INFORME TÉCNICO · SEPTIEMBRE DE 2026

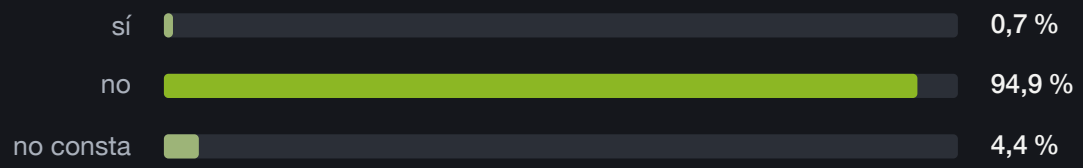
Modelos de decisión y ajuste fino

Una introducción para programadores que ya han usado modelos de lenguaje

RESPUESTA REAL DE KEV-4B

«El interesado presentó la solicitud el 15 de enero, pero no consta el pago de la tasa.»

¿Está la solicitud completa?



QUÉ INCLUYE

- Qué cambia frente a un chat, y la API con peticiones reales
- Cómo funcionan por dentro: logits, softmax, calibración
- El ajuste fino paso a paso, en una máquina propia
- Un caso medido con 350 textos del BOE
- Veredicto provisional sobre el estado de esta tecnología

Joaquín Herrero Pintado

Creative Codeworks

Antes de empezar

Este texto es para quien programa y ya ha usado modelos de lenguaje: has llamado a la API de un chat, has escrito *prompts*, quizá has pedido la respuesta en JSON y has tenido que validarla. No hace falta saber nada de aprendizaje automático.

Al terminar deberías saber:

- qué es un **modelo de decisión**, en qué se parece a un modelo de lenguaje y en qué no;
- cómo se le hace una pregunta y cómo leer lo que devuelve;
- qué significan sus probabilidades y por qué hay que comprobarlas antes de fiarse de ellas;
- qué es el **ajuste fino** (*fine-tuning*), cómo se hace con una máquina propia y qué hay que medir para saber si ha salido bien;
- qué pasó al hacerlo de verdad, con textos del BOE, incluidos los problemas que aparecieron.

Los primeros apartados explican los conceptos; después viene el ajuste fino, el caso medido con el BOE y, al final, usos, límites y un glosario.

El código, los datos de la prueba y un laboratorio para probar varios modelos en tu máquina están en este repositorio:

https://github.com/joakinen/toolkit-modelos-de-decision

Funciona como un **toolkit de evaluación de modelos de decisión**: se irá actualizando con los modelos nuevos que salgan. Al final del texto hay un veredicto provisional sobre el estado de esta tecnología. Esta es la versión del 28 de septiembre de 2026; la última está siempre en https://creativecodeworks.com/toolkit-modelos-de-decision.html.

Qué es un modelo de decisión

Un **modelo de decisión** es un modelo de inteligencia artificial que no escribe. Recibe un texto, una pregunta cerrada y la lista de respuestas posibles, y devuelve **una probabilidad para cada respuesta**. Si lo escribieras como una función, su firma sería esta:

```
def decidir(texto: str, pregunta: str,
            opciones: list[str]) -> dict[str, float]:
    """Una probabilidad por opción. Suman 1.
    Nunca devuelve una opción que no esté en la lista."""
```

Un ejemplo real, con Kev-4B (un modelo de decisión abierto que se ejecuta en un ordenador de sobremesa):

> **Texto:** «El interesado presentó la solicitud el 15 de enero, pero no consta el pago de la tasa.»
>
> **Pregunta:** ¿Está la solicitud completa?
>
> **Respuesta:** sí 0,7 % · no 94,9 % · no consta 4,4 %

No redacta una explicación ni puede contestar «depende». Solo reparte la probabilidad entre las opciones que le das.

Admite tres tipos de pregunta. Los nombres de la segunda columna son los que usa la API (vienen de Jev, el primer modelo de este tipo, y Kev los copia para ser compatible):

Tipo	Nombre en la API	Qué devuelve	Ejemplo
Elegir una opción	`choice`	Probabilidad de cada opción	¿Qué tipo de escrito es: solicitud, alegación, recurso u otro?
Puntuar en una escala	`score`	Probabilidad de cada nivel	¿Qué urgencia tiene: baja, media o alta?
Sí o no	`noul`	Probabilidad de «sí»	¿Menciona el texto un plazo?

