KERVYN

RESEARCH PROJECT

PAPER NO. 1 · NOVEMBER 2026

How Kervyn models are trained

Orchestrator, tutors, S3 storage, on-demand GPUs, benchmarks, tests and retraining, with the actual results of every version.

- kervyn-dev · development
- kervyn-embed · search

AUTHOR

Luis Blanco Hijazo

linkedin.com/in/luisbh

Personal research project · Kervyn Paper No. 1

1.8 → 22.0%	0 → 35.4%	81.6 → 85.1	18,787
Generator HumanEval, from SFT v2 (Nov 7) to the 200M SFT v7 (Nov 9)	MBPP over the same period, with decontamination against the benchmarks	Mean embedder nDCG@10, from v2 to v3 (in-house encoder)	SFT examples generated by tutors and verified (v7)

ABOUT

What is Kervyn

Kervyn is a personal research project by Luis Blanco Hijazo (linkedin.com/in/luisbh). Its goal is to learn and test as much as possible about language models and artificial intelligence by building them end to end: architecture, tokenizer, data, training, evaluation, infrastructure and deployment.

Instead of starting from an existing model, every Kervyn model is trained from scratch on public data. This means learning firsthand what is usually hidden behind a finished model: what data works, how to clean and decontaminate it, how much compute it takes, how to measure honestly and what to do when a result isn't what it seems.

Questions guiding the research

- How far can a small in-house model (100M to 1B parameters) go in programming, DevOps and Spanish-language search, and how close can it get to reference models trained on far more data?
- How much of the work of training models can an **orchestrator** take on, and with what checks?
- How can **tutors** generate data without the model inheriting errors, contamination or someone else's identity?
- Which changes to data, recipe or size give the most improvement per hour of training?

Kervyn Paper No. 1

Kervyn is a living project, and its progress will be published on an ongoing basis in numbered papers, each one covering what was trained, how it was measured and what went wrong, always with real numbers. This document is **Kervyn Paper No. 1**: it describes the complete training pipeline and the results obtained up to November 9, 2026.

> **How to cite this paper.** Blanco Hijazo, L. (2026). *How Kervyn models are trained*. Kervyn Paper No. 1, November 2026.

CONTENTS

01 What gets trained, and under what rules . . . 3
02 Who does what . . . 4
03 The training cycle . . . 5
04 Data: public and tutor-generated . . . 6
05 S3 as the single source of truth . . . 8
06 On-demand compute . . . 8
07 The training recipes . . . 9
08 Benchmarks and tests . . . 10
09 Real results, version by version . . . 11
10 Retraining: four real cases . . . 13
11 Current status and next steps . . . 14

SUMMARY

At a glance

Kervyn is a family of **in-house** language models: the architecture, the tokenizer and the weights are its own, trained from scratch on public data. Two model lines are currently in training: the generator **kervyn-dev-flash** (programming and DevOps in Spanish and English, fast on CPU) and the embedder **kervyn-embed** (document search and RAG).

Training is a **cycle**: data → training on a rented GPU → benchmarks → tests → diagnosis → retraining. Each iteration changes one specific thing and is measured against the previous one with the same tests.

- **Orchestrator.** Writes the code, prepares each job, picks the right machine, monitors, evaluates and documents. It does not write training data.
- **Tutors.** Generate and critique synthetic data according to the orchestrator's specifications; everything they produce is verified and filtered before it is used.
- **S3.** A bucket in Europe holds data, checkpoints, logs and the state of every job, so any job can run on any provider and resume where it left off.
- **GPUs as needed.** Each job is launched on the machine that meets its memory, speed and time requirements, and the instance destroys itself when the job ends.
- **Honest evaluation.** Decontaminated public benchmarks, tests in real apps and blind review. A correct answer that turns out to be memorization does not count.

01 · SCOPE

What gets trained, and under what rules

The product goal is a programming and DevOps assistant that runs on the customer's laptop or server **without a GPU**, and an embedder for searching documents in Spanish and across Spanish and English. That is why the models are small and every family keeps at least one version that is fast on CPU.

Model	What it does	Size	Status as of Nov 9, 2026
kervyn-dev-flash-110m-8k	Generator: code, SQL, JSON, DevOps and ES/EN chat	110M, 8k context	Published privately. GGUF Q8_0 131 MiB / Q4_K_M 77 MiB
kervyn-dev-flash 200M	The same generator, grown in depth	200M (32×768), 32k context	SFT v7 evaluated (HumanEval 22.0%); SFT v7b in preparation
kervyn-embed v3	Bi-encoder for retrieval/RAG (ES, EN, ES↔EN, code)	104.5M (in-house encoder)	Phase 3 in training; final benchmark pending
kervyn-dev-pro 1B	Higher-quality generator	1.09B	Planned, in phases, with a numeric gate before each one

Rules applied to every training run

- **Own model.** Third-party models are not fine-tuned, and their weights, probabilities or vectors are not distilled. The architectures follow published designs, but the weights always start from scratch.
- **Public data with the license recorded** for each source, plus synthetic data from tutors whose terms allow training on what they generate.
- **The orchestrator does not write training data.** It writes code, specifications and prompts; the text the model learns from comes from public datasets or from the tutors.
- **Identity.** The model always introduces itself as Kervyn. Every SFT set goes through a filter that drops conversations where the assistant claims to be another model or to have been trained by another company.
- **Honest evaluation.** Data is decontaminated against the benchmarks, and the “clean” score is also published.

- **Right-sized machines.** Before each launch, live offers are compared and the one that meets the job's memory, speed and time requirements is chosen.
- **Nothing unmonitored.** Every job has a watcher; if it fails, it is diagnosed and relaunched from the last checkpoint without waiting for someone to notice.

