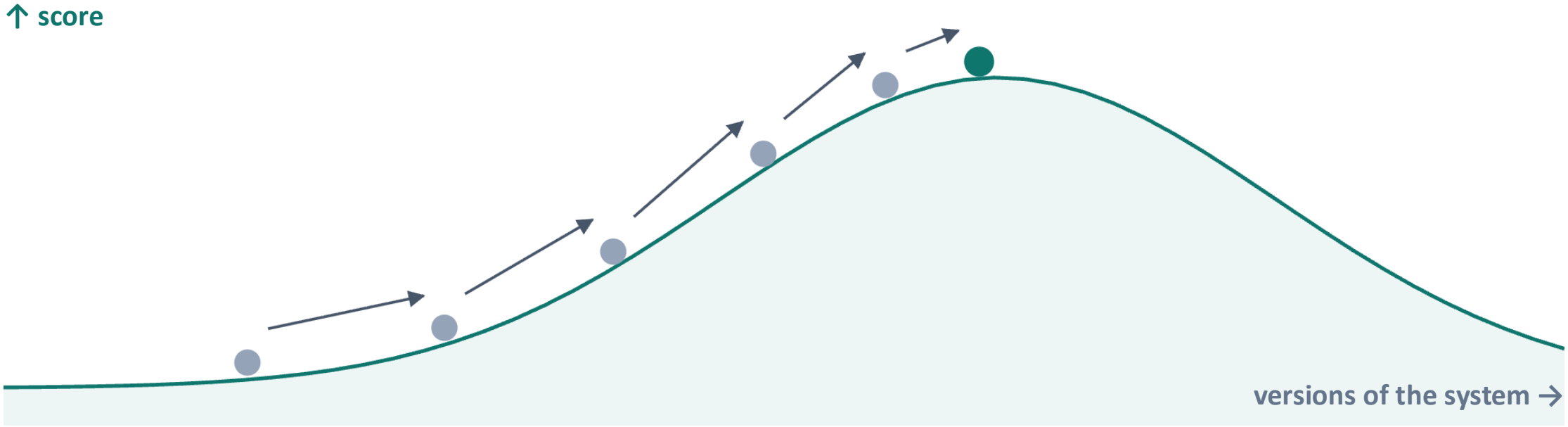
Hill-Climbing I

Climbing a prompt on your own data, and why the climb overfits so easily

Fabio Casati

Hill-climbing

Assess, Improve, Repeat.



Ideally, we climb in terms of metrics we care about on "future" production use cases

The question a climb must answer

Will this change help on new cases of the kind we care about, within the same budget?

CLAIM 1

Better on these cases

The cases we climbed on. Usually true, and weak: we chose the change for them.

CLAIM 2

Better on new cases of the same kind

Needs cases that never steered the climb.

CLAIM 3

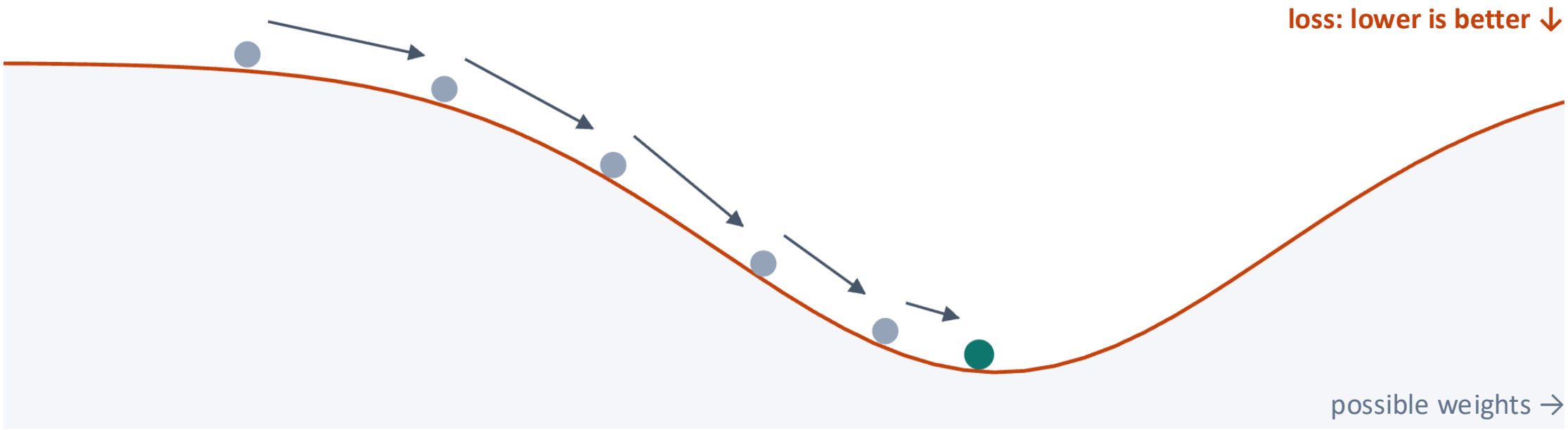
Better in production

Needs evidence from production, which drifts away from any frozen test.

Pick the claim before climbing: it decides what you must hold out.

Training a model is the same climb, downhill

A loss measures how wrong the model is. Each training step changes the weights a little, in the direction that lowers the loss.



Maximize a score or minimize a loss: it is the same search.
Again - we want the model to do well on "unseen" datasets (to generalize well), but what we have is our training / validation / test datasets

The five ingredients of a climb

1

The system we evolve

In ML: the model's weights

2

The data

In ML: training, validation and test sets

3

The measure

In ML: the loss function

4

The eval workflow and infrastructure

In ML: run a batch, compute the loss, pass on the gradient

5

The optimizer

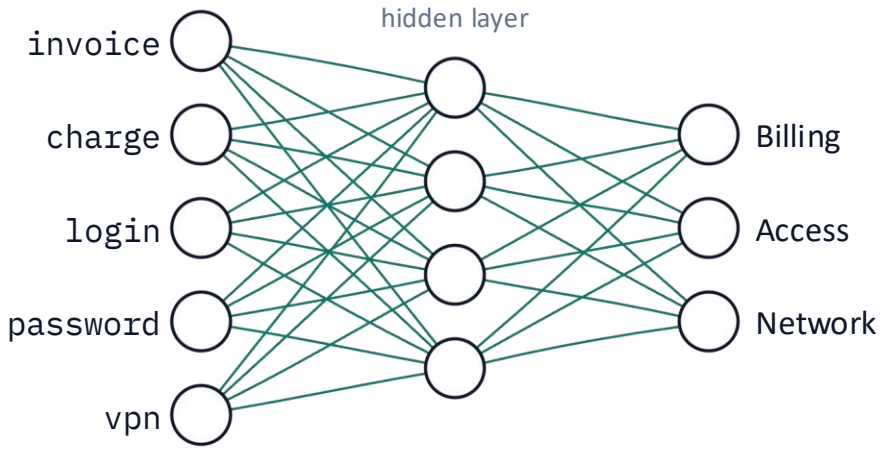
In ML: gradient descent

Each ingredient is a decision, and each one shapes what the final number means.