A veces se les llama también modelos de «Sistema 1», por la distinción de Daniel Kahneman entre el pensamiento rápido e intuitivo (Sistema 1) y el lento y deliberado (Sistema 2). Un modelo de decisión hace juicios rápidos sobre un texto: no razona en varios pasos ni planifica.

Lo que ya sabes de los modelos de lenguaje, y lo que cambia

Un modelo de lenguaje (ChatGPT, Claude, Gemini, Llama, Qwen…) **genera texto**: calcula la probabilidad de cada posible siguiente trozo de palabra (*token*), elige uno, lo añade al texto y repite. Un modelo de decisión parte de uno de esos modelos, pero le quita la parte que escribe (se explica cómo en el apartado «Cómo funciona por dentro»). La tabla resume lo que eso cambia para quien lo usa desde código:

	Modelo de lenguaje	Modelo de decisión
Qué le envías	Un *prompt*: instrucciones, contenido y formato mezclados en texto libre	Campos fijos: el texto, la pregunta y las opciones
Qué devuelve	Texto, generado *token* a *token*	Un JSON con una probabilidad por opción, calculado de una vez
¿Puede salirse de las opciones?	Sí, aunque le pidas JSON: hay que validar la respuesta	No: la salida es siempre una de las opciones
La «temperatura»	Controla la aleatoriedad al elegir el siguiente *token*	No hay nada que elegir al azar; la temperatura corrige lo seguro que se muestra (se explica más abajo)
Coste	Pagas sobre todo los *tokens* que genera	Casi no genera nada: el coste es leer la entrada
Tiempo por consulta	Segundos	De décimas de segundo a un segundo, en un ordenador de sobremesa
Cómo se evalúa	Hay que valorar textos, a mano o con otro modelo	Se cuenta cuántas veces acierta

La consecuencia práctica: para **clasificar, encaminar o comprobar**, un modelo de decisión es más barato, más rápido y más fácil de controlar que un chat. Para **redactar, resumir o explicar**, no sirve.

¿No basta con pedirle a un chat que responda con una letra?

Es lo primero que se le ocurre a cualquiera que ha usado la API de un chat: escribir las opciones como A, B, C, pedir «responde solo con la letra» y leer la probabilidad que el modelo da a cada letra en el primer *token* (muchas APIs la devuelven con la opción `logprobs`). Funciona, y el laboratorio del repositorio lo usa para comparar un modelo de chat general de 9.000 millones de parámetros (Qwen3.5 9B) con los modelos de decisión.

Tiene tres problemas. El modelo no se entrenó para esto, así que su probabilidad sobre las letras no está pensada para significar «cuánto acierto»; depende de detalles del *prompt*, como el orden de las opciones; y sigue pagando el coste de un modelo grande. En la prueba con textos del BOE que se cuenta más abajo, ese modelo de 9.000 millones acierta el 66 %; un modelo de decisión de 800 millones, ajustado con ejemplos, acierta el 95 %.

Cómo se llama: la API

Kev, el modelo abierto que se usa en este texto, se sirve con un servidor HTTP local que imita la API de Jev. La petición es un JSON con el texto (`state`) y un diccionario de preguntas; cada pregunta lleva un identificador que eliges tú:

```
{
  "state": "El interesado presentó la solicitud el 15 de enero,
            pero no consta el pago de la tasa.",
  "questions": {
    "completa": {"type": "choice",
                 "instructions": "¿Está la solicitud completa?",
                 "criteria": {"sí": null, "no": null,
                              "no consta": "El texto no permite saberlo"}},
    "plazo":    {"type": "noul",
                 "instructions": "¿Menciona el texto un plazo?"},
    "urgencia": {"type": "score",
                 "instructions": "¿Qué urgencia tiene?",
                 "criteria": ["baja", "media", "alta"]}
  }
}
```

En `criteria` van las opciones. En `choice`, cada opción puede llevar una descripción corta (aquí, «no consta») o `null`. En `score`, las opciones son los niveles de la escala, de menor a mayor. En `noul` no hacen falta.