02 · ROLES

Who does what

The work is split between a human project lead, an orchestrator, tutors and cloud services. The diagram shows who talks to whom; the table shows what each one does and does not do.

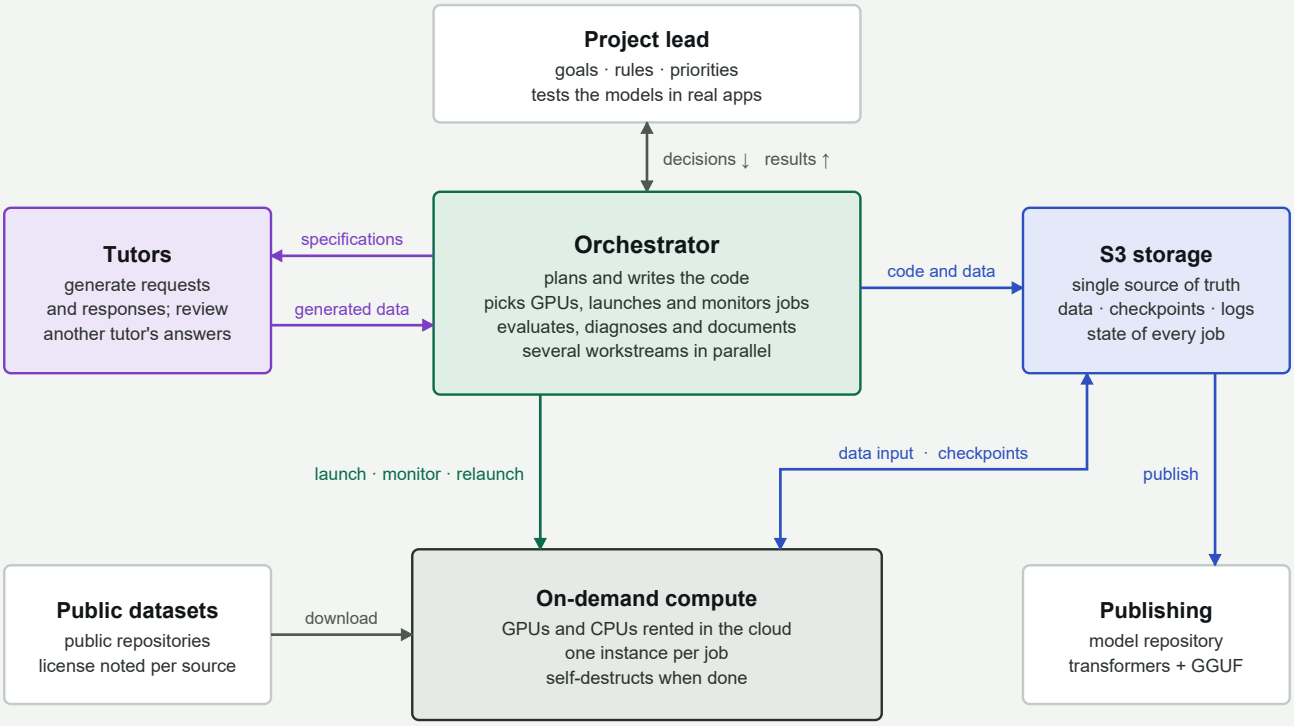


FIG. 1 Architecture of the training system. Arrows show the main flow of information.

Actor	What it does	What it does not do
Project lead (person)	Sets goals, rules and priorities. Tests the models in real apps (that is how the chat failure in §10 was found). Decides what gets published.	—
Orchestrator	Plans experiments; writes the code for data, training, evaluation and infrastructure; picks GPUs, launches and monitors jobs; diagnoses and relaunches; runs benchmarks and audits; documents every decision with numbers in the repository.	Does not write training data. Does not publish anything that has not passed evaluation.
Tutors	Generate realistic user requests, responses, hidden tests and SQL schemas; critique another tutor's responses; generate queries for training embeddings.	Contribute no weights or scores to the model; what they generate only goes in if it passes verification and the filters.
S3 (storage)	Single source of truth: packaged code, data, checkpoints, results and job state.	—
Compute providers	GPUs and CPUs by the hour, one instance per job.	Keep nothing: the instance is destroyed when the job ends.

In practice the orchestrator runs **several workstreams in parallel**, one per model family (flash, embeddings), each with its own machines and jobs. Major decisions go through **independent reviews**: four parallel audits diagnosed the SFT that “looked good” (§10), several cross-evaluated proposals shaped the plan for the 1B model, and independent reviewers compared versions blind on 48 real tasks.

03 · PROCESS

The training cycle

No model comes out of a single training run. Each version is one iteration of the same cycle, and each iteration changes one specific thing (data, recipe or size) so that the improvement can be attributed.

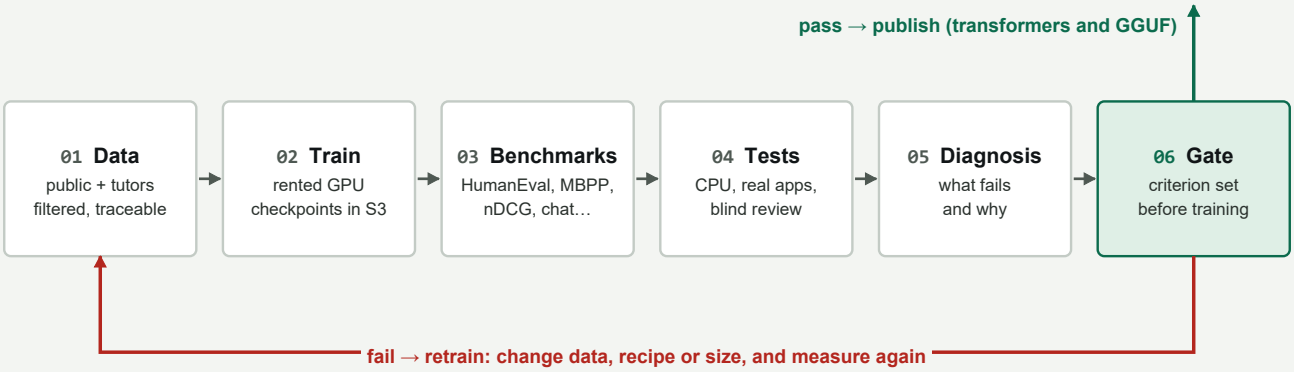


FIG. 2 The cycle. The gate is a numeric criterion set before moving on to the next phase.