The classifier: the system and the data

1 The system: the model's parameters

Words in, a hidden layer, one probability per team out.



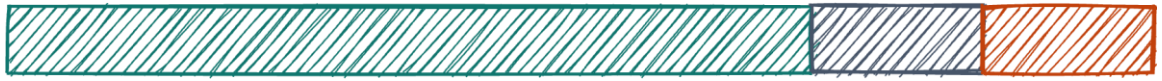
the weights: the numbers training changes

20,000 words, 64 hidden units: about 1.3 million weights.

x: the ticket's word counts, from the data →

2 The data: tickets with ground truth

ticket	x: word counts	ground truth
My invoice shows a double charge	`invoice: 1` `charge: 1`	Billing
Can't log in after the password reset	`login: 1` `password: 1`	Access
The VPN drops every ten minutes	`vpn: 1`	Network
Double charge on my card, please refund	`charge: 1`	Billing



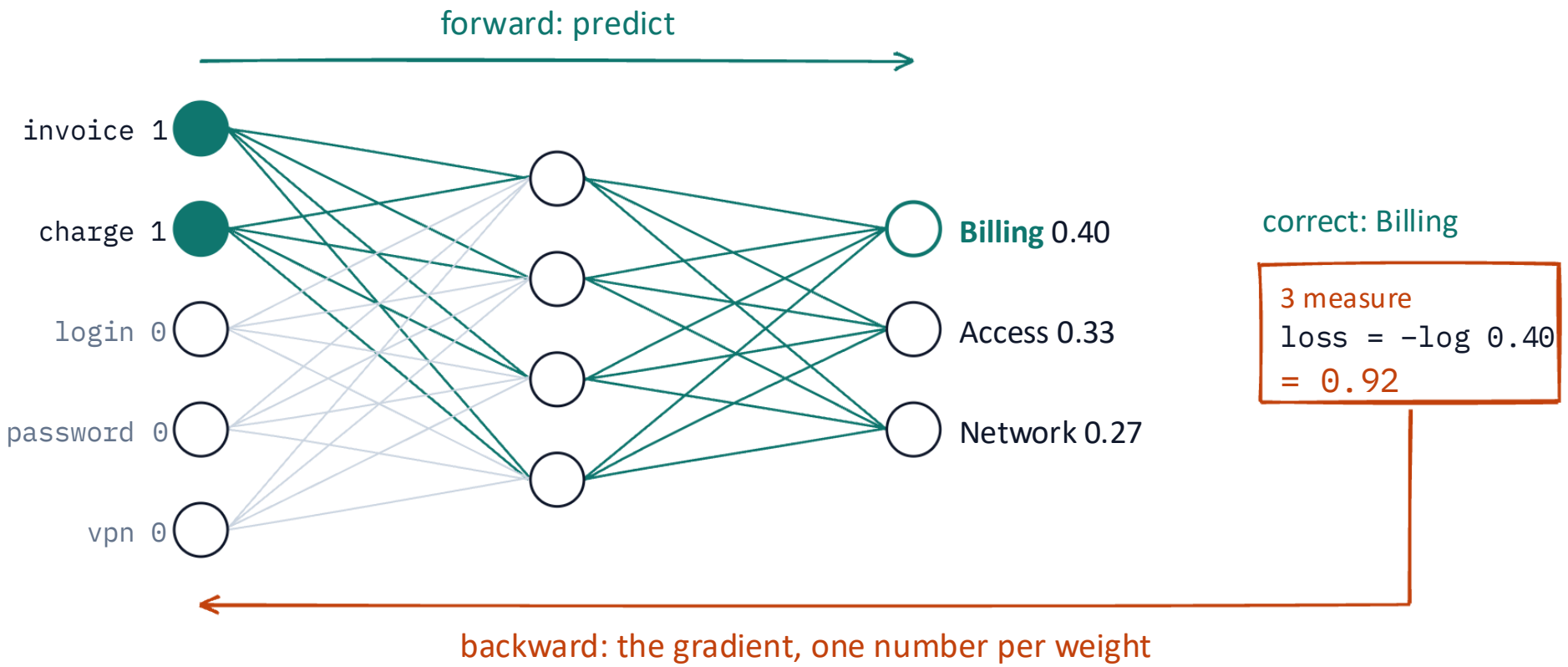
training validation test

35,000 tickets 7,500 7,500

Ground truth: the team that actually solved the ticket

Eval: the loss and the gradient

4 eval



"my invoice shows a double charge" →

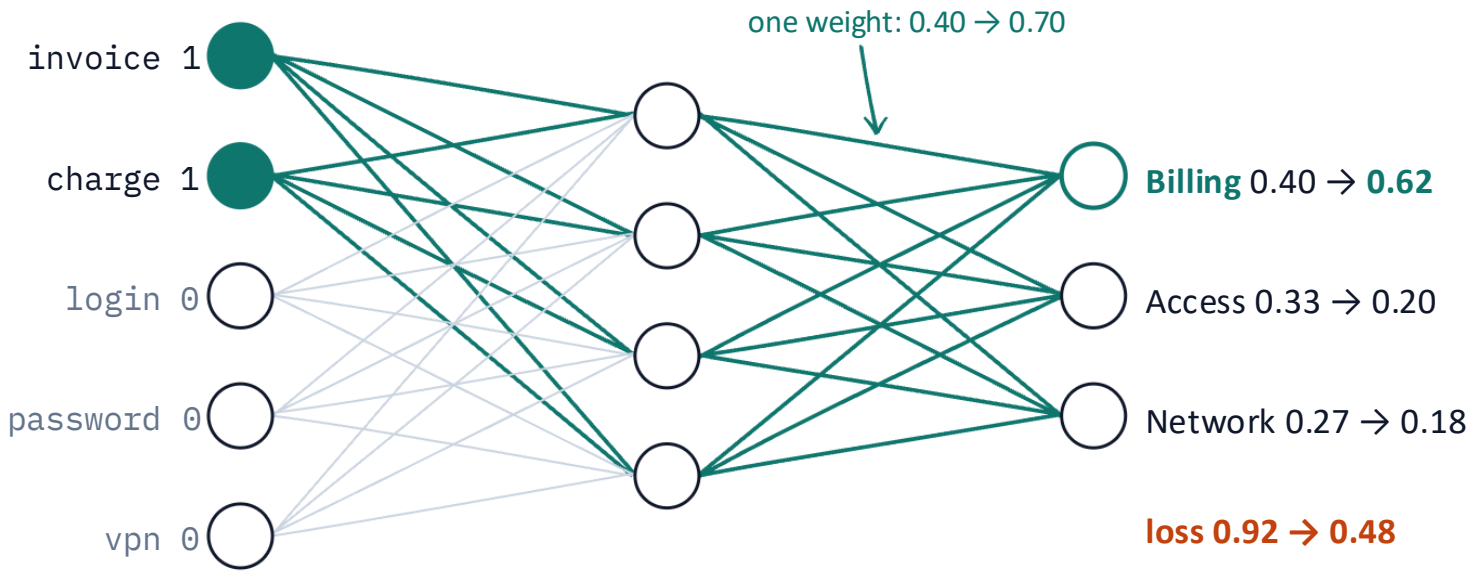
For a whole batch of tickets, averaged. The gradient is what goes on to the optimizer.

Optimizer: apply the gradient descent step

5 optimizer

every weight moves against its gradient

“my invoice shows a double charge” →



gradient descent

$w \leftarrow w - \eta \cdot \text{grad}$

$\eta = 0.5$, the learning rate

Then the next batch, over several passes through the data. The validation loss tells us when to stop.