Desde Python es una llamada normal:

```
import requests

peticion = {...}   # el JSON de arriba
url = "http://127.0.0.1:8009/v1/systemone"   # Kev-4B servido en local
r = requests.post(url, json=peticion, timeout=60)
print(r.json()["answers"]["completa"]["probabilities"])
# {'sí': 0.0072, 'no': 0.9488, 'no consta': 0.044}
```

Esta es la respuesta completa que dio Kev-4B, recortada a lo esencial:


```
{
  "answers": {
    "completa": {"type": "choice", "choice": "no",
                 "probabilities": {"sí": 0.0072, "no": 0.9488,
                                   "no consta": 0.044}},
    "plazo":    {"type": "noul", "noul": 0.729},
    "urgencia": {"type": "score", "score": 0.7953,
                 "probabilities": {"0": 0.4677, "1": 0.2692, "2": 0.2631}}
  }
}
```


Cómo leerla:

- **`completa`**: elige «no» con un 94,9 %. Es la respuesta correcta y la da con mucha seguridad.
- **`plazo`**: devuelve la probabilidad de «sí», un 72,9 %. El texto trae una fecha, pero no un plazo, así que la respuesta correcta sería «no». El modelo se equivoca, pero con una seguridad moderada. Esa duda es información: una respuesta al 73 % no debería aceptarse sin que la mire una persona.
- **`urgencia`**: en una escala, devuelve la probabilidad de cada nivel (numerados desde 0) y su media (0,80, entre «baja» y «media»). El reparto (47 %, 27 %, 26 %) dice que no lo tiene claro, lo que tiene sentido: el texto no da pistas sobre la urgencia.

La misma petición a Kev-0.8B, el modelo pequeño, da en `completa` un 48 % a «no consta», un 29 % a «no» y un 23 % a «sí». Acierta menos y duda más. Es un patrón que se repite: los modelos más pequeños son más baratos y rápidos, pero dudan más.

Tres detalles prácticos:

- **Varias preguntas en una llamada.** El modelo lee el texto una vez y responde cada pregunta por separado, sin que la respuesta a una influya en las demás.
- **Las opciones son el nuevo *prompt*.** Escribirlas bien es lo que más influye en el resultado: que no se solapen, que cubran todos los casos y que haya un «no consta» cuando el texto puede no decir nada.
- **La respuesta trae también un campo `confidence`.** No es la probabilidad de acertar: es una transformación de la probabilidad más alta. Para decidir, usa las probabilidades.

Cómo funciona por dentro

No hace falta entender esto para usarlo, pero ayuda a entender sus límites y qué cambia al ajustarlo.

1. Se parte de un modelo de lenguaje ya entrenado. Kev-0.8B y Kev-4B parten de los modelos base de Qwen3.5. Un modelo de lenguaje ha leído enormes cantidades de texto y ya «entiende» el idioma: sabe qué es una negación, un plazo o una fecha. Esa comprensión es lo que se aprovecha.

2. Se le cambia la salida. El texto, la pregunta y las opciones se escriben en una sola secuencia con marcas:

```
<state> …el texto…
<q> ¿Está la solicitud completa?
<opt> sí </opt> <opt> no </opt> <opt> no consta </opt> <decide>
```

El modelo de lenguaje la lee entera, como leería un *prompt*, pero no se le pide que escriba nada. En lugar de la pieza que calcula el siguiente *token*, se le añade una pieza pequeña, la **cabeza**, que compara lo que el modelo «tiene en mente» al llegar a `<decide>` con lo que tenía al terminar cada opción, y da **una puntuación por opción**. A esas puntuaciones se les llama ***logits***. Después, una función llamada **softmax** las convierte en probabilidades que suman 1. En pseudocódigo:

```
# un número por opción
logits = [cabeza(estado_en_decide, estado_al_final_de(op))
          for op in opciones]
probabilidades = softmax(logits)

def softmax(z):
    e = [math.exp(x) for x in z]
    return [x / sum(e) for x in e]
```

Con tres opciones y *logits* 2,0, 0,5 y -1,0, softmax da 78,6 %, 17,5 % y 3,9 %. Cuanto mayor es la diferencia entre los *logits*, más seguro se muestra el modelo.

