KERVYN

PROYECTO DE INVESTIGACIÓN

PAPER N.º 1 · NOVIEMBRE 2026

Cómo se entrenan los modelos Kervyn

Orquestador, tutores, almacenamiento S3, GPU bajo demanda, benchmarks, pruebas y re-entrenamientos, con los resultados reales de cada versión.

● kervyn-dev · desarrollo ● kervyn-embed · búsqueda

AUTOR

Luis Blanco Hijazo

linkedin.com/in/luisbh Proyecto de investigación personal · Paper n.º 1 de Kervyn

1,8 → 22,0 %	**0 → 35,4 %**	**81,6 → 85,1**	**18.787**
HumanEval del generador, del SFT v2 (7 nov) al 200M SFT v7 (9 nov)	MBPP en el mismo periodo, con descontaminación contra los benchmarks	nDCG@10 medio del embedder, de v2 a v3 (encoder propio)	ejemplos SFT generados por tutores y verificados (v7)

PRESENTACIÓN

Qué es Kervyn

Kervyn es un proyecto de investigación personal de Luis Blanco Hijazo (linkedin.com/in/luisbh). Su objetivo es aprender y probar lo máximo posible sobre modelos de lenguaje e inteligencia artificial construyéndolos de principio a fin: arquitectura, tokenizer, datos, entrenamiento, evaluación, infraestructura y despliegue.

En lugar de partir de un modelo ya hecho, cada modelo Kervyn se entrena desde cero con datos públicos. Eso obliga a resolver de primera mano lo que normalmente queda oculto detrás de un modelo terminado: qué datos sirven, cómo se limpian y se descontaminan, cuánto cómputo hace falta, cómo se mide de forma honesta y qué hacer cuando un resultado no es lo que parece.

Preguntas que guían la investigación

- ¿Hasta dónde llega un modelo pequeño y propio (de 100M a 1B parámetros) en programación, DevOps y búsqueda en español, y cuánto se puede acercar a modelos de referencia mucho más entrenados?
- ¿Qué parte del trabajo de entrenar modelos puede llevar un **orquestador**, y con qué controles?
- ¿Cómo se usan **tutores** para generar datos sin que el modelo herede errores, contaminación o una identidad ajena?
- ¿Qué cambios de datos, receta o tamaño dan más mejora por hora de entrenamiento?

Paper n.º 1 de Kervyn

Kervyn es un proyecto vivo y sus avances se irán publicando de forma continua en papers numerados, cada uno con lo que se entrenó, cómo se midió y qué salió mal, siempre con cifras reales. Este documento es el **Paper n.º 1 de Kervyn**: describe el pipeline completo de entrenamiento y los resultados obtenidos hasta el 9 de noviembre de 2026.

> **Cómo citar este paper.** Blanco Hijazo, L. (2026). *Cómo se entrenan los modelos Kervyn*. Paper n.º 1 de Kervyn, noviembre de 2026.

CONTENIDO

01 Qué se entrena y con qué reglas . . . 3
02 Quién hace qué . . . 4
03 El ciclo de entrenamiento . . . 5
04 Datos: públicos y de tutores . . . 6
05 Un S3 como fuente única . . . 8
06 Cómputo bajo demanda . . . 8
07 Las recetas de entrenamiento . . . 9
08 Benchmarks y pruebas . . . 10
09 Resultados reales, versión a versión . . . 11
10 Re-entrenamientos: cuatro casos reales . . . 13
11 Estado actual y próximos pasos . . . 14

RESUMEN

En una página

Kervyn es una familia de modelos de lenguaje **propia**: la arquitectura, el tokenizer y los pesos se entrenan desde cero, con datos públicos. Hoy hay dos líneas en entrenamiento: el generador **kervyn-dev-flash** (programación y DevOps en español e inglés, rápido en CPU) y el embedder **kervyn-embed** (búsqueda en documentos y RAG).

El entrenamiento es un **ciclo**: datos → entrenamiento en GPU alquilada → benchmarks → pruebas → diagnóstico → re-entrenamiento. Cada vuelta cambia una cosa concreta y se mide contra la anterior con las mismas pruebas.

- **Orquestador.** Escribe el código, prepara cada trabajo, elige la máquina adecuada, vigila, evalúa y documenta. No redacta datos de entrenamiento.
- **Tutores.** Generan y critican datos sintéticos según las especificaciones del orquestador; todo lo que producen se verifica y filtra antes de usarse.
- **S3.** Un bucket en Europa guarda datos, checkpoints, logs y el estado de cada trabajo, así que cualquier trabajo puede correr en cualquier proveedor y reanudarse donde se quedó.
- **GPU según necesidad.** Cada trabajo se lanza en la máquina que cumple su memoria, su velocidad y su plazo, y la instancia se destruye sola al terminar.
- **Evaluación honesta.** Benchmarks públicos descontaminados, pruebas en apps reales y revisión a ciegas. Un acierto que resulta ser memorización no cuenta.

01 · ALCANCE

Qué se entrena y con qué reglas

El objetivo de producto es un asistente de programación y DevOps que funcione en el portátil o el servidor del cliente **sin GPU**, y un embedder para buscar en documentos en español y entre español e inglés. Por eso los modelos son pequeños y cada familia conserva al menos una versión rápida en CPU.

Modelo	Qué hace	Tamaño	Estado a 9 nov 2026
kervyn-dev-flash-110m-8k	Generador: código, SQL, JSON, DevOps y chat ES/EN	110M, contexto 8k	Publicado en privado. GGUF Q8_0 131 MiB / Q4_K_M 77 MiB
kervyn-dev-flash 200M	El mismo generador, crecido en profundidad	200M (32×768), contexto 32k	SFT v7 evaluado (HumanEval 22,0 %); SFT v7b en preparación
kervyn-embed v3	Bi-encoder para retrieval/RAG (ES, EN, ES↔EN, código)	104,5M (encoder propio)	Fase 3 en entrenamiento; benchmark final pendiente
kervyn-dev-pro 1B	Generador de más calidad	1,09B	Planificado, por fases, con una puerta numérica antes de cada una

Reglas que se aplican en cada entrenamiento

- **Modelo propio.** No se afinan modelos de terceros ni se destilan sus pesos, probabilidades o vectores. Las arquitecturas siguen diseños publicados, pero los pesos parten siempre de cero.
- **Datos públicos con licencia anotada** por fuente, más datos sintéticos de tutores cuyos términos permiten entrenar con lo que generan.
- **El orquestador no escribe datos de entrenamiento.** Escribe código, especificaciones y prompts; el texto que aprende el modelo sale de datasets públicos o de los tutores.
- **Identidad.** El modelo se presenta siempre como Kervyn. Todo SFT pasa por un filtro que descarta conversaciones donde el asistente dice ser otro modelo o haber sido entrenado por otra empresa.
- **Evaluación honesta.** Se descontamina contra los benchmarks y se publica también la cifra «limpia».