How an LLM is pretrained

PRETRAINING

Step 1: download and preprocess the internet

https://huggingface.co/spaces/HuggingFaceFW/blogpost-fineweb-v1

The FineWeb pipeline

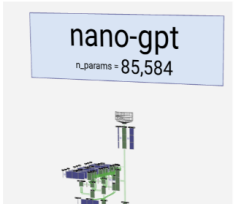
Step 2: tokenization

Converts text <---> sequences of symbols (/tokens)
- Start with stream of bytes (256 tokens)
- Run the Byte Pair Encoding algorithm (iteratively merge the most common token pair to mint new token)

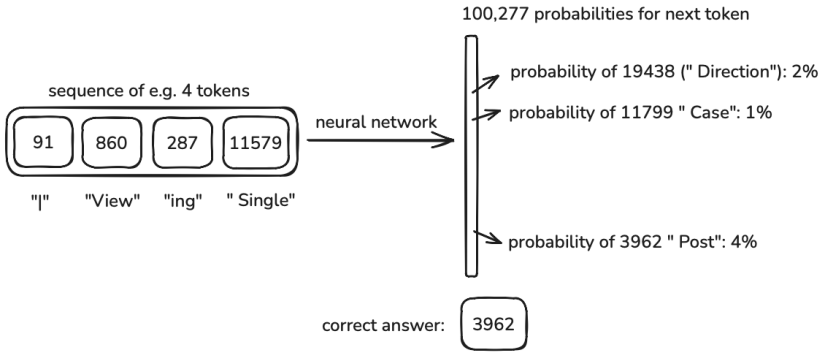
Example: ~5000 text characters
~= 40,000 bits (with vocabulary size of 2 tokens: bits 0/1)
~= 5000 bytes (with vocabulary size of 256 tokens: bytes)
~= 1300 GPT-4 tokens (vocabulary size 100,277)

https://tiktokenizer.vercel.app/

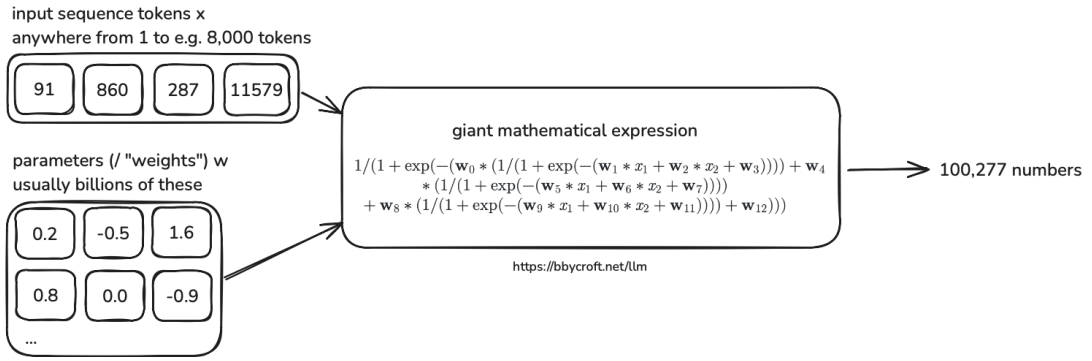
```
|Viewing Single Post From: Spoilers for the Week of Fe
bruary 11th|\n|Lil||Feb 1 2013, 09:58 AM|\nDon't care
about Chloe/Taniel/Jen-Jen. Don't care about Sami, rea
lly, but hoping that we get
```



Step 3: neural network training



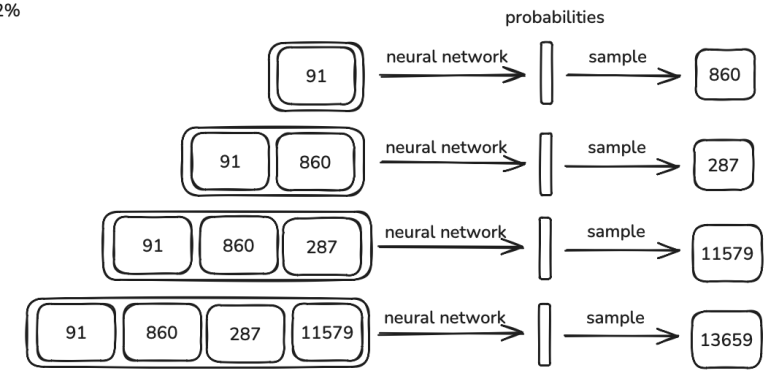
neural network internals



https://bbycroft.net/llm

Step 4: inference

to generate data, just predict one token at a time



Source: Andrej Karpathy, “How I use LLMs”, Excalidraw notes

Hill-climbing agents

What we improve: the model, the agent and its harness. And two ways to climb them.