- **Data.** Built with reproducible scripts; every build leaves a MANIFEST with counts and discard reasons (§4).
- **Train.** On a rented instance that pulls the data from S3 and uploads checkpoints every 30 minutes (§5-6).
- **Benchmarks.** Always the same ones, so results can be compared with the full history (§8).
- **Tests.** CPU speed, behavior in real chat apps and blind review.
- **Diagnosis.** If a number doesn't add up, it is audited before going further: a high number can also be a bug.
- **Gate.** If the result misses the criterion, work stops for analysis instead of more blind training.

Example gate (kervyn-embed v3). Before moving on to the long phases, the gate required: mean nDCG@10 at least 2 points above v2; improvement in Spanish and in ES→EN with the 95% confidence interval (95% CI) of the difference above zero; and code search dropping no more than 2 points. Result: +3.43 on average, +7.64 on ES→EN, code −0.41 (a tie). It passed and was scaled up.

04 · DATA

Data: public and tutor-generated

Public data

The bulk of the data comes from public datasets. Each source is recorded with its license and its token or example count, and all of them go through the same quality and decontamination filters.

Use	Main sources	Volume
Pretraining v0.3 (flash)	C4 Spanish and English, Python/TS/JS/Java/SQL/HTML code, documentation and ~50 repos (FastAPI, Angular, React, Docker, Kubernetes)	~1-2B tokens
Code CPT, round 1	python-edu, OpenCoder (annealing and fineweb-code), stack-edu (JS, TS, SQL, Shell), fineweb-2 Spanish, filtered HTML, v0.3 replay	2.18B tokens
CPT round 2	New, non-overlapping material + DevOps block (Kubernetes YAML, Dockerfile, Makefile, Shell, PowerShell, CMake…)	2.63B + 165M tokens
CPT round 3 (200M)	New slices of the same source families	4.16B tokens seen
Public SFT (v5full)	Verified OpenCoder SFT, self-oss-instruct, OpenHermes-ES, CodeAlpaca, Magicoder, sql-create-context, Tested-Python, alpaca-spanish, json-mode, NL2SH, DevOps…	304k examples, 95.7M tokens
Encoder MLM (embeddings)	fineweb-2 ES 34%, Wikipedia ES 10%, fineweb-edu 27%, Wikipedia EN 7%, permissively licensed code 22%	8.6B unique, 8B seen
Weak pairs (embeddings)	Wikipedia title→text and ES↔EN, Europarl, UN, Tatoeba, Global Voices, StackExchange (including es.stackoverflow), S2ORC, CodeSearchNet	25.4M pairs (38% with Spanish)

Tutor-generated data

Where public data does not cover what is needed (Spanish responses in Kervyn's style, in-app chat, DevOps, debugging), a tutor generates it from a **specification** written by the orchestrator: task families, topic areas, user personas and a style guide. The text of the requests and responses is written by the tutors; then everything is verified and filtered. The process can be resumed because each stage keeps its cache on disk.

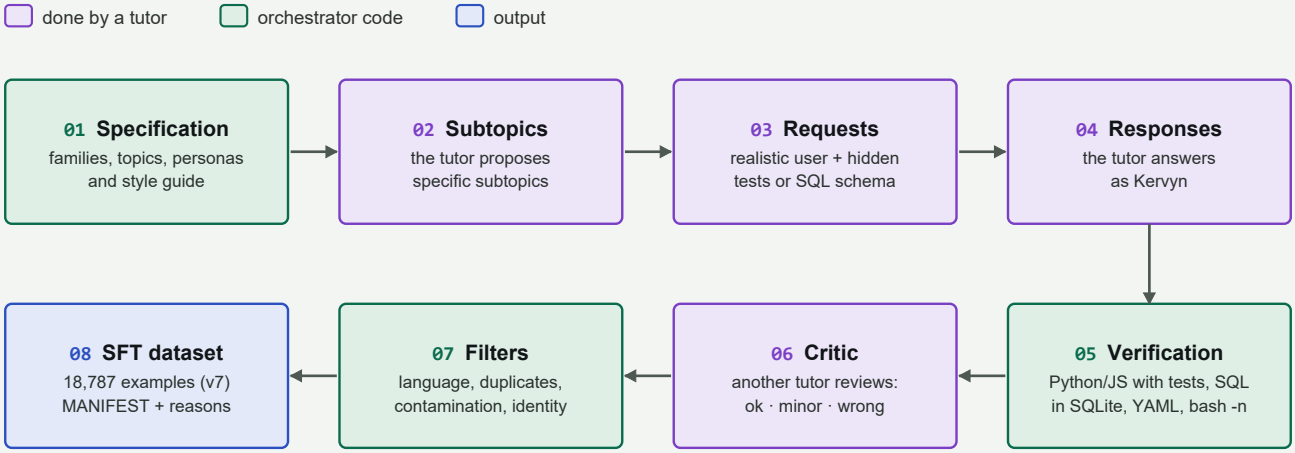


FIG. 3 Tutor data pipeline (`gen_teacher_sft.py`). The critique is always done by a different tutor from the one that answered.

