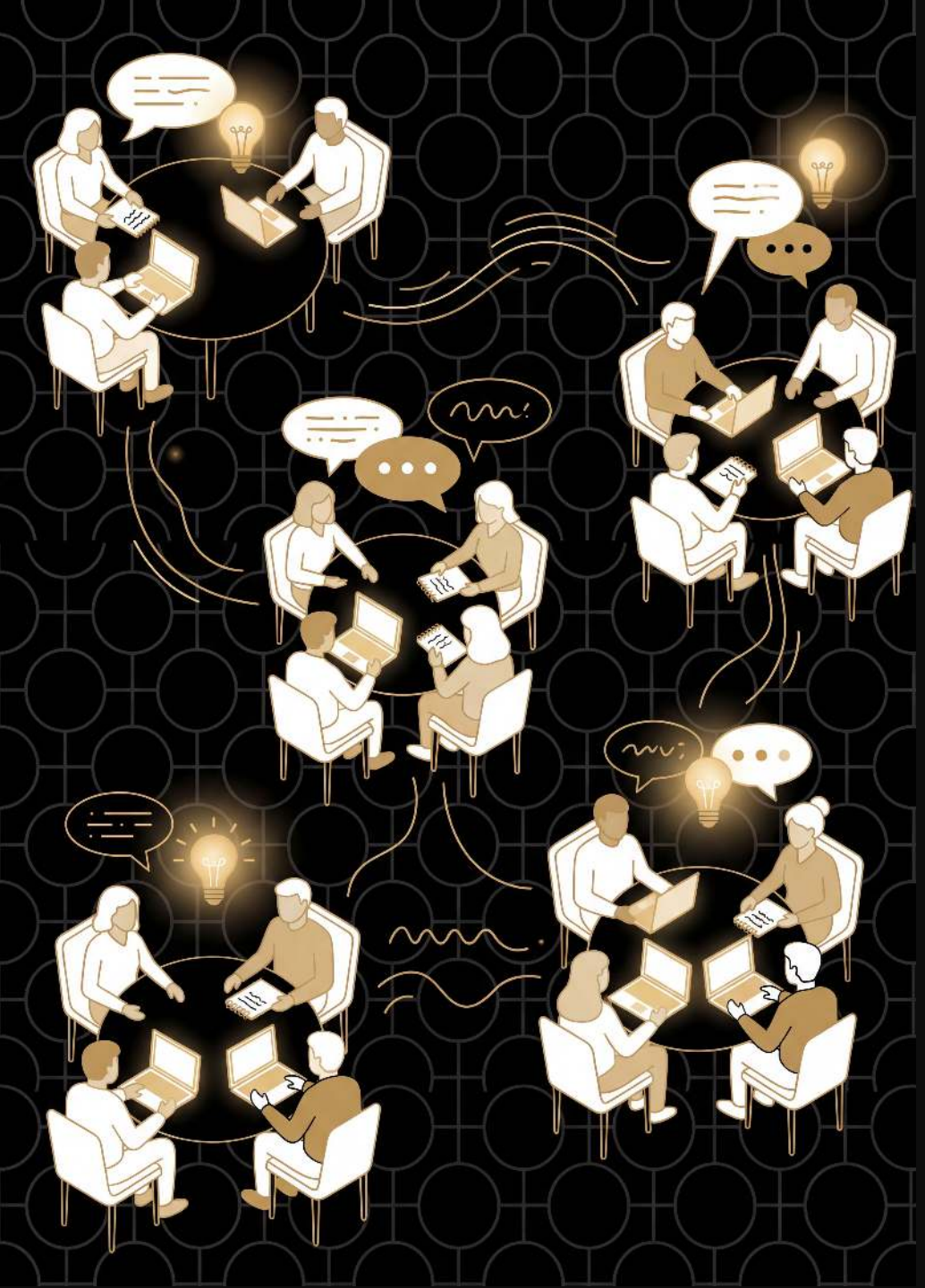
Speaker notes

Put the two scales side by side. Training data: a page or two of text versus tens of trillions of words (the open figures land around 30 trillion---Qwen3 cites ~36T, GLM-5.2 ~28.5T---but the number keeps climbing, so keep it as "tens of trillions"). Parameters: a few dozen tally marks versus hundreds of billions of tunable numbers (frontier dense models run into the hundreds of billions; the big mixture-of-experts ones past a trillion). Two ballparks to make it physical. Paper: at the booklet's 1cm^2 per parameter, a frontier model's ~1.5 trillion parameters would need a sheet of about 150 km^2---a square roughly 12 km on a side, about 100 ANU Acton campuses, or twenty-odd Lake Burley Griffins (your grid was a postcard). Time: if hand-training 100 tokens took you ~10 minutes, 30 trillion tokens at that rate is ~5.7 million years---you'd have had to start about when our ancestors split from the chimpanzees to be finishing now. The punchline isn't any one figure---it's that the mechanism didn't change, only the dosage.

still just **tokens in → tokens out**

Speaker notes

Every reply from Claude or ChatGPT is sampled one word at a time, exactly like their dice rolls --- that's why "regenerate" gives a different answer. Bigger context, learned where to look, billions of numbers, a politeness pass --- but the loop never changed.



Questions

what new questions do you have about large language models?

What next?

how will this change the way you think about and use LLMs in the future?

Australian National University

School of Cybernetics

www.llmsunplugged.org