What exactly are we hill-climbing for?

Agent?	The prompt: role, goals, rules, examples
Agent harness?	Context, planning, tools, memory, retries, stopping
Model?	Which model, which version, which settings

= **The AIS?**

the system under test

Two examples

EXAMPLE 1 · TODAY

Climb the prompt on our own data

Agent	**CLIMBED**
Agent harness	FIXED
Model	FIXED

Data: our own labelled support tickets

EXAMPLE 2 · HILL-CLIMBING II

Climb the harness on a benchmark

Agent	FIXED
Agent harness	**CLIMBED**
Model	FIXED

Data: a public benchmark, built by others

Example 1: hill-climbing the agent

We climb the prompt on our own tickets, one ingredient at a time: system, data, measure, eval workflow, optimizer.

Example 1: climbing a prompt

THE PROMPT, VERSION 1

```
You route support tickets.
Teams: Billing, Access, Network.
Reply with the team name only.

Ticket: {ticket}
```

The same model and the same harness run it on every ticket.

1	**System**	The prompt. Model and harness stay fixed.
2	**Data**	Our tickets: training to climb, validation to pick a style, test to report
3	**Measure**	Accuracy: the share routed to the right team
4	**Eval workflow**	Run every ticket, compare with the ground truth, keep the errors
5	**Optimizer**	Us, or an LLM, reading the errors and editing the prompt

Only the prompt changes, so a real change in the score is the prompt's; a small one may be noise.

System: the prompt, and what stays fixed

CLIMBED

The prompt

Role and task

A definition of each team

Labelled examples

Output format: the team name only

HELD FIXED

Everything else

Model: one pinned version

Settings: temperature 0, a token limit

Harness: one call per ticket, read the answer

Budget: cost and time per ticket

Every prompt gets a version number and is tracked so the experiment is repeatable

Data: choosing the tickets

Source	Past tickets, labelled with the team that actually solved them
Coverage	All three teams, recent months, short and long tickets, the hard cases
Labels	Check a sample by hand: was the solving team the right one?
Size	Training 200+ to climb; validation 100+ to pick a style; test 100+ once
Split	can be plain or stratified. Test from the latest month

Choose and split the data before the first prompt is written.

Measure: choosing the metric

HEADLINE

Accuracy

The share of tickets routed to the right team. Simple, but it hides which team pays for the errors.

PER TEAM

Recall per team

If Network tickets are rare, high accuracy can hide half of them going astray.

CONSTRAINTS

Cost and latency

Tokens and seconds per ticket. A longer prompt can buy accuracy with money.

Resolution: near 80% accuracy, a score on 200 tickets is uncertain by about ±5.5 points; on 100 tickets, by about ±8.

Fix the metric before climbing.

Eval workflow: what runs, what is kept

1	**Run**	Every training ticket, with the same model and settings
2	**Read the answer**	Not a team name? It counts as wrong, and it is logged
3	**Compare**	With the ground truth, ticket by ticket
4	**Keep**	Accuracy, a confusion table, and the misrouted training tickets
5	**Log**	Prompt version, scores, cost and date, for rejected versions too

Training errors are read; validation gives only a score; test stays locked.

Optimizer: who edits the prompt

A PERSON

Reads the errors, rewrites the prompt

Spots patterns: Access and Network confused, vague tickets, answers that are not a team

→ sees the training errors, nothing else

AN LLM

Proposes prompts from the errors

Given the prompt and the misrouted tickets, it writes candidates; each is scored on the training set

→ sees the same, and only that

Like gradient descent, it can memorise; unlike it, it can copy a ticket into the prompt in one step. We could even literally copy all 200 training tickets in the prompt - would that help?

One step of the climb

1 · A TRAINING MISS

“VPN won’t connect since my password reset.”

Router: **Access**
Label: **Network**

2 · ONE EDIT

```
+ Connection problems, including VPN, go to Network even if a password is mentioned.
```

One line added to the prompt: a pattern, not a patch for this ticket.

3 · WHAT FLIPPED

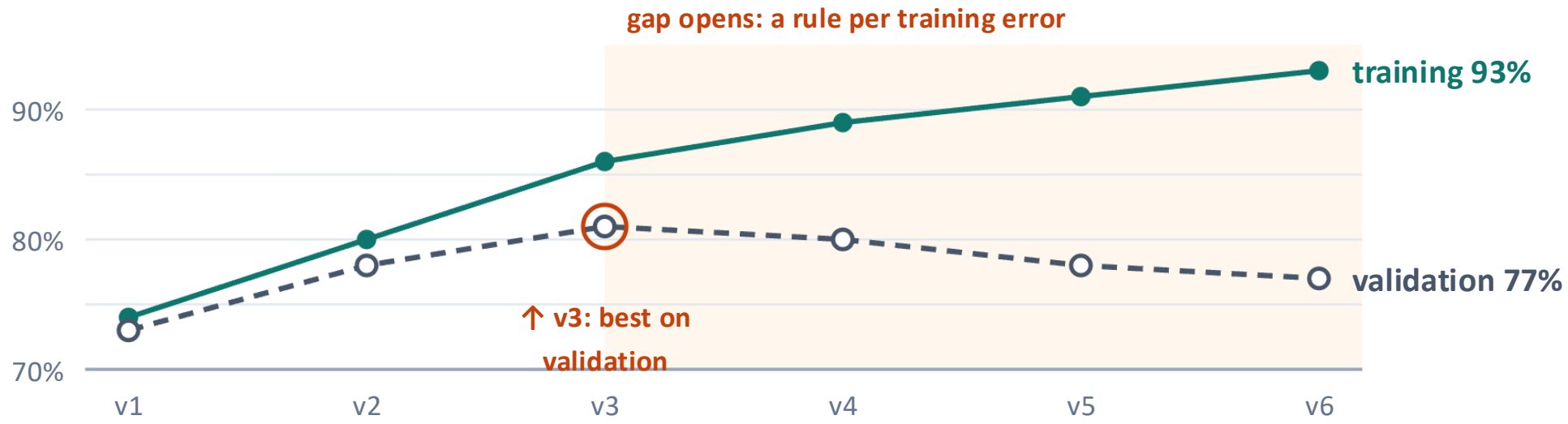
The VPN ticket: Access → **Network** ✓

“My password expired and the VPN keeps asking for it.”