SFT v7 result: 18,787 of 22,146 candidates are kept (84.8%)

Discard reason	Examples
Fails verification (Python/JS tests, SQL result, execution)	2,435
Marked as wrong by the critic tutor	694
Contamination with evaluations (benchmarks and chat prompts from the eval)	148
Invalid syntax (Python, Bash, YAML)	32
Language other than the one requested	27
Duplicates, made-up context, length, CJK characters, identity	23
Total discarded	**3,359**

v7 composition	
By tutor	three different tutors: 7,505 · 7,428 · 3,854
By language	Spanish 13,220 · English 5,567
By family	chat (greeting, thanks, identity, capabilities, vague request) 7,039 · explain 3,072 · how-to 2,956 · Python 2,073 · DevOps file 855 · multi-turn 728 · review 702 · JavaScript 647 · SQL 371 · debugging 344
Critic verdicts	in the families that get critiqued, 74% come out “ok”, 18% “minor error” (kept) and 8% “wrong” (discarded)

For the embeddings, the tutors write **search queries** over real passages that have already been decontaminated: Spanish questions about English text (the hardest cross-lingual task) and Spanish requests for code functions. Before choosing a tutor, a pilot was run with the four candidates:

Tutor (pilot, 200 passages)	Passages / hour	Verbatim copying of the passage	Kervyn model R@1 (lower = harder)
Tutor A	21,000	17%	71%
Tutor B	15,700	19%	77%
Tutor C	4,200	16%	81%
Tutor D	13,000	30%	77%

A and B were chosen: the two fastest, with little verbatim copying of the passage, and two styles instead of one. In total: 224,998 passages → **428,304 training pairs** and 5,900 for validation.

Data quality and traceability

- **Decontamination.** All training data is compared against everything that is evaluated: function names and 10-grams in SFT, informative 8-grams in embeddings, 3-line windows from the CodeSearchNet test functions, and 16-grams for the “clean” report. Each source keeps a sample of what was discarded.
- **Deduplication and caps** per template and per source. Lesson learned: in SFT v2, 91.6% of the responses were duplicates, and the model memorized them (§10).
- **Verification by execution.** Code is run against its tests, SQL is run in SQLite and the rows compared, YAML is parsed, and shell has to pass `bash -n`.
- **Format filters.** Expected language, length, broken encoding (mojibake), unexpected CJK characters, claims of another identity.
- **Traceability.** Every build leaves `MANIFEST.json` and `stats.json` with counts and discard reasons. The MANIFEST is uploaded last: its presence in S3 marks the dataset as complete, and it is the signal the training jobs wait for.

05 · STORAGE

S3 as the single source of truth

Everything a job needs or produces lives in an **S3 bucket** (1 TB, hosted in Europe).

Prefix	Contents
`code/`	Code packaged per commit (`tar.gz`) and bootstrap scripts.
`data/`	Tokenized bins, embedding pairs and tutor data, each with its MANIFEST.
`runs/<run>/`	`last.pt` for resuming (weights, optimizer, RNG), `best.pt`, logs and benchmark results in JSON.
`deploy/`	Exports ready to publish: transformers and GGUF.
`jobs/<job>/`	`STARTED`, `ALIVE` (heartbeat), `EXIT` (exit code) and `runner.log` for each job.

- **Integrity.** Uploads carry Content-MD5; a file is skipped if it already exists with the same size and the same MD5. Only files untouched for more than 2 minutes are uploaded, to avoid uploading half-written checkpoints.
- **No S3 mount.** Training reads the bins with random access, so they are first copied to the instance's local disk.

06 · COMPUTE

On-demand compute

There are no permanent machines. Each job (training, evaluation, testing) rents just the machine it needs and releases it when done. The GPU is chosen not out of habit but by what **the job needs**: memory, speed, duration and where the data is.

Choosing the machine

`gpu_market.py` reads live offers from several cloud GPU providers and, for each one, estimates the actual duration of the job: hours adjusted by that GPU's relative speed (measured on earlier jobs when available) plus the time to download the data from S3. It discards offers that do not meet:

- the job's minimum memory (VRAM) and a sufficient driver version;
- host reliability ≥ 0.99 and advertised bandwidth ≥ 2,000 Mbps;
- a host contract of at least 1.5 × the job's hours + 2 h.

GPUs matched to each training job's needs

Job	What it needs	Machine chosen
Flash 110M CPT at 8k (v4)	Memory for micro-batch 4 at 8k, and maximum speed	96 GB GPU
200M CPT round 3 (continuation)	32 GB is enough with micro-batch 2 and the same tokens per step	32 GB GPU
Encoder MLM up to 8B tokens	Sustained throughput (~244k tokens/s)	32 GB GPU
Embeddings phases 2-7 (16k batches)	More than 32 GB after an OOM on the first step	48 GB GPU
Flash 8k: base, FIM and SFT	24 GB is enough	24 GB GPU
Tests (pytest)	CPU, ~40 s end to end	4-vCPU CPU machine
Bulk verification / reviewers	Many cores, SSH	16-core CPU machine