- **Máquina a medida.** Antes de cada lanzamiento se comparan ofertas en vivo y se elige la que cumple la memoria, la velocidad y el plazo del trabajo.
- **Nada sin vigilar.** Cada trabajo tiene un vigilante; si falla, se diagnostica y se relanza desde el último checkpoint sin esperar a que alguien lo note.

02 · ROLES

Quién hace qué

El trabajo se reparte entre una persona responsable, un orquestador, tutores y servicios en la nube. El diagrama muestra quién habla con quién; la tabla, qué hace y qué no hace cada uno.

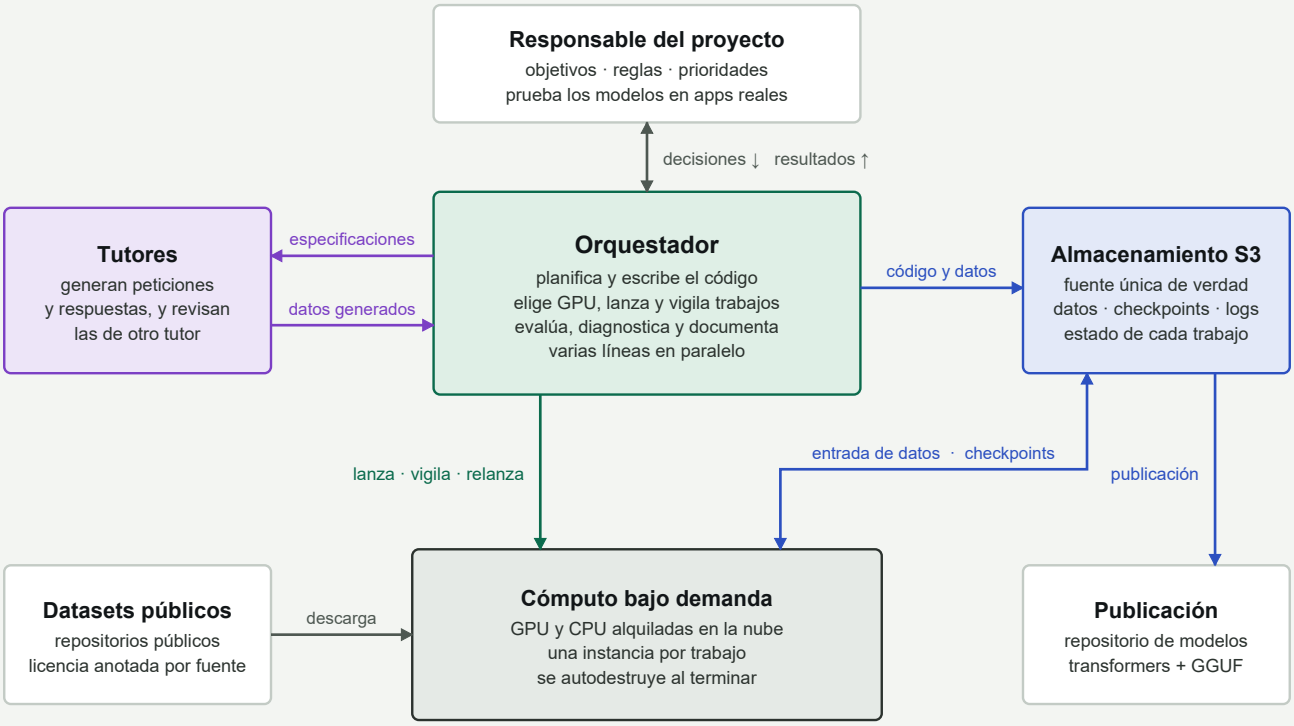


FIG. 1 Arquitectura del sistema de entrenamiento. Las flechas indican el flujo principal de información.

Actor	Qué hace	Qué no hace
Responsable (persona)	Fija objetivos, reglas y prioridades. Prueba los modelos en apps reales (así se detectó el fallo de chat del §10). Decide qué se publica.	—
Orquestador	Planifica experimentos; escribe el código de datos, entrenamiento, evaluación e infraestructura; elige GPU, lanza y vigila trabajos; diagnostica y relanza; ejecuta benchmarks y auditorías; documenta cada decisión con cifras en el repositorio.	No redacta datos de entrenamiento. No publica sin pasar la evaluación.
Tutores	Generan peticiones de usuario realistas, respuestas, tests ocultos y esquemas SQL; critican respuestas de otro tutor; generan queries para entrenar embeddings.	No aportan pesos ni puntuaciones al modelo; lo que generan solo entra si pasa la verificación y los filtros.
S3 (almacenamiento)	Fuente única de verdad: código empaquetado, datos, checkpoints, resultados y estado de los trabajos.	—
Proveedores de cómputo	GPU y CPU por horas, una instancia por trabajo.	No guardan nada: la instancia se destruye al terminar.

En la práctica el orquestador lleva **varias líneas en paralelo**, una por familia de modelo (flash, embeddings), cada una con sus propias máquinas y trabajos. Las decisiones grandes pasan por **revisiones independientes**: cuatro auditorías en paralelo diagnosticaron el SFT que «parecía bueno» (§10), varias propuestas evaluadas de forma cruzada definieron el plan del modelo de 1B, y revisores independientes compararon versiones a ciegas en 48 tareas reales.