Access → **Network** ✗
Label: **Access**

On all 200 training tickets: 9 fixed, 4 broken. Read the flips, not just the +2.5.

Validation: when to stop, which version to keep



Gap opens → go back to v3

Keep the version validation scored best.

Strategy widens the gap → change it

Stop patching single tickets; try examples instead.

Scores only, a few looks, and the rule fixed before the climb.

Three ways to use the validation set

1 · PICK

Strategies fixed in advance

Climb each on training, score each once on validation, pick by a written rule.

Cost: the winner looks about 3 points better with 3 strategies, about 7 with 20.

Best

2 · STEER

Scores at checkpoints

Look at the score a few times; roll back or switch strategy.

Cost: every look fits validation a little.

Fine, with the rule fixed first

3 · READ

Its misrouted tickets

Use the validation errors to decide what to change.

Cost: validation becomes training data; its score stops checking.

Avoid: read training errors instead

The more you learn from validation, the less it can tell you.

Example 1: one climb, style by style

Prompt style, climbed on the training set	Training (200)	Validation (100)
A · Team definitions and general rules	86%	81%
A′ · A, plus a rule per training error: “double charge” → Billing, “Acme” → Network, …	93%	77%
B · Definitions plus six labelled training examples	87%	86%
C · Reason step by step, then answer	89%	84%

Final test, planned in advance: v1 75%, B 85%, on the same 100 tickets. B fixes 13 and breaks 3: +10 points (+2 to +18).

Training errors drove the edits; validation picked the style.

Good habits for a climb

One change per step

Instructions, examples and format in separate steps: we know which one helped.

Same input as production

Keep or strip signatures and quoted threads as production will. A quoted invoice thread can pull a VPN ticket to Billing.

Read what flipped

Compare answers ticket by ticket between versions, not only the score.

Keep every version

Every prompt tried, kept or dropped, so we can roll back and count the tries.

Small, recorded steps on a fixed setup.

What can go wrong

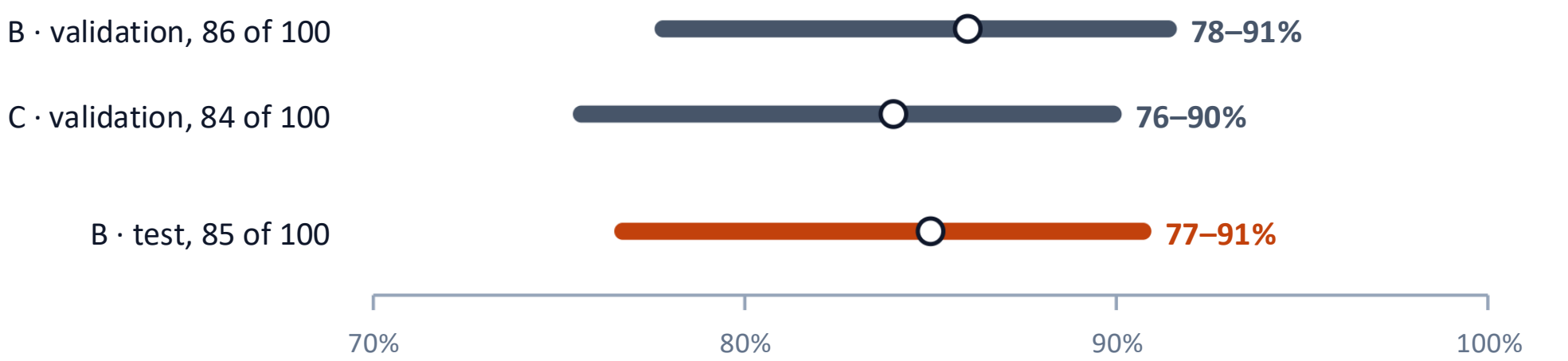
The causes of a wrong conclusion, from Part II, and how each shows up in a prompt climb.

What can lead us to the wrong conclusion

1	**Small dataset size**	Sample size
2	**Testing many hypotheses on the same data**	Closed-book overfitting
3	**Looking at the cases we evaluate on**	Open-book overfitting
4	**Other creative and misunderstood forms of test data leakage**	here
5	**Wrong labels or a biased judge**	Noisy and biased judges
6	**Data unlike the real use**	Representativeness
7	**Metric vs goal discrepancy**	Goal mismatch
8	**Fragile setup choices**	Other brittleness

And they compound rather than cancel.

Too few tickets for the decision



REPORT THE BAND

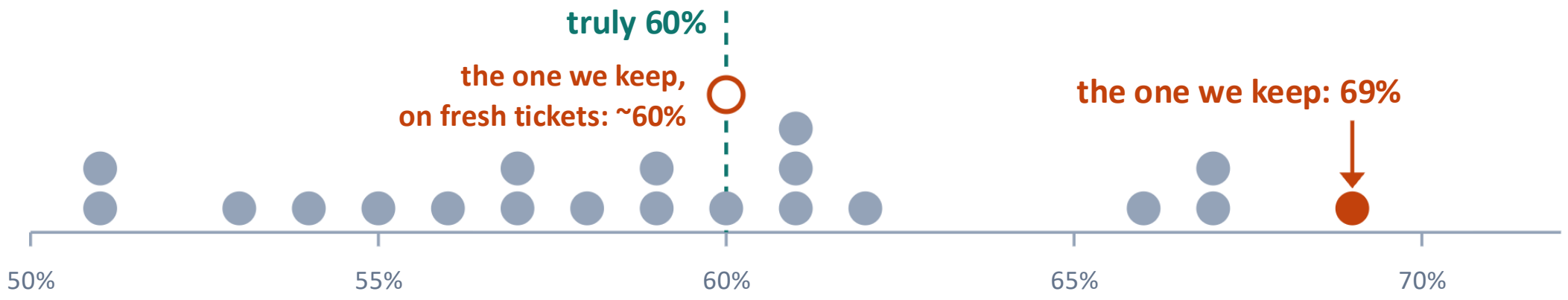
“85%, most likely between 77% and 91%,” not “85%.”