Job lifecycle

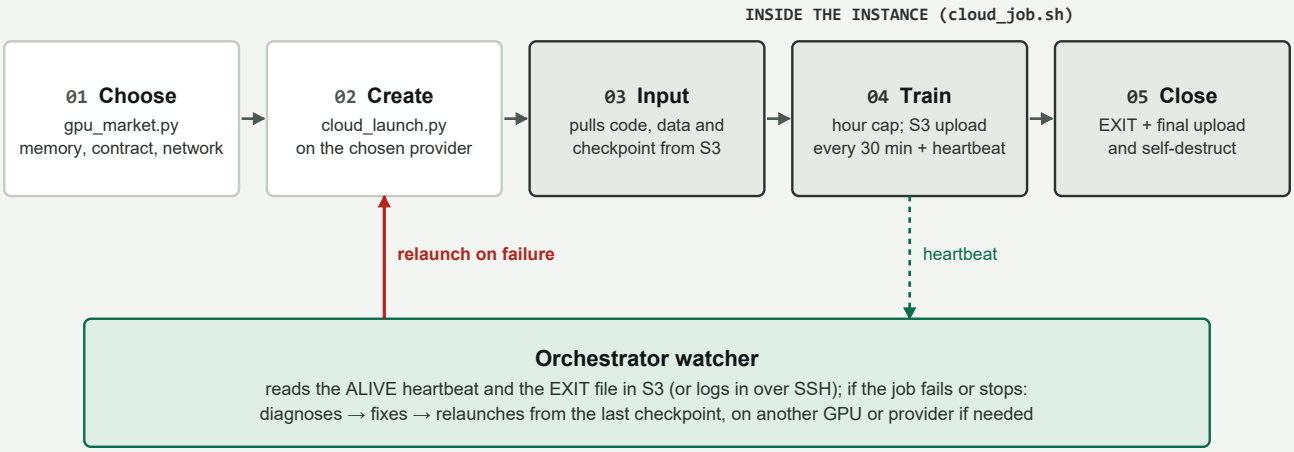


FIG. 4 Job lifecycle. If the instance restarts and `EXIT` already exists, the job is not repeated; if it is missing, it resumes from `last.pt`.

Automatic recovery: real incidents

Incident	Cause	Response
Embeddings phase 2: out of memory (OOM) on the first step, on 32 GB	The large-batch technique (GradCache) recomputed 2,048 queries at once	Token-capped chunks and automatic retry that splits the chunk; relaunched on a 48 GB GPU from the checkpoint. Actual peak: 15.2 GB.
A job died while compiling the model	The base image lacked the Python headers that torch.compile needs	Tested default image (torch 2.14 + toolchain); the scripts install whatever is missing.

07 · RECIPES

The training recipes

kervyn-dev-flash generator

Decoder transformer (RoPE, RMSNorm, SwiGLU, grouped-query attention and tied embeddings) with an in-house byte-level BPE tokenizer of 24,576 tokens. The 200M has 32 layers of width 768, 12 heads (4 KV) and a 32k context. The stages are chained, and each one starts from the best checkpoint of the previous one:

Stage	What is done	Scale	Outcome
01 Tokenizer	Byte-level BPE trained on public data; no unknown tokens	24,576 ids	—
02 Pretraining	Public corpus v0.3 (ES/EN text + code)	~1-2B tokens	bpb 0.99 (110M)
03 Code CPT, round 1	Decontaminated public code + corpus replay	1.97B tokens	Base HumanEval 1.2 → 7.3%
04 CPT round 2	New material + DevOps block; LR re-warmed	2.36B tokens	HumanEval 10.4%, MBPP 21.8%
05 Grow 110M → 200M	16 → 32 layers; the new ones start as identity, so the grown model computes exactly the same as the original	—	More capacity without losing what was learned
06 CPT round 3	New tokens on the grown model	4.16B tokens	Base HumanEval 18.3%
07 Long context	8k → 32k (RoPE $\theta = 10^6$)	2,085 steps	Base MBPP 25.3%; uses the context up to 32k
08 SFT	Public (304k examples) + tutors v7 (18,787) + chat robustness	3,060 steps	HumanEval 22.0%, MBPP 35.4% (chat)
09 Export	transformers + GGUF Q8_0 / Q4_K_M for llama.cpp	—	Runs on CPU

Two recipe changes made a big difference: **per-document attention** (each packed file or conversation only attends to itself) with windows aligned to the start of each document, and `torch.compile` with fused AdamW. On a 96 GB GPU at 8k, throughput went from 118k to 259k tokens/s: the same machine trains twice as much in the same time.

kervyn-embed v3 embedder

v2 was an adapted decoder, and its base had never seen Spanish prose. v3 is an **in-house bidirectional encoder** (modern encoder architecture, randomly initialized, 12 layers of 768, 104.5M parameters) that loads with transformers or sentence-transformers without extra code.

Phase	What is done	Scale	Outcome
00 Reliable benchmark	Ties count against, XQuAD, bootstrap 95% CI, 150 in-house queries	—	Comparable measurements
01 MLM from scratch	30% masking; 45% Spanish, 35% English, 20% code	8B tokens	—
1b A/B gate	The same fine-tune as v2 on the new base	1.5M pairs	+3.43 nDCG@10: pass
02 Weak contrastive	Natural human-written pairs, 16k batches with GradCache	25.4M pairs · 1,220 steps	Code 89.3 (beats the reference)
03 Hard negatives	Mined with the Kervyn model itself	2.05M pairs	—
04 Supervised	Public pairs + 428k synthetic queries from tutors	8,012 steps	In progress
05 Final benchmark	Against v2 and market reference embedders	—	Pending
06 CPU version	12×384 distilled from Kervyn's own base, Matryoshka 256, int8	—	Planned

08 · EVALUATION

Benchmarks and tests

The tests are always the same, so that every version is compared with the full history. The raw results (one row per problem) are stored in S3 next to the checkpoint.