03 · PROCESO

El ciclo de entrenamiento

Ningún modelo sale de un único entrenamiento. Cada versión es una vuelta del mismo ciclo, y cada vuelta cambia una cosa concreta (datos, receta o tamaño) para poder atribuir la mejora.

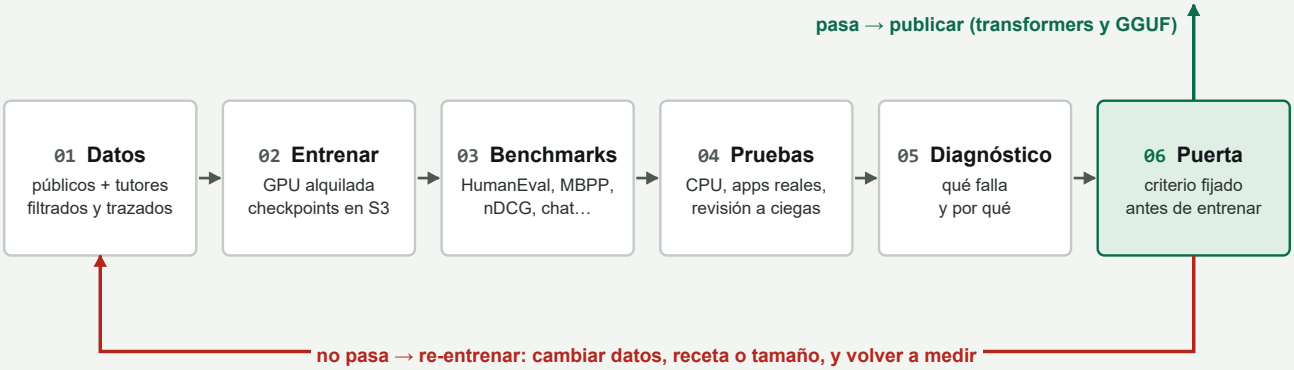


FIG. 2 El ciclo. La puerta es un criterio numérico que se fija antes de pasar a la siguiente fase.

- **Datos.** Se construyen con scripts reproducibles; cada build deja un MANIFEST con recuentos y motivos de descarte (§4).
- **Entrenar.** En una instancia alquilada que baja los datos de S3 y sube checkpoints cada 30 minutos (§5-6).
- **Benchmarks.** Siempre los mismos para comparar con el histórico (§8).
- **Pruebas.** Velocidad en CPU, comportamiento en apps de chat reales y revisión a ciegas.
- **Diagnóstico.** Si una cifra no cuadra se audita antes de seguir: un número alto también puede ser un error.
- **Puerta.** Si el resultado no supera el criterio, se para y se analiza en vez de entrenar más a ciegas.

> **Ejemplo de puerta (kervyn-embed v3).** Antes de pasar a las fases largas se exigió: media de nDCG@10 al menos 2 puntos por encima de v2; mejora en español y en ES→EN con el intervalo de confianza (IC95) de la diferencia por encima de cero; y que la búsqueda de código no cayera más de 2 puntos. Resultado: +3,43 de media, +7,64 en ES→EN, código −0,41 (empate). Pasó y se escaló.

04 · DATOS

Datos: públicos y de tutores

Datos públicos

El volumen viene de datasets públicos. Cada fuente se anota con su licencia y su número de tokens o ejemplos, y todas pasan por los mismos filtros de calidad y descontaminación.

Uso	Fuentes principales	Volumen
Preentreno v0.3 (flash)	C4 español e inglés, código Python/TS/JS/Java/SQL/HTML, documentación y ~50 repos (FastAPI, Angular, React, Docker, Kubernetes)	~1-2B tokens
CPT de código, ronda 1	python-edu, OpenCoder (annealing y fineweb-code), stack-edu (JS, TS, SQL, Shell), fineweb-2 español, HTML filtrado, replay v0.3	2,18B tokens
CPT ronda 2	Material nuevo sin solape + bloque DevOps (Kubernetes YAML, Dockerfile, Makefile, Shell, PowerShell, CMake…)	2,63B + 165M tokens
CPT ronda 3 (200M)	Nuevas porciones de las mismas familias de fuentes	4,16B tokens vistos
SFT público (v5full)	OpenCoder SFT verificado, self-oss-instruct, OpenHermes-ES, CodeAlpaca, Magicoder, sql-create-context, Tested-Python, alpaca-spanish, json-mode, NL2SH, DevOps…	304k ejemplos, 95,7M tokens
Encoder MLM (embeddings)	fineweb-2 ES 34 %, Wikipedia ES 10 %, fineweb-edu 27 %, Wikipedia EN 7 %, código con licencia permisiva 22 %	8,6B únicos, 8B vistos
Pares débiles (embeddings)	Wikipedia título→texto y ES↔EN, Europarl, ONU, Tatoeba, Global Voices, StackExchange (también es.stackoverflow), S2ORC, CodeSearchNet	25,4M pares (38 % con español)

Datos generados por tutores

Donde los datos públicos no cubren lo que hace falta (respuestas en español al estilo Kervyn, chat de app, DevOps, depuración), los genera un tutor a partir de una **especificación** que escribe el orquestador: familias de tareas, áreas temáticas, personas de usuario y guía de estilo. El texto de peticiones y respuestas lo escriben los tutores; después todo se verifica y se filtra. El proceso se puede reanudar porque cada etapa guarda su caché en disco.



FIG. 3 Pipeline de datos de tutores (`gen_teacher_sft.py`). La crítica la hace siempre un tutor distinto del que respondió.