SIZE FOR THE DECISION

Separating 86% from 84% takes about 2,500 tickets per prompt; running both on the same tickets can cut that several-fold.

Scores alone are enough to overfit

20 prompt variants, all equally good (60%), each scored on the same 100 validation tickets



Mechanism: adaptive overfitting. Simulated; the 20 scores are assumed independent.

Open the Inflated Estimates playground

Choosing after the fact

PICK THE METRIC

B loses on accuracy, wins on Network recall. The report leads with recall.

PICK THE SET

93% on training, 85% on test. The slide shows 93%.

PICK THE SLICE

Flat overall, up 9 points on email tickets. That becomes the headline.

Fix: write the metric, the set, the slice and the decision rule before the first run. Explore later, and label it exploration.

Ask enough questions of one set, and one answer looks good by chance.

Closed-book overfitting on the test set

MANY FINALISTS, ONE TEST

B, B2 and B3 all scored on test

We ship the best. Its 85% is now a best-of-3, about 3 points optimistic.

TEST AFTER EVERY FIX

Fix, re-test, fix again

After five rounds the test has steered five choices: it is a validation set now.

UNINTENDED ACCESS

Nobody decided to look

Per-team test scores on a dashboard steer the next edit; or last quarter's climb already chose on this test.

Fix: one look at the test, and no test results anywhere before it. A second look needs fresh tickets; report how many looks there were.

A test set used to choose is no longer a test set.

Reading the misses: on which set?

TRAINING

How we climb

Reading the misses is the point. The question is whether an edit fits a pattern or one ticket.

Possibly overfitting: only validation can tell

VALIDATION

It turns into training data

Its misses steer the next edit, and its score stops being an independent check.

Its score now flatters the prompt: replace it with fresh tickets

TEST

The test is gone

A test that steered a change can no longer confirm it.

Get a new test set

Reading the misses is how we climb, and only on training.

What patching looks like on training

An AI optimizer writes one rule for each missed ticket it reads, round after round.

Prompt	What happens	Tickets it read	Held-out tickets
V1	A short, general prompt: the task, the three teams, answer with one team name. 19 words.	85%	83%
V2	For each training ticket it got wrong, one new rule (“Acme → Network”). 378 words.	91%	77%
V3	More rules to fix what the first rules broke, plus examples copied from training tickets with the numbers changed. 695 words.	100%	80%

WARNING SIGNS

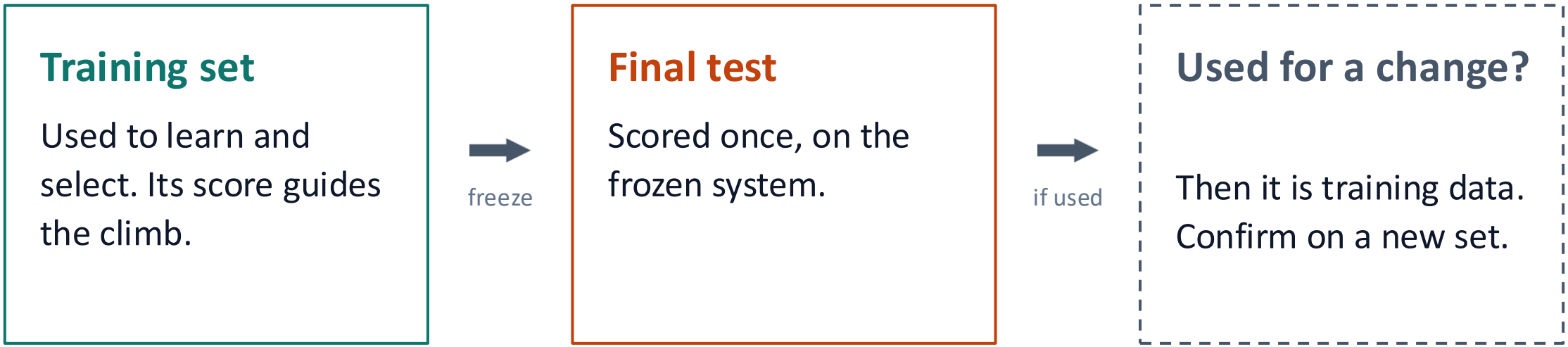
The prompt keeps growing, and more and more examples get added.

INSTEAD

Patch patterns when possible, explain example behavior rather than just add the category. Check on validation at planned checkpoints.

Open-Book Overfitting playground

The training score is not evidence of generalization



Mechanism: adaptive overfitting, through a reused test set

Once a result drives a change, it is training data.

How test content leaks into a prompt climb

Paraphrases	Test misses rewritten as new training examples
Copies	A test ticket pasted in as a prompt example
Paraphrasing test data	An LLM shown test cases, asked for “more tickets like these”
Unintended optimizer access to test data	Test file in its working folder; test results in its eval feedback; it runs the eval that also scores the test

Noisy labels and a lenient parser

NOISE: ERRORS IN NO DIRECTION

Some labels are wrong

“The team that solved it” is not always the right team. With 10% wrong labels, a router that is 80% right measures about 73%.

→ Audit a sample of labels; add a tie-break rule.

BIAS: ERRORS IN ONE DIRECTION

A lenient parser

If any answer that names the right team counts, “Billing or Access” scores. The climb keeps the hedging prompt.

→ Strict parser; count unparseable answers; never change prompt and parser together.

Judges & Compounding Lab: see what judge noise and bias do to a score

The ±8 points cover sampling only, not wrong labels or a lenient parser.

Wrong labels in training, validation and test

SIMULATED: 10% WRONG LABELS, IN ONE SET AT A TIME · 200 CLIMBS

TRAINING

The climb learns them

A miss that is a label error becomes a rule. The training score still ends at 95%.

+11, not +17

real gain of the climb; +6 with 20% wrong

→ Before patching a miss: router or label? Relabel.

VALIDATION

The check misreads

The climb is untouched, but the score sinks and every difference shrinks.