Test	What it measures	How
Validation bpb	Language quality, comparable across tokenizers	Fixed validation windows
HumanEval (164) and MBPP (257)	Writing correct functions	Greedy pass@1, running the tests; base, chat and prefill modes
“Clean” score	The same, without contaminated problems	16-grams against all training data
eval-exec (41) and code bench (50)	JSON, SQL, Python, JS, HTML with prompts in Spanish	Deterministic checks
Chat robustness	Language, system-prompt leaks, greetings, identity, asking on vague requests, code	5 system prompts, 3 of them from apps never seen in training
Effective context	Whether the model makes use of long context	bpb with a real vs. random prefix, from 1k to 32k
Qualitative evaluation	Varied real tasks	48 tasks in Spanish, independent reviewers and blind comparison
CPU speed	Prompt and generation tokens/s, RAM	llama.cpp with GGUF Q8_0/Q4_K_M, on cloud CPU machines
Embeddings	nDCG@10, R@k, MRR, Spearman	Belebele ES/EN/ES→EN, XQuAD, CodeSearchNet, STS17 and kervyn-docs (150 in-house queries)
Software tests	That the code does what it claims	pytest in a cloud CPU job (~40 s)

- **Significance.** Differences are checked with paired tests (McNemar, sign test) or with the bootstrap 95% CI of the per-query difference. An improvement within the noise is not accepted.
- **High numbers get audited too.** The flash's first “good” result was template memorization (§10); since then, every big jump is reviewed problem by problem.
- **Inference settings are measured, not assumed.** For example, repetition penalty: 1.05 gives the best HumanEval and 1.1 cuts chat loops from 21 to 7 out of 48 tasks, so each is recommended depending on the use.

09 · RESULTS

Real results, version by version

Generator: writing correct functions

Each row is a real training run with one specific change. From November 7 to 9, HumanEval went from 1.8% to 22.0% and MBPP from 0% to 35.4%, without using third-party weights.

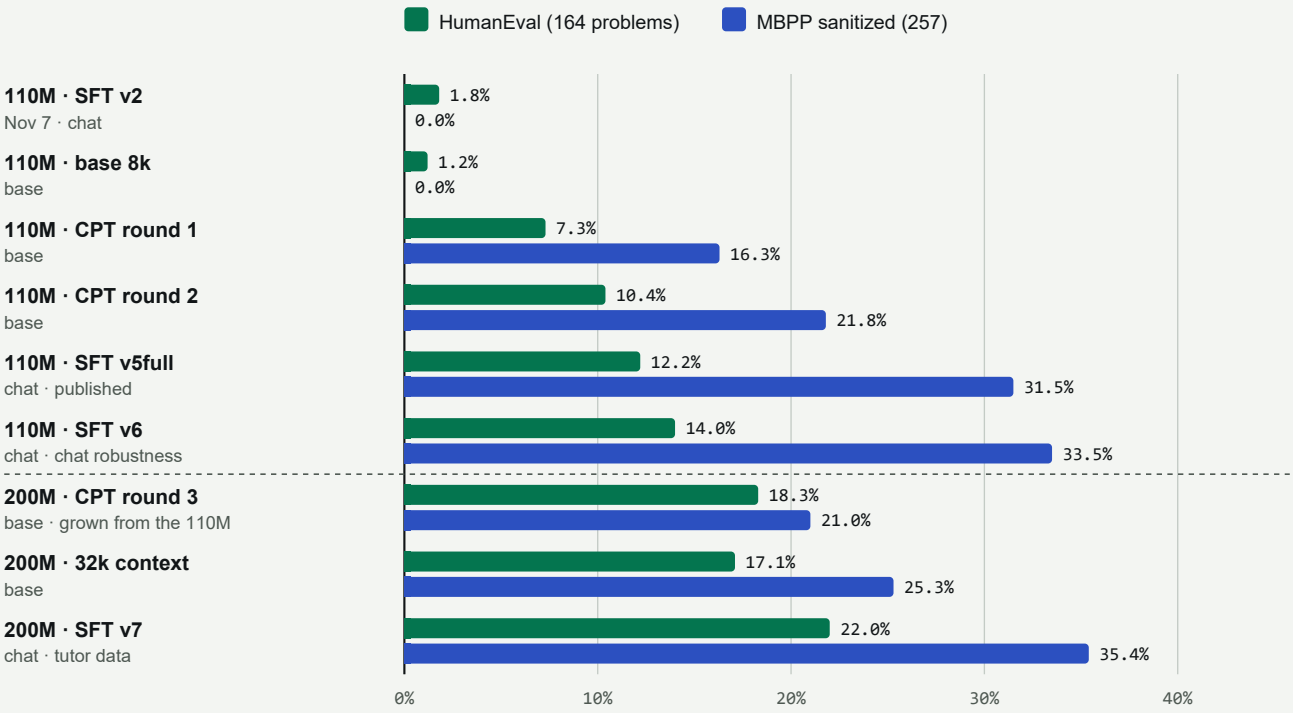


FIG. 5 Greedy pass@1. Base models are evaluated in completion mode and SFT models in chat mode. The dashed line separates the 110M from the 200M. “Clean” score of the published 110M (v5full): 12.5% / 30.5%. Source: `runs/*/bench-*.summary.json` in S3.

Generator: behavior in a chat app

Metrics from `eval_chat.py` summed over five system prompts, three of them from apps the model never saw in training. They show the v6 fix and the regression that v7 brought when it grew (see §10, case 3).