Resultado del SFT v7: de 22.146 candidatos se quedan 18.787 (84,8 %)

Motivo de descarte	Ejemplos
No pasa la verificación (tests de Python/JS, resultado SQL, ejecución)	2.435
El tutor crítico la marca como incorrecta	694
Contaminación con evaluaciones (benchmarks y prompts de chat de la eval)	148
Sintaxis inválida (Python, Bash, YAML)	32
Idioma distinto del pedido	27
Duplicados, contexto inventado, longitud, caracteres CJK, identidad	23
Total descartado	**3.359**

Composición del v7	
Por tutor	tres tutores distintos: 7.505 · 7.428 · 3.854
Por idioma	español 13.220 · inglés 5.567
Por familia	chat (saludo, gracias, identidad, capacidades, petición vaga) 7.039 · explicar 3.072 · cómo hacer 2.956 · Python 2.073 · fichero DevOps 855 · multi-turno 728 · revisión 702 · JavaScript 647 · SQL 371 · depuración 344
Veredictos del crítico	en las familias que se critican, el 74 % sale «ok», el 18 % «error menor» (se conserva) y el 8 % «incorrecta» (se descarta)

Para los embeddings los tutores escriben **queries de búsqueda** sobre pasajes reales ya descontaminados: preguntas en español sobre texto en inglés (la tarea cruzada más difícil) y peticiones en español para funciones de código. Antes de elegir tutor se hizo un piloto con los cuatro candidatos:

Tutor (piloto, 200 pasajes)	Pasajes / hora	Copia literal del pasaje	R@1 del modelo Kervyn (menos = más difícil)
Tutor A	21.000	17 %	71 %
Tutor B	15.700	19 %	77 %
Tutor C	4.200	16 %	81 %
Tutor D	13.000	30 %	77 %

Se eligieron A y B: los dos más rápidos, con poca copia literal del pasaje, y dos estilos en vez de uno. En total: 224.998 pasajes → **428.304 pares de entrenamiento** y 5.900 de validación.

Calidad y trazabilidad de los datos

- **Descontaminación.** Se compara todo el entrenamiento con todo lo que se evalúa: nombres de función y 10-gramas en SFT, 8-gramas informativos en embeddings, ventanas de 3 líneas de las funciones de test de CodeSearchNet y 16-gramas para el informe «limpio». Cada fuente guarda una muestra de lo descartado.
- **Deduplicación y topes** por plantilla y por fuente. Lección aprendida: el SFT v2 tenía el 91,6 % de las respuestas duplicadas y el modelo memorizó (§10).
- **Verificación por ejecución.** El código se ejecuta contra sus tests, el SQL en SQLite comparando filas, el YAML se parsea y el shell pasa `bash -n`.
- **Filtros de forma.** Idioma esperado, longitud, codificación rota (mojibake), caracteres CJK inesperados, identidad ajena.
- **Trazabilidad.** Cada build deja `MANIFEST.json` y `stats.json` con recuentos y motivos de descarte. El MANIFEST se sube el último: su presencia en S3 marca el dataset como completo y es la señal que esperan los trabajos de entrenamiento.

05 · ALMACENAMIENTO

Un S3 como fuente única

Todo lo que un trabajo necesita o produce vive en un **bucket S3** (1 TB, alojado en Europa).

Prefijo	Contenido
`code/`	Código empaquetado por commit (`tar.gz`) y scripts de arranque.
`data/`	Bins tokenizados, pares de embeddings y datos de tutores, cada uno con su MANIFEST.
`runs/<run>/`	`last.pt` para reanudar (pesos, optimizador, RNG), `best.pt`, logs y resultados de benchmarks en JSON.
`deploy/`	Exports listos para publicar: transformers y GGUF.
`jobs/<job>/`	`STARTED`, `ALIVE` (latido), `EXIT` (código de salida) y `runner.log` de cada trabajo.

- **Integridad.** Las subidas llevan Content-MD5; un fichero se salta si ya existe con el mismo tamaño y el mismo MD5. Solo se suben ficheros que llevan más de 2 minutos sin tocarse, para no subir checkpoints a medio escribir.
- **Sin montar el S3.** El entrenamiento lee los bins con acceso aleatorio, así que se copian al disco local de la instancia al empezar.

06 · CÓMPUTO

Cómputo bajo demanda

No hay máquinas fijas. Cada trabajo (entrenar, evaluar, probar) alquila la máquina justa para él y la libera al acabar. La GPU no se elige por costumbre sino por lo que **necesita el trabajo**: memoria, velocidad, duración y dónde están los datos.

Elegir la máquina

`gpu_market.py` lee en vivo las ofertas de varios proveedores de GPU en la nube y estima para cada una la duración real del trabajo: horas ajustadas por la velocidad relativa de esa GPU (medida en trabajos anteriores cuando la hay) más el tiempo de bajar los datos de S3. Descarta las que no cumplen:

- memoria mínima (VRAM) del trabajo y versión de drivers suficiente;
- fiabilidad del host ≥ 0,99 y ancho de banda anunciado ≥ 2.000 Mbps;
- contrato del host de al menos 1,5 × las horas del trabajo + 2 h.

GPU según la necesidad del entrenamiento

Trabajo	Qué necesita	Máquina elegida
CPT del flash 110M a 8k (v4)	Memoria para micro-lote 4 a 8k y máxima velocidad	GPU de 96 GB
CPT ronda 3 del 200M (continuación)	32 GB bastan con micro-lote 2 y los mismos tokens por paso	GPU de 32 GB
MLM del encoder hasta 8B tokens	Rendimiento sostenido (~244k tokens/s)	GPU de 32 GB
Embeddings fases 2-7 (lotes de 16k)	Más de 32 GB tras un OOM en el primer paso	GPU de 48 GB
Flash 8k: base, FIM y SFT	24 GB bastan	GPU de 24 GB
Tests (pytest)	CPU, ~40 s de principio a fin	Máquina CPU de 4 vCPU
Verificación masiva / revisores	Muchos núcleos, SSH	Máquina CPU de 16 núcleos