11-point gap

training 96%, validation 85%; only 2 points are overfitting

→ Label validation twice; settle disagreements.

TEST

The number moves

A router that is 85% right reports 77%, and the ±8 band does not stretch to cover it.

37%, not 95%

how often the band contains the truth; 2% with 20% wrong

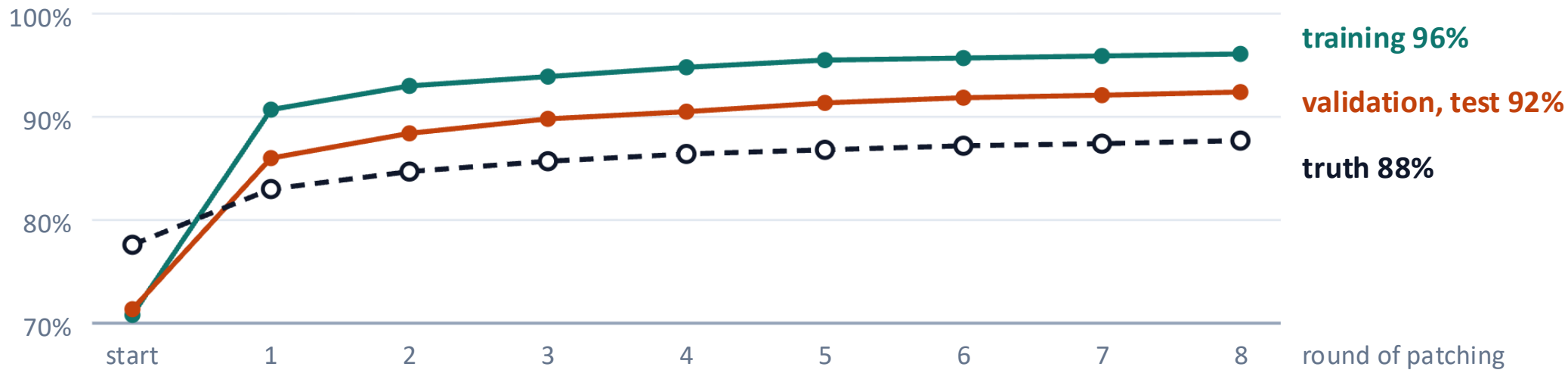
→ Audit a sample; report the error rate with the score.

The same wrong labels, three different failures.

Climbing on Noisy Labels playground: wrong labels in one set at a time

One labeler for all three sets

Simulated, 200 climbs: one labeler sends 10% of the traffic to the same wrong team, every time, in all three sets.



Every score says

Validation up 21 points; the test reports 92%.

The truth

Up 10 points; 88% right on new tickets.

Fix

A second labeler for validation and test; audit by kind of ticket.

A shared habit passes every check, and pushes the number up.

Tickets unlike next month's

HIDDEN SPREAD

One number, many slices

86% overall can be 95% on Billing and 60% on Network, or weaker in Spanish.

Score each team, channel and language, with its band.

FEW SOURCES

Three customers, three data points

Tickets from three customers say little about a fourth.

Test many customers, not many tickets from one.

DRIFT

The mix moves

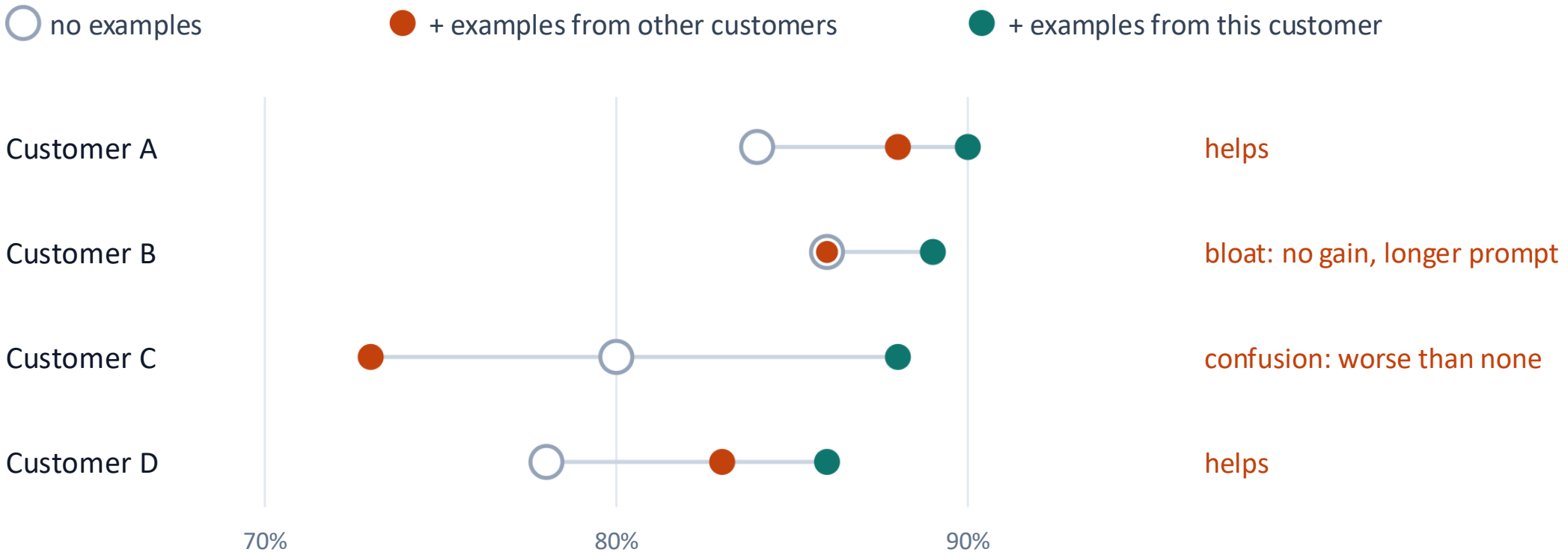
New products and new words change the tickets; last month's test ages.

Test on the latest month; re-test when the mix changes.

Hidden slices: find weak slices by reading training misses; confirm them on validation before acting.

A claim needs a scope: whose tickets, which languages, collected when.

One prompt for every customer?



The spread across customers says how much the prompt must adapt.