3. Se entrena con muchísimos ejemplos resueltos. Cada ejemplo es un texto, una pregunta, unas opciones y la respuesta correcta. Salen de colecciones públicas (noticias, reseñas, consultas de clientes, preguntas sobre un texto, inferencia) y de ejemplos generados con reglas, en los que la respuesta correcta se conoce por construcción (por ejemplo, si una fecha cae dentro de un plazo). El entrenamiento compara la probabilidad que el modelo dio a la respuesta correcta con la que debería haber dado, y ajusta los números del modelo para acercarlas. El error de cada ejemplo se mide con la **log-loss**: menos el logaritmo de la probabilidad que dio a la respuesta correcta.

Probabilidad dada a la respuesta correcta	Log-loss
99 %	0,01
90 %	0,11
50 %	0,69
10 %	2,30
1 %	4,61

Fíjate en la forma: dudar cuesta poco, pero **equivocarse con seguridad cuesta mucho**. Dar un 1 % a la respuesta correcta cuesta 40 veces más que darle un 90 %. Por eso el modelo aprende a repartir bien la probabilidad y no solo a acertar. La log-loss media sobre un conjunto de casos es también una de las medidas que se usan para comparar modelos.

Durante el entrenamiento se toman precauciones para que aprenda la tarea y no atajos:

- **Se baraja el orden de las opciones**, para que no aprenda que «la buena suele ser la primera».
- **Se incluyen casos en los que la respuesta no está en el texto**, para que aprenda a decir «no consta» en lugar de adivinar.

- **Se aparta una parte de los ejemplos que el modelo nunca ve al entrenar**, para medir con ellos cuánto acierta de verdad y para corregir al final lo seguro que se muestra (el apartado siguiente).

Todo esto lo hace quien publica el modelo. Lo que puedes hacer tú es el paso siguiente: ajustarlo con tus propios casos.

Probabilidad, calibración y umbrales

Lo más útil de estos modelos no es la respuesta, sino **cuánta seguridad tiene en ella**. Con una probabilidad fiable puedes repartir el trabajo con una regla sencilla:

```
p = max(respuesta["probabilities"].values())
if p >= 0.95:
    proponer_automaticamente(respuesta)   # la persona solo confirma
else:
    enviar_a_revision(respuesta)          # la persona decide
```

Para que esa regla funcione, la probabilidad tiene que significar lo que dice: de todas las veces que el modelo dice «95 %», debe acertar unas 95 de cada 100. Eso se llama **calibración**. Un modelo que acierta mucho pero falla a menudo con un 99 % de seguridad es peor para este uso que uno que acierta menos pero sabe cuándo duda, porque sus errores se cuelan por la puerta automática.

Cómo se mide. Se usan dos cifras en este texto:

- **Fallos con seguridad alta:** de las respuestas que el modelo da con un 90 % o más, cuántas son erróneas. Es la que más importa si vas a poner un umbral.
- **ECE** (*expected calibration error*, error de calibración esperado): se agrupan las respuestas por tramos de seguridad (del 90 al 100 %, del 80 al 90 %…) y se compara, en cada tramo, la seguridad media con el acierto real. La ECE es la media de esas diferencias. Cero es la calibración perfecta.

Cómo se corrige: la temperatura. Si un modelo se muestra demasiado seguro, se divide cada *logit* por un número T mayor que 1 antes de aplicar softmax:

```
probabilidades = softmax([z / T for z in logits])
```

Con los *logits* del ejemplo anterior (2,0, 0,5 y -1,0), con $T = 2$ las probabilidades pasan de 78,6 %, 17,5 % y 3,9 % a 59,0 %, 27,9 % y 13,2 %. La opción ganadora sigue siendo la misma, así que **el acierto no cambia**: solo cambia lo seguro que se muestra. T se elige con un conjunto de casos apartado, buscando la que da la menor log-loss. Kev trae ya la suya (2,41 en el 4B, 2,35 en el 0.8B).

Ojo con el nombre: en un modelo de lenguaje, la «temperatura» controla la aleatoriedad al generar. Aquí no se genera nada; es la misma operación matemática, pero sirve para corregir la seguridad, no para dar variedad.