Ciclo de vida de un trabajo

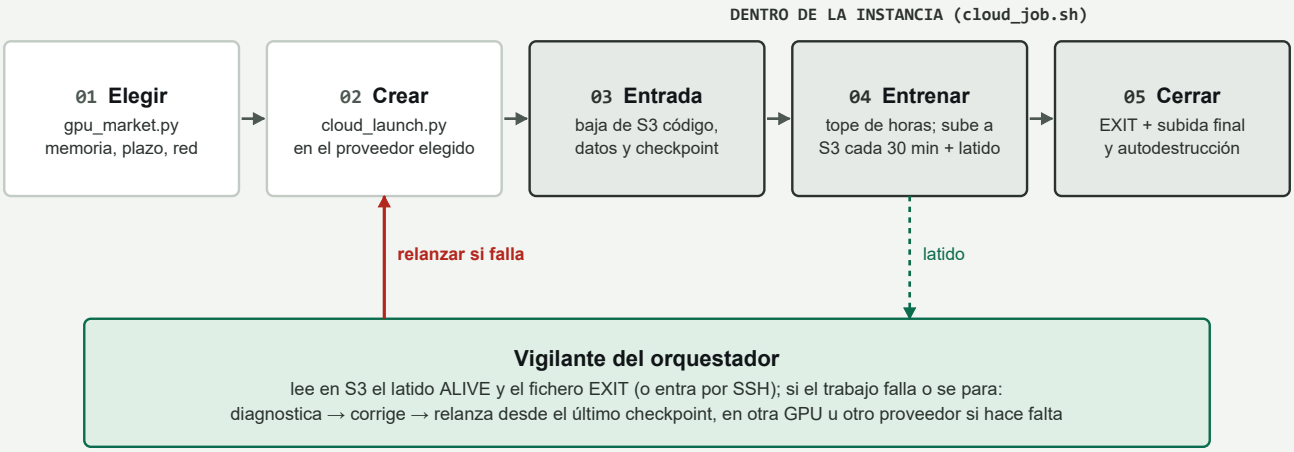


FIG. 4 Ciclo de vida de un trabajo. Si la instancia se reinicia y `EXIT` ya existe, no repite el trabajo; si falta, reanuda desde `last.pt`.

Recuperación automática: incidentes reales

Incidente	Causa	Respuesta
Fase 2 de embeddings: falta de memoria (OOM) en el primer paso, en 32 GB	La técnica de lotes grandes (GradCache) rehacía 2.048 queries de golpe	Trozos con tope de tokens y reintento automático partiendo el trozo; relanzado en una GPU de 48 GB desde el checkpoint. Pico real: 15,2 GB.
Un trabajo murió al compilar el modelo	La imagen base no traía las cabeceras de Python que necesita torch.compile	Imagen por defecto probada (torch 2.14 + toolchain); los scripts instalan lo que falte.

07 · RECETAS

Las recetas de entrenamiento

Generador kervyn-dev-flash

Transformer decoder (RoPE, RMSNorm, SwiGLU, atención con grupos de cabezas KV y embeddings atados) con tokenizer BPE byte-level propio de 24.576 tokens. El 200M tiene 32 capas de 768, 12 cabezas (4 KV) y contexto de 32k. Las etapas se encadenan y cada una parte del mejor checkpoint de la anterior:

Etapa	Qué se hace	Escala	Resultado al salir
01 Tokenizer	BPE byte-level entrenado con datos públicos; sin tokens desconocidos	24.576 ids	—
02 Preentreno	Corpus público v0.3 (texto ES/EN + código)	~1-2B tokens	bpb 0,99 (110M)
03 CPT código, ronda 1	Código público descontaminado + replay del corpus	1,97B tokens	HumanEval base 1,2 → 7,3 %
04 CPT ronda 2	Material nuevo + bloque DevOps; LR recalentado	2,36B tokens	HumanEval 10,4 %, MBPP 21,8 %
05 Crecer 110M → 200M	16 → 32 capas; las nuevas empiezan como identidad, así que el modelo crecido calcula exactamente lo mismo que el original	—	Más capacidad sin perder lo aprendido
06 CPT ronda 3	Tokens nuevos sobre el modelo crecido	4,16B tokens	HumanEval base 18,3 %
07 Contexto largo	8k → 32k (RoPE $\theta = 10^6$)	2.085 pasos	MBPP base 25,3 %; usa el contexto hasta 32k
08 SFT	Público (304k ejemplos) + tutores v7 (18.787) + robustez de chat	3.060 pasos	HumanEval 22,0 %, MBPP 35,4 % (chat)
09 Export	transformers + GGUF Q8_0 / Q4_K_M para llama.cpp	—	Ejecutable en CPU

Dos cambios de receta pesaron mucho: la **atención por documento** (cada fichero o conversación empaquetada solo se ve a sí misma) con ventanas alineadas al inicio de cada documento, y `torch.compile` con AdamW fusionado. En una GPU de 96 GB a 8k el rendimiento pasó de 118k a 259k tokens/s: la misma máquina entrena el doble en el mismo tiempo.

Embedder kervyn-embed v3

La v2 era un decoder adaptado y su base no había visto prosa en español. La v3 es un **encoder bidireccional propio** (arquitectura de encoder moderna, inicializada con pesos aleatorios, 12 capas de 768, 104,5M parámetros) que se carga con transformers o sentence-transformers sin código adicional.

Fase	Qué se hace	Escala	Resultado
00 Benchmark fiable	Empates cuentan en contra, XQuAD, IC95 por bootstrap, 150 queries propias	—	Medidas comparables
01 MLM desde cero	30 % de máscara; 45 % español, 35 % inglés, 20 % código	8B tokens	—
1b Puerta A/B	La misma fine-tune que v2 sobre la base nueva	1,5M pares	+3,43 nDCG@10: pasa
02 Contrastivo débil	Pares naturales escritos por personas, lotes de 16k con GradCache	25,4M pares · 1.220 pasos	Código 89,3 (supera a la referencia)
03 Negativos duros	Minados con el propio modelo Kervyn	2,05M pares	—
04 Supervisado	Pares públicos + 428k queries sintéticas de tutores	8.012 pasos	En curso
05 Benchmark final	Contra v2 y embedders de referencia del mercado	—	Pendiente
06 Versión CPU	12×384 destilado de la propia base Kervyn, Matryoshka 256, int8	—	Planificado

08 · EVALUACIÓN

Benchmarks y pruebas

Las pruebas son siempre las mismas para que cada versión se compare con todo el histórico. Los resultados crudos (una fila por problema) se guardan en S3 junto al checkpoint.