Metric / goal mismatch

WHAT WE MEASURE

Accuracy against the team that solved the ticket

≠

WHAT WE WANT

Each ticket reaches someone who can fix it, fast

Unequal costs

A misrouted outage costs more than a misrouted invoice; accuracy counts both as one.

A proxy label

Reroutes and delays never show in the label.

The climb exploits it

Gains on common Billing tickets can hide losses on rare Network ones.

Fix: choose the metric from the goal, before climbing, and add guardrails: per-team recall, cost per ticket.

Reward the wrong thing, and we climb the wrong hill, efficiently.

Small things that flip the result

Model version

The provider updates the model behind the same name.

Shared origins

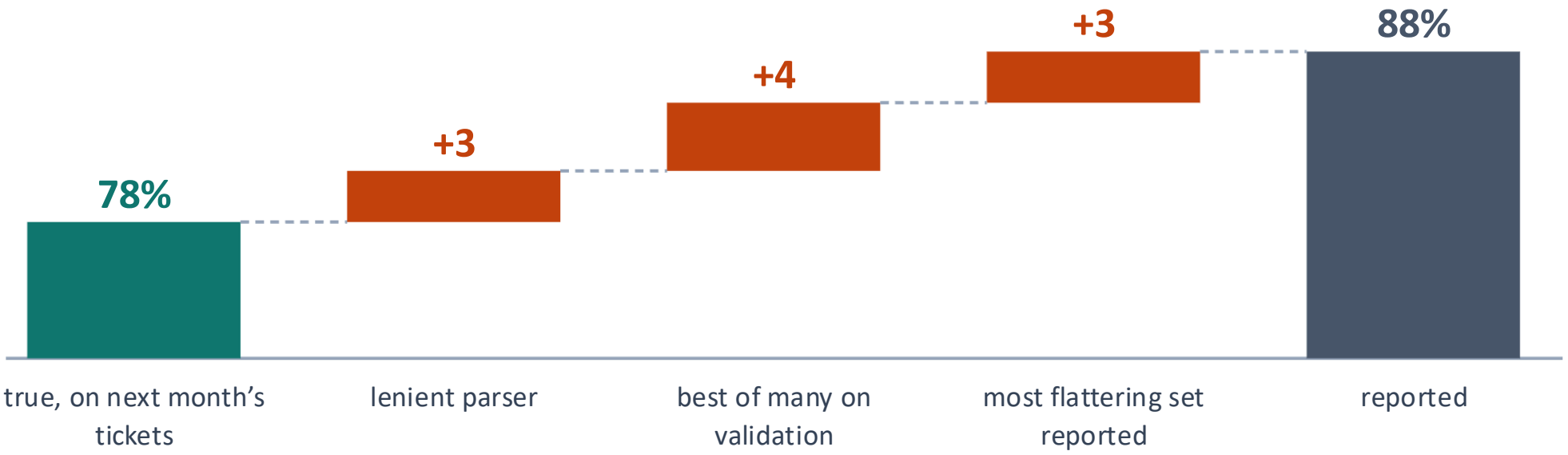
Templated tickets, like monitoring alerts, can be a large share. In a small test set, how many of them land there swings the score.

Reruns

Temperature 0, and some answers still flip on a rerun.

Fix them before the climb, and record them.

They add up, they don’t cancel



Illustrative numbers. How Problems Compound playground

Errors that all flatter the prompt add up; they do not cancel.

Three mechanisms to keep distinct

Adaptive overfitting

Repeated selection on feedback. No test items needed.

In our router: twenty prompts compared on the same 100 validation tickets; the best kept.

Control: log every round; confirm on fresh cases.

Contamination or leakage

Test information enters development, training, or memory.

In our router: test misses paraphrased into new training examples.

Control: keep test content and results away from people and optimizers.

Verifier exploitation

Credit without meeting the objective. Needs no leakage.

In our router: a lenient parser accepts “Billing or Access” for Billing.

Control: audit what passes; make the checks strict.

Hill-climbing can select verifier exploits without anyone intending to.

What to write down

BEFORE THE CLIMB	
Claim	Which claim, and the metric and guardrails that test it
System	What changes, what stays fixed: model version, input text
Data	Source, splits and sizes, made before looking
Who sees what	Training errors read; validation as scores only; test locked
Checks	Who labeled each set; label audit; strict parser

DURING AND AFTER	
Versions tried	Every prompt and style, kept or not
Selection rule	How validation chooses, fixed in advance
Final test	Frozen prompt and baseline, once
Result	Gain over the baseline, its band, label error rate, limits
Slices	Score per team, channel and customer, each with its band

Write the left column before the first prompt; the right one as you go.

Thank you

Join us for Hill-Climbing II

Climbing a harness on a benchmark, and how the benchmark gets into the improvement

Appendix

Where each cause strikes, on our training, validation and test sets.

Where it strikes: overfitting and leakage

Issue	Training (200)	Validation (100)	Test (100)
Open-book overfitting	By design: we read misses and patch them. Fine, unless we report it.	Read its misses and it turns into training data.	Read its misses and fix them: it can no longer confirm the fix.
Closed-book overfitting	Not a risk: we fit it on purpose.	Best of many strategies, checkpoints, metrics or slices.	Several finalists, re-tests after fixes, or test scores on a dashboard.
Test leakage	Paraphrased test misses added as training examples.	Validation tickets used as prompt examples.	Test tickets as examples; an optimizer that can open the test file.

The closer to the test set, the more it costs.

Where it strikes: the other issues

Issue	Training (200)	Validation (100)	Test (100)
Small sample	±5.5 points: a 2-point gain may be noise.	±8 points: B and C can't be told apart.	The gain is a band: +2 to +18.
Labels and parser	Patching a wrong label writes it into the prompt.	Wrong labels fake a gap; a lenient parser favors hedging.	Wrong labels move the number; its band ignores them.
Representativeness	Weak slices are spotted here.	Slices too small to confirm much.	Latest month only; it ages after launch.
Metric vs goal	We climb the metric, not the goal.	It picks the prompt the metric likes.	The number can miss the goal.
Other brittleness	The model changes under the climb.	Reruns flip close picks.	Template copies swing a small test.
Compounding	Patched noise looks like progress.	Lucky picks look like progress.	The flattering errors stack in the number.

These limit what any number can say, however clean the climb.