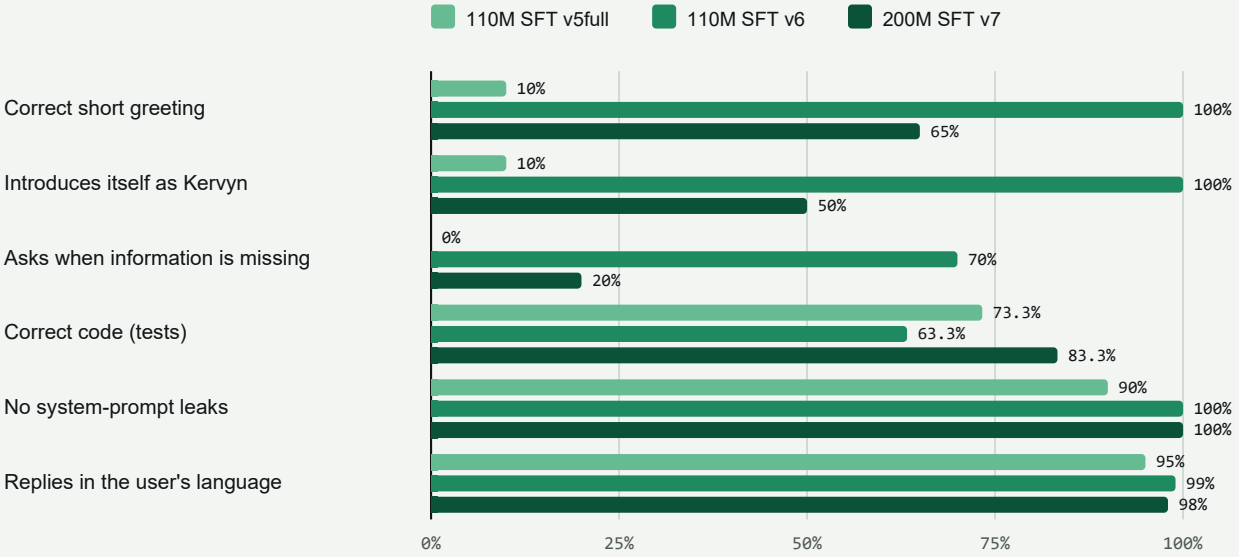


FIG. 6 Chat robustness by version (darker = more recent). In addition, v7 never claims another identity (100/100): what it loses is introducing itself by name when asked.

Embedder: search in Spanish, across languages and in code

nDCG@10 (0-100, higher is better). Each dot is a version; the diamond is a market multilingual embedder of similar size, as a reference. The axis starts at 60 so the differences are visible.

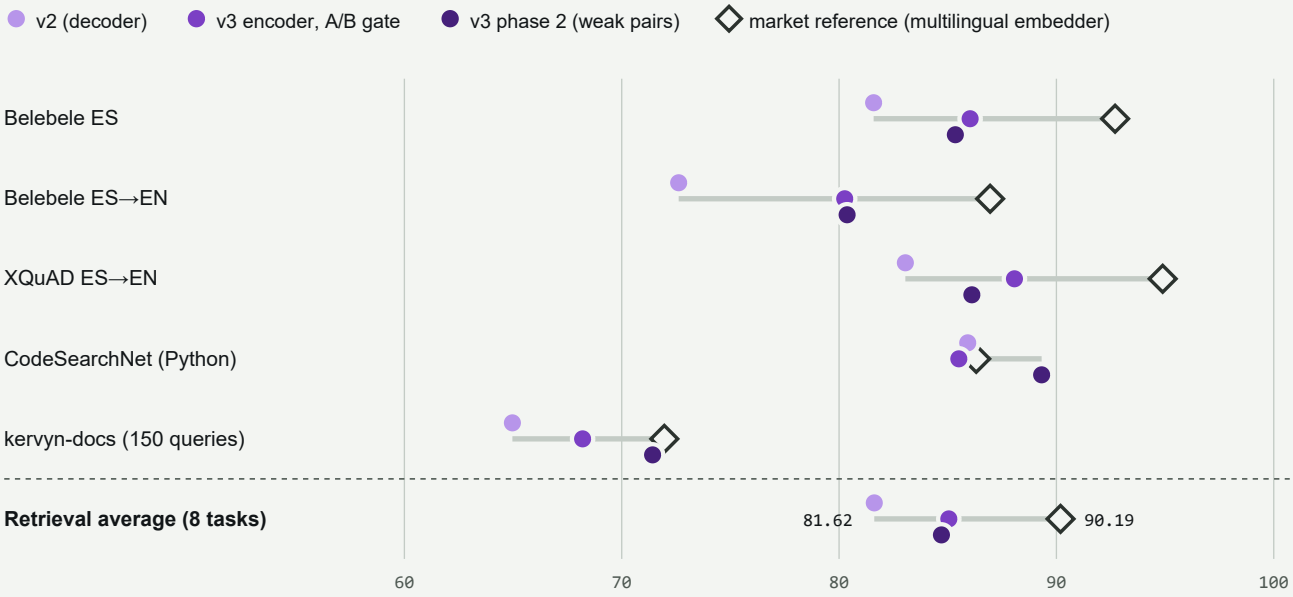


FIG. 7 kervyn-embed: v2 vs. the v3 stages measured so far. The final benchmark (phase 3) is still to come. v1 scored 38 on Belebele ES.

Task (nDCG@10)	v2	v3 A/B gate	v3 phase 2	Reference
Belebele ES	81.59	86.03	85.35	92.70
Belebele EN	86.81	90.12	88.40	94.86
Belebele ES→EN	72.62	80.26	80.37	86.95
XQuAD ES	87.63	90.31	87.74	96.40

Task (nDCG@10)	v2	v3 A/B gate	v3 phase 2	Reference
XQuAD EN	90.36	91.90	88.98	97.48
XQuAD ES→EN	83.05	88.07	86.11	94.89
CodeSearchNet (Python)	85.92	85.51	89.32	86.31
kervyn-docs (150 queries)	64.97	68.20	71.42	71.96
Retrieval average	**81.62**	**85.05**	**84.71**	**90.19**

Combined with BM25 (hybrid search), the A/B-gate encoder scores 75.56 on kervyn-docs, above the reference (71.96).