Prueba	Qué mide	Cómo
bpb de validación	Calidad de lenguaje, comparable entre tokenizers	Ventanas fijas de validación
HumanEval (164) y MBPP (257)	Escribir funciones correctas	pass@1 greedy, ejecutando los tests; modos base, chat y prefill
Cifra «limpia»	Lo mismo, sin problemas contaminados	16-gramas contra todo lo entrenado
eval-exec (41) y code bench (50)	JSON, SQL, Python, JS, HTML con prompts en español	Comprobaciones deterministas
Robustez de chat	Idioma, fugas del prompt de sistema, saludos, identidad, preguntar ante peticiones vagas, código	5 prompts de sistema, 3 de apps que el entrenamiento nunca vio
Contexto efectivo	Si el modelo aprovecha el contexto largo	bpb con prefijo real frente a aleatorio, de 1k a 32k
Evaluación cualitativa	Tareas reales variadas	48 tareas en español, revisores independientes y comparación a ciegas
Velocidad en CPU	tokens/s de prompt y de generación, RAM	llama.cpp con GGUF Q8_0/Q4_K_M, en máquinas CPU en la nube
Embeddings	nDCG@10, R@k, MRR, Spearman	Belebele ES/EN/ES→EN, XQuAD, CodeSearchNet, STS17 y kervyn-docs (150 queries propias)

Prueba	Qué mide	Cómo
Tests de software	Que el código hace lo que dice	pytest en un trabajo de CPU en la nube (~40 s)

- **Significación.** Las diferencias se comprueban con tests pareados (McNemar, test de signos) o con el IC95 por bootstrap de la diferencia query a query. Una mejora dentro del ruido no se da por buena.
- **Los números altos también se auditan.** El primer «buen» resultado del flash era memorización de plantillas (§10); desde entonces cada salto grande se revisa problema a problema.
- **Ajustes de inferencia medidos, no supuestos.** Por ejemplo, la penalización de repetición: 1,05 da el mejor HumanEval y 1,1 reduce los bucles en chat de 21 a 7 de 48 tareas, así que se recomiendan según el uso.

09 · RESULTADOS

Resultados reales, versión a versión

Generador: programar funciones correctas

Cada fila es un entrenamiento real con un cambio concreto. Del 7 al 9 de noviembre HumanEval pasó del 1,8 % al 22,0 % y MBPP del 0 % al 35,4 %, sin usar pesos de terceros.

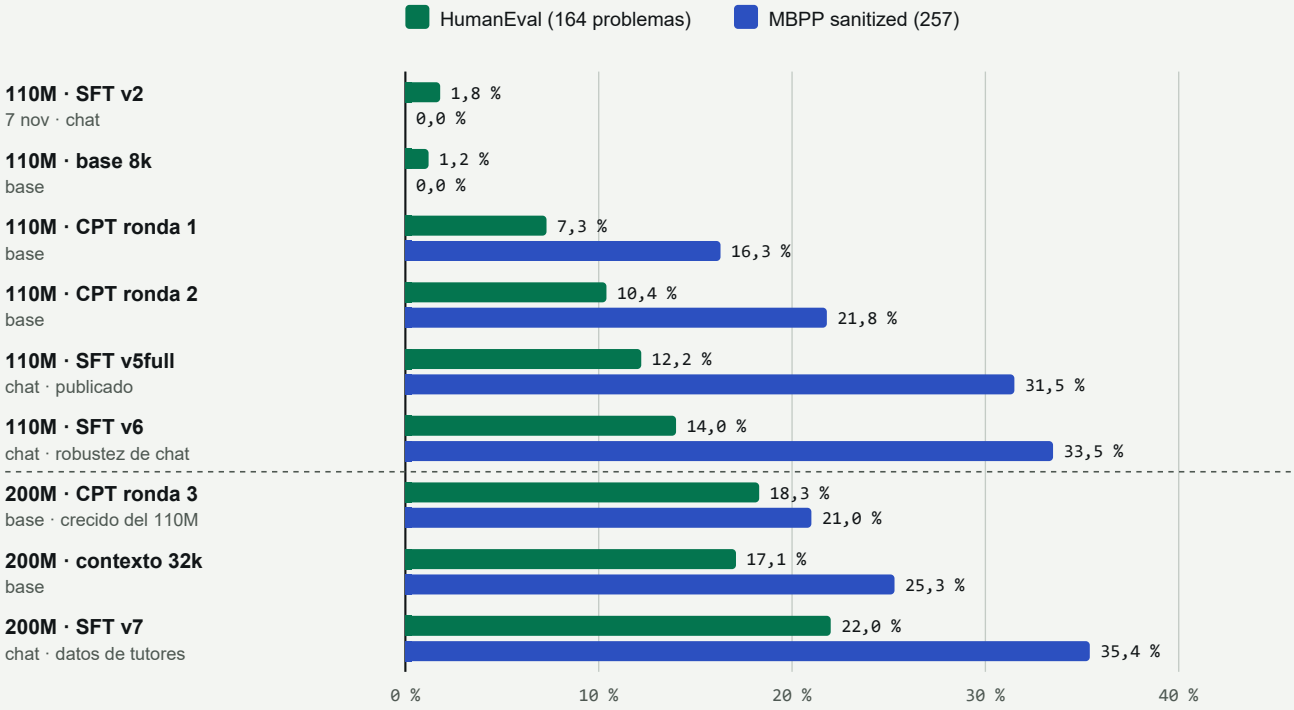


FIG. 5 pass@1 greedy. Los modelos base se evalúan en modo completion y los SFT en modo chat. La línea discontinua separa el 110M del 200M. Cifra «limpia» del 110M publicado (v5full): 12,5 % / 30,5 %. Fuente: `runs/*/bench-*.summary.json` en S3.

Generador: comportamiento en una app de chat

Métricas de `eval_chat.py` sumadas sobre cinco prompts de sistema, tres de ellos de apps que el modelo nunca vio en el entrenamiento. Muestran el arreglo del v6 y la regresión que trajo el v7 al crecer (ver §10, caso 3).