El umbral se elige con datos, no a ojo. Con casos ya resueltos se mira qué porcentaje de error hay por encima de cada umbral posible y cuántos casos quedarían para revisión, y se elige el

equilibrio que convenga. Esos casos tienen que ser distintos de los que se usaron para entrenar y para calibrar: si no, el umbral se ajusta a ellos y promete menos error del que habrá.

Ajuste fino: enseñarle tus casos

Un modelo de decisión recién descargado sabe hacer juicios generales, pero no conoce los tipos de escrito, la terminología ni los criterios de tu organización. Con un modelo de lenguaje resolverías eso metiendo instrucciones y ejemplos en el *prompt*. Aquí no hay *prompt* libre donde meterlos: se le enseñan **entrenándolo un poco más** con casos propios. A todo entrenamiento que se hace sobre un modelo ya entrenado se le llama **post-entrenamiento** (*post-training*); la forma que está al alcance de una organización es el **ajuste fino** (*fine-tuning*).

Qué se entrena: LoRA

Un modelo como Kev-4B tiene unos 4.000 millones de números (los **pesos** o **parámetros**). Re-entrenarlos todos exige mucha memoria y mucho tiempo. La técnica habitual, **LoRA** (*low-rank adaptation*), congela todos esos pesos y entrena solo unas matrices pequeñas que se suman a algunas de las grandes:

```
W_efectiva = W_congelada + B · A
# A y B son estrechas: su «rango» (16 en Kev) fija su tamaño
```

Piensa en ello como un parche sobre un binario que no tocas: el modelo original queda intacto y lo que aprendes se guarda aparte. En la prueba de este texto, el parche ocupa 43 MB en el 0.8B y 130 MB en el 4B, en torno al 1 % de los parámetros. Junto al parche se entrena también la cabeza. Así, el ajuste cabe en una máquina propia.

Los datos: casos que ya existen

Cada decisión que una persona ha tomado y ha quedado registrada (el tipo asignado a un escrito, la unidad a la que se envió, si se pidió subsanación) es un ejemplo resuelto. No hay que etiquetar desde cero: hay que extraer y revisar. Cada caso es una línea de un fichero JSONL con el mismo formato que la petición a la API, más la respuesta correcta en `label`:

```
{"state": "... interpone recurso de alzada contra la resolución de ...",
 "questions": {"tipo": {
   "type": "choice",
   "instructions": "¿Qué tipo de escrito es?",
   "criteria": {"solicitud": null, "alegación": null,
                "recurso de alzada": null, "recurso de reposición": null,
                "consulta": null, "otro": null},
   "label": "recurso de alzada"}}}
```

En `choice`, `label` es el nombre de la opción; en `noul`, `true` o `false`; en `score`, la posición del nivel, empezando por 0.

El reparto: tres grupos que no se mezclan

Si pruebas tu código solo con los casos que tenías delante al escribirlo, pasará las pruebas aunque no funcione con casos nuevos. Con un modelo pasa lo mismo, y peor: puede **memorizar** los ejemplos. Por eso los casos se reparten en tres grupos:

Grupo	Parte orientativa	Para qué
Entrenamiento	70 %	Lo único que el modelo ve al aprender
Calibración	15 %	Para elegir la temperatura
Prueba	15 %	Para medir el resultado; no se mira hasta el final

Cuidado con las **fugas** (*data leakage*): si dos escritos del mismo expediente quedan uno en entrenamiento y otro en prueba, el modelo «reconoce» el caso y la medida sale mejor de lo que es. El reparto se hace por expediente, no por documento. Si los datos tienen fecha, lo más seguro es repartir por periodos: entrenar con lo antiguo y probar con lo reciente, que es lo que pasará en producción.

El entrenamiento y sus ajustes

Con Kev, entrenar es un comando (simplificado; el real añade los datos del modelo de partida):

```
python -m kev.train --init_from jaredpalmer/kev-4b \
  --data mis-casos/entrenamiento.jsonl \
  --suite evals/v7/decision-v7 --replay 160 \
  --epochs 3 --lr 2e-5 --batch 4 --lora 16 --out modelos/mi-ajuste
```

Qué significa cada ajuste:

- **`--init_from`**: se parte del modelo ya publicado, no del modelo base de Qwen, para conservar lo que ya sabe hacer.
- **`--epochs`** (épocas): cuántas pasadas completas se dan por los datos.
- **`--lr`** (*learning rate*, ritmo de aprendizaje): cuánto se mueven los pesos en cada paso. Si es demasiado alto, el modelo olvida lo que sabía; si es demasiado bajo, apenas aprende. Al partir de un modelo ya entrenado se usa uno pequeño.
- **`--batch`** (lote): cuántos ejemplos se procesan antes de mover los pesos.
- **`--replay`** (repaso): mezcla ejemplos del entrenamiento original con los tuyos. Un modelo ajustado solo con casos nuevos tiende a estropear lo que sabía (se llama **olvido catastrófico**); el repaso lo reduce, como mantener las pruebas de regresión mientras cambias el código.

Estos valores **se fijan antes de ver ningún resultado**. Si los vas cambiando hasta que la prueba sale bien, acabas ajustándolos a la prueba, igual que retocar un código hasta que pasa un test que falla de forma intermitente: el test pasa y el problema sigue ahí.

Después de entrenar: recalibrar y medir

Tras el ajuste, el modelo suele volverse demasiado seguro. Además, el entrenador de Kev guarda el modelo ajustado con temperatura 1, sin la corrección que traía el publicado. Hay que **recalibrarlo** con el grupo de calibración y después **medirlo** con el de prueba, comparando siempre con el modelo sin ajustar:

- **acierto en cada opción**, no solo el global: un modelo que dijera siempre «solicitud» podría acertar el 60 % y no servir para nada;
- **calibración**: si su seguridad se corresponde con su acierto;
- **olvido**: que siga respondiendo bien a preguntas ajenas a tu tarea que ya sabía responder, y que su seguridad siga siendo de fiar también ahí.

Si mejora de forma clara, se fija un umbral y se pasa a un piloto en el que el modelo propone y una persona confirma o corrige. Esas correcciones son, a su vez, nuevos ejemplos para el siguiente ajuste.

Cuántos casos hacen falta. La guía de Kev: de 100 a 200 sirven para probar el proceso, con resultados ruidosos; de 300 a 600, para un primer ajuste real; de 1.000 a 3.000, para algo que vaya a producción. Importa sobre todo tener bastantes **de cada opción**: si una opción es rara (un 3 % de los casos, pongamos), el modelo tenderá a ignorarla.

Un aviso útil: antes de ajustar, prueba el modelo **sin ajustar** con tu pregunta y tu grupo de prueba. A veces basta con escribir mejor las opciones, y el ajuste no compensa el trabajo.

Un caso medido: ¿en qué apartado del BOE se publica?

Para comprobar si todo esto funciona con textos administrativos reales, se ha seguido el proceso anterior con datos públicos del Boletín Oficial del Estado. Es una clasificación documental muy parecida a la de un registro de entrada: leer un texto y decir de qué tipo es. El código, los resultados y las instrucciones para repetirlo están en https://github.com/joakinen/toolkit-modelos-de-decision.

La pregunta. *¿En qué apartado del BOE se publica este texto?* Las opciones son las siete secciones del sumario:

Apartado	Qué contiene
I	Disposiciones generales (leyes, reales decretos, órdenes de alcance general)
II.A	Nombramientos, situaciones e incidencias del personal
II.B	Oposiciones y concursos
III	Otras disposiciones (actos concretos: convenios, subvenciones, resoluciones)
IV	Administración de Justicia
V.A	Anuncios de contratación del sector público
V.B	Otros anuncios oficiales

Los datos. Se descargaron de la API de datos abiertos del BOE. La respuesta correcta de cada texto es la sección en que lo publicó el propio BOE, así que es fiable por construcción. Al modelo se le da solo el cuerpo del texto, sin el título, que a menudo delata la sección. El reparto es por fechas, como se recomendaba más arriba:

- **Entrenamiento:** 1.400 textos de enero a junio de 2026, 200 por apartado.
- **Prueba:** 350 textos de julio y agosto de 2026, 50 por apartado.
- **Calibración:** 140 textos de septiembre de 2026, 20 por apartado (más casos generales; se explica más abajo).
- Como mucho tres textos casi idénticos por tipo (por ejemplo, los cambios diarios del euro), para que no dominen.