CPU speed of the published flash 110M

GGUF	Size	Prompt at 512 tokens	Prompt at 8k	Generation
Q8_0	131 MiB	~1,020 tok/s	~460 tok/s	120-180 tok/s
Q4_K_M	77 MiB	~1,500 tok/s	~540 tok/s	200-220 tok/s

llama.cpp, 6 threads, under 1 GB of RAM. The speed of the 200M will be measured on a cloud CPU machine before it is published.

10 · RETRAINING

Retraining: four real cases

This is how the cycle works when something goes wrong. In all four cases the problem was found by measuring, it was diagnosed before moving on, and the next training run changed one specific thing.

CASE 01

“Its correct answers were memorized” (flash 110M, SFT v2)

SYMPTOM	11/50 on the code bench and 6/41 on eval-exec looked like a good result for a 110M model.
DIAGNOSIS	Four parallel audits: the correct answers were byte-for-byte identical to training examples; the SFT was ~92% templates, with 91.6% duplicated responses; decontamination only removed exact copies; and 82% of the validation set was in train. It scored 1.8% on HumanEval and 0% on MBPP.
CHANGE	Two CPT rounds with decontaminated public code, SFT on public datasets only, per-document attention, and public benchmarks with a “clean” score.
RESULT	**HumanEval 1.8 → 12.2% and MBPP 0 → 31.5%** (clean: 12.5% / 30.5%). In a blind comparison with independent reviewers it beat the intermediate version 15-5 (sign test p ≈ 0.04).

CASE 02

“In a real app its answers make no sense” (published flash 110M)

SYMPTOM	Tested in a desktop chat app, the model echoed the system prompt and couldn't greet or say who it was.
DIAGNOSIS	The app sends ~3,000 tokens of system prompt (rules and tools); the SFT only had two short system prompts and no identity, greeting or multi-turn data.
CHANGE	SFT v6 with app system prompts, identity, greetings and multi-turn, plus a new evaluation with prompts from apps that training never sees.
RESULT	**Prompt leaks 10 → 0 out of 100; greetings 2 → 20 out of 20; identity 2 → 20 out of 20**; HumanEval rises to 14.0% and MBPP to 33.5%.

CASE 03

“Bigger, but it lost its chat skills” (flash 200M, SFT v7)

SYMPTOM	The 200M codes better (HumanEval 22.0%, correct code 25/30), but it greets properly 13/20, introduces itself as Kervyn 10/20 and asks about vague requests only 4/20.
DIAGNOSIS	In SFT v7, chat mixed in with the other families made up less than 1% of the data, and the larger model forgot it.
CHANGE	v7b: chat subfamilies requested separately from the tutors, each with its own quota and guide: 1,095 greetings, 834 thank-yous, 1,454 identity, 692 capabilities and 1,758 vague requests.
RESULT	In preparation (314,784 examples). The same chat evaluation decides, without losing HumanEval.

CASE 04

“The base didn't know Spanish” (kervyn-embed v2 → v3)

SYMPTOM	v2 averaged 8.6 points below the market reference and did worse on Spanish → English (72.6 vs. 87.0).
DIAGNOSIS	The base was a causal decoder pretrained without Spanish prose, and the contrastive phase used 1.5M pairs versus the reference's ~1,000M.
CHANGE	In-house bidirectional encoder with 45% Spanish in pretraining, an A/B gate before scaling, and then 25.4M natural pairs with 16k batches.
RESULT	**+3.43 on average and +7.64 on ES→EN** with the same fine-tune; after phase 2 it already beats the reference on code (89.3 vs. 86.3). The average gap shrinks from 8.6 to 5.1 points.

11 · STATUS

Current status and next steps

Line	Now	Next
kervyn-embed v3	Phase 3 (supervised) at step ~6,900 of 8,012	Final benchmark against reference embedders; publish or scale up; CPU version 12×384
kervyn-dev-flash 200M	SFT v7 evaluated: better code, chat to be recovered	SFT v7b, chat evaluation and CPU speed; publish if it improves without breaking chat
kervyn-dev-pro 1B	Plan with architecture, data and phases	48k tokenizer and de-risking experiment with gates G0-G4

Each line follows the same cycle. Kervyn Paper No. 2 will report these results, measured with the same tests so they can be compared with everything that came before.

APPENDIX

Glossary

Term	Meaning
Pretraining / CPT	Training the model to predict the next token on text and code. CPT is “continued pretraining”: further pretraining an existing model on new data.
SFT	Supervised fine-tuning on question-answer conversations so the model follows instructions.
Checkpoint	A copy of the training state; `last.pt` allows resuming and `best.pt` is the best one on validation.
bpb	Bits per byte on validation: how “surprised” the model is; lower is better.
pass@1	Percentage of problems solved on the first attempt, running the tests.
HumanEval / MBPP	Public Python programming benchmarks (164 and 257 problems).
Contamination	When a benchmark problem (or its solution) is in the training data.
MLM	Encoder pretraining: masking tokens and learning to recover them.
Contrastive	Training so that a query and its document end up close together in vector space.
Hard negatives	Similar but incorrect documents that force the model to make finer distinctions.
GradCache	A technique for using batches of thousands of pairs without fitting them in memory all at once.
nDCG@10	Ranking quality over the top 10 results of a search (0-100).
GGUF, Q8_0, Q4_K_M	The llama.cpp format, and 8-bit and ~4-bit quantizations for running on CPU.
Orchestrator	The central piece of the system: plans, writes the code, launches and monitors jobs, evaluates and documents. Does not write training data.
Tutor	Generates or reviews training data according to a specification from the orchestrator. Contributes no weights to the model.

From Logroño with ❤ and lots of ☕

Luis Blanco Hijazo · linkedin.com/in/luisbh