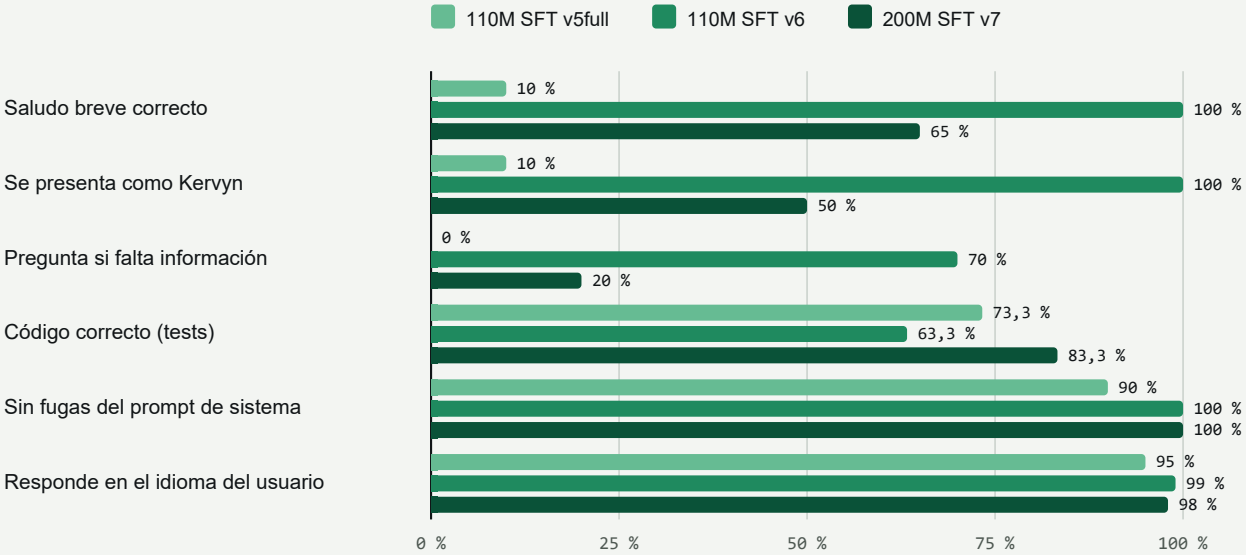


FIG. 6 Robustez de chat por versión (más oscuro = más reciente). Además, el v7 nunca se atribuye otra identidad (100/100): lo que pierde es presentarse por su nombre cuando se le pregunta.

Embedder: búsqueda en español, entre idiomas y en código

nDCG@10 (0-100, más es mejor). Cada punto es una versión; el rombo es un embedder multilingüe de mercado de tamaño parecido, como referencia. El eje empieza en 60 para que se vean las diferencias.

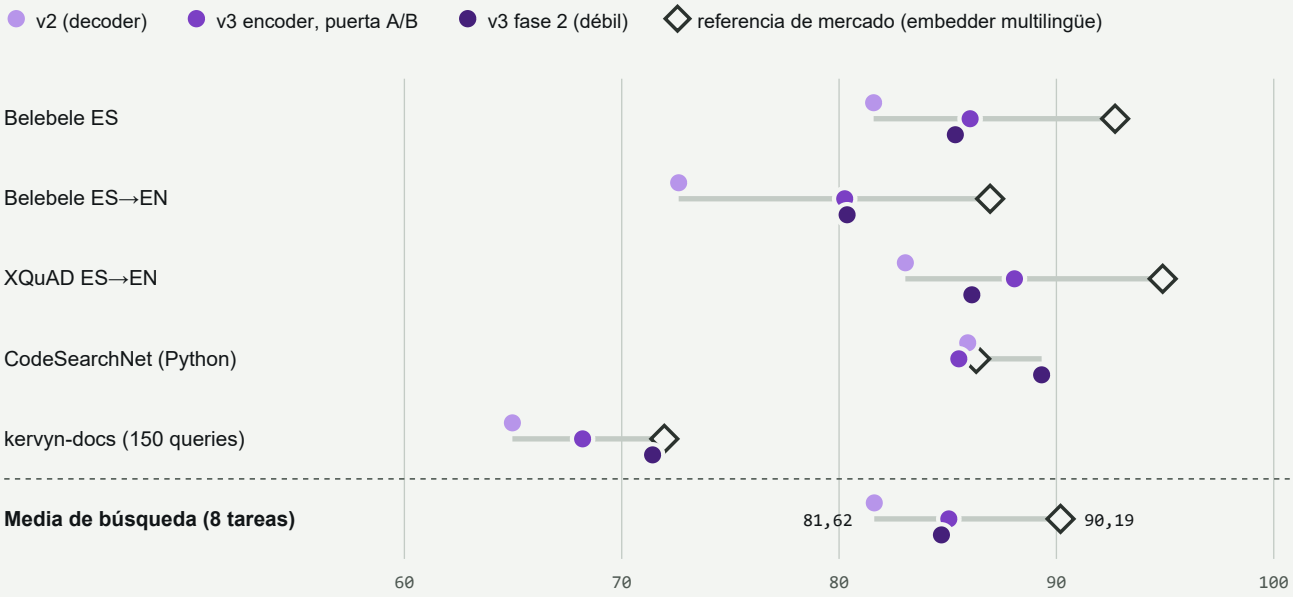


FIG. 7 kervyn-embed: v2 frente a las etapas de v3 ya medidas. Falta el benchmark final (fase 3). La v1 sacaba 38 en Belebele ES.

Tarea (nDCG@10)	v2	v3 puerta A/B	v3 fase 2	Referencia
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

Tarea (nDCG@10)	v2	v3 puerta A/B	v3 fase 2	Referencia
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
Media de búsqueda	**81,62**	**85,05**	**84,71**	**90,19**

Combinado con BM25 (búsqueda híbrida), el encoder de la puerta A/B saca 75,56 en kervyn-docs, por encima de la referencia (71,96).

Velocidad en CPU del flash 110M publicado

GGUF	Tamaño	Prompt a 512 tokens	Prompt a 8k	Generación
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 hilos, menos de 1 GB de RAM. La velocidad del 200M se medirá en una máquina CPU en la nube antes de publicarlo.

10 · RE-ENTRENAMIENTOS

Re-entrenamientos: cuatro casos reales

Así funciona el ciclo cuando algo sale mal. En los cuatro casos el problema se encontró midiendo, se diagnosticó antes de seguir y el siguiente entrenamiento cambió una cosa concreta.

CASO 01

«Los aciertos eran memorización» (flash 110M, SFT v2)

SÍNTOMA	11/50 en el code bench y 6/41 en eval-exec parecían un buen resultado para un modelo de 110M.
DIAGNÓSTICO	Cuatro auditorías en paralelo: los aciertos eran byte a byte iguales a ejemplos de entrenamiento; el SFT era un ~92 % plantillas con el 91,6 % de respuestas duplicadas; la descontaminación solo quitaba copias exactas; y el 82 % de la validación estaba en train. En HumanEval sacaba 1,8 % y en MBPP 0 %.
CAMBIO	Dos rondas de CPT con código público descontaminado, SFT solo con datasets públicos, atención por documento y benchmarks públicos con cifra «limpia».
RESULTADO	**HumanEval 1,8 → 12,2 % y MBPP 0 → 31,5 %** (limpio: 12,5 % / 30,5 %). En una comparación a ciegas con revisores independientes ganó 15-5 a la versión intermedia (test de signos p ≈ 0,04).

CASO 02

«En una app real responde sin sentido» (flash 110M publicado)

SÍNTOMA	Probado en una app de chat de escritorio, el modelo repetía el prompt de sistema y no sabía saludar ni decir quién era.
DIAGNÓSTICO	La app envía ~3.000 tokens de prompt de sistema (reglas y herramientas); el SFT solo tenía dos prompts de sistema cortos y ningún dato de identidad, saludo o multi-turno.
CAMBIO	SFT v6 con prompts de sistema de app, identidad, saludos y multi-turno, y una evaluación nueva con prompts de apps que el entrenamiento no ve.
RESULTADO	**Fugas del prompt 10 → 0 de 100; saludos 2 → 20 de 20; identidad 2 → 20 de 20**; HumanEval sube a 14,0 % y MBPP a 33,5 %.

CASO 03

«Más grande, pero ha perdido el chat» (flash 200M, SFT v7)

SÍNTOMA	El 200M programa mejor (HumanEval 22,0 %, código correcto 25/30) pero saluda bien 13/20, se presenta como Kervyn 10/20 y pregunta ante peticiones vagas solo 4/20.
DIAGNÓSTICO	En el SFT v7 el chat mezclado con el resto de familias pesaba menos del 1 % y el modelo más grande lo olvidó.
CAMBIO	v7b: subfamilias de chat pedidas por separado a los tutores, con cupo y guía propios: 1.095 saludos, 834 agradecimientos, 1.454 de identidad, 692 de capacidades y 1.758 peticiones vagas.
RESULTADO	En preparación (314.784 ejemplos). Decide la misma evaluación de chat, sin perder HumanEval.

CASO 04

«La base no sabía español» (kervyn-embed v2 → v3)

SÍNTOMA	La v2 quedaba 8,6 puntos por debajo de la referencia de mercado de media y peor en español → inglés (72,6 frente a 87,0).
DIAGNÓSTICO	La base era un decoder causal preentrenado sin prosa en español, y la fase contrastiva usó 1,5M pares frente a los ~1.000M de la referencia.
CAMBIO	Encoder bidireccional propio con 45 % de español en el preentreno, puerta A/B antes de escalar y después 25,4M pares naturales con lotes de 16k.
RESULTADO	**+3,43 de media y +7,64 en ES→EN** con la misma fine-tune; tras la fase 2 ya supera a la referencia en código (89,3 frente a 86,3). La distancia de media baja de 8,6 a 5,1 puntos.

11 · ESTADO

Estado actual y próximos pasos

Línea	Ahora	Siguiente
kervyn-embed v3	Fase 3 (supervisada) en el paso ~6.900 de 8.012	Benchmark final contra embedders de referencia; publicar o escalar; versión CPU 12×384
kervyn-dev-flash 200M	SFT v7 evaluado: mejor código, chat a recuperar	SFT v7b, evaluación de chat y velocidad en CPU; publicar si mejora sin romper el chat
kervyn-dev-pro 1B	Plan con arquitectura, datos y fases	Tokenizer de 48k y experimento de des-riesgo con puertas G0-G4

Cada línea sigue el mismo ciclo. El Paper n.º 2 de Kervyn recogerá estos resultados, medidos con las mismas pruebas para que se puedan comparar con todo lo anterior.

ANEXO

Glosario

Término	Significado
Preentreno / CPT	Entrenar el modelo a predecir el siguiente token sobre texto y código. CPT es «continued pretraining»: seguir preentrenando un modelo existente con datos nuevos.
SFT	Ajuste supervisado con conversaciones pregunta-respuesta para que el modelo siga instrucciones.
Checkpoint	Copia del estado del entrenamiento; `last.pt` permite reanudar y `best.pt` es el mejor en validación.
bpb	Bits por byte en validación: cuánto «se sorprende» el modelo; menos es mejor.
pass@1	Porcentaje de problemas resueltos al primer intento, ejecutando los tests.
HumanEval / MBPP	Benchmarks públicos de programación en Python (164 y 257 problemas).
Contaminación	Que el problema del benchmark (o su solución) esté en los datos de entrenamiento.
MLM	Preentreno de encoders: tapar tokens y aprender a recuperarlos.
Contrastivo	Entrenar para que una query y su documento queden cerca en el espacio de vectores.
Negativos duros	Documentos parecidos pero incorrectos, que obligan al modelo a afinar.
GradCache	Técnica para usar lotes de miles de pares sin que quepan a la vez en memoria.
nDCG@10	Calidad del ranking en los 10 primeros resultados de una búsqueda (0-100).
GGUF, Q8_0, Q4_K_M	Formato de llama.cpp y cuantizaciones a 8 y ~4 bits para ejecutar en CPU.
Orquestador	Pieza central del sistema: planifica, escribe el código, lanza y vigila los trabajos, evalúa y documenta. No escribe datos de entrenamiento.
Tutor	Genera o revisa datos de entrenamiento según una especificación del orquestador. No aporta pesos al modelo.

Desde Logroño con ❤ y mucho ☕

Luis Blanco Hijazo · linkedin.com/in/luisbh