Los ajustes del entrenamiento (3 épocas, lote de 4, 160 ejemplos de repaso, LoRA de rango 16, ritmo 4e-5 en el 0.8B y 2e-5 en el 4B) se fijaron antes de medir nada.

Resultados en los 350 textos de prueba (acierto medio por apartado):

Modelo	Acierto	Log-loss
Kev-4B, ajustado con los 1.400 textos	**96 %**	**0,22**
Kev-0.8B, ajustado con los 1.400 textos	**95 %**	**0,35**
Réplica abierta de Jev de 2.000 millones de parámetros, sin ajustar	73 %	1,24
Kev-4B, sin ajustar	69 %	0,83
Modelo de chat general de 9.000 millones (Qwen3.5), con el truco de las letras	66 %	0,78
Réplica abierta de Jev de 800 millones, sin ajustar	64 %	1,10
Kev-0.8B, sin ajustar	49 %	1,32

«Acierto medio por apartado» es la media de los siete aciertos por apartado: así un modelo no puede sacar buena nota acertando solo los apartados fáciles.

Cómo saber si una diferencia es real. Con 350 casos, parte de cualquier diferencia es suerte. Para medirlo se usa un ***bootstrap***: se sortean 2.000 veces 350 casos con repetición entre los de la prueba, se recalcula la diferencia en cada sorteo y se mira entre qué valores cae el 95 % de las veces. Ese es el **intervalo de confianza del 95 %**. Si no incluye el cero, la diferencia no se explica por azar.

Lo que enseña:

- **Con datos suficientes, el ajuste funciona.** Kev-0.8B pasa del 49 % al 95 %: +45 puntos, con un intervalo de confianza del 95 % de +41 a +49. Kev-4B pasa del 69 % al 96 % (+27 puntos; de +24 a +31).
- **Sin ajustar, ningún modelo conoce las convenciones del BOE.** Ninguno reconoce más de 5 de las 50 disposiciones generales: las confunden con «otras disposiciones». Distinguir una norma de alcance general de un acto concreto no se deduce del texto; se aprende con ejemplos. El 0.8B ajustado acierta 42 de 50, y el 4B ajustado, 47.
- **Un modelo pequeño y ajustado supera a uno grande sin ajustar.** El 0.8B ajustado tiene 800 millones de parámetros; el modelo general de 9.000 millones se queda en el 66 % y tarda unas trece veces más por texto que Kev-0.8B.
- **No es memoria.** Revisando a mano una muestra de aciertos y separando los textos cuyo tipo de título aparece en el entrenamiento de los que no, el 4B ajustado acierta casi igual en los dos grupos (96,6 % en los conocidos y 95,9 % en los nuevos). Si hubiera memorizado, acertaría mucho más en los conocidos. Donde falla es en fronteras dudosas de verdad, sobre todo entre disposiciones generales y otras disposiciones.
- **Fuera del BOE acierta lo mismo, pero ya no sabe cuándo dudar.** Se explica más abajo, en «Lo que pasa fuera del BOE».

Lo que cuesta el ajuste (medido por el propio entrenador de Kev, en un Mac mini con chip M4 Pro y 24 GB de memoria):

	Kev-0.8B	Kev-4B
Tiempo de ajuste	3 h 29 min	11 h 35 min
Ejemplos propios	1.400, más 160 generales de repaso	Los mismos
Pasadas por los datos	3 (4.680 ejemplos procesados)	Las mismas
Segundos por ejemplo	2,68	8,91
Memoria máxima de GPU	3,3 GB	9,2 GB
Qué se entrena	En torno al 1 % de los parámetros (LoRA)	Lo mismo, con la base en media precisión

A eso hay que sumar reunir los datos: descargar los 1.400 textos del BOE tardó unos 27 minutos. El tiempo de ajuste crece en proporción a los ejemplos y a las pasadas, y con el tamaño del modelo. Con el 4B, un equipo de 24 GB está en su límite: llegó a usar unos 14 GB de intercambio a disco. Todo se hizo sin enviar nada fuera de la máquina.

Lo que pasa fuera del BOE

Para ver si el ajuste estropea lo que el modelo ya sabía, se le hicieron antes y después preguntas ajenas al BOE: 560 en inglés, de colecciones públicas (noticias, reseñas, consultas de clientes, inferencia) que no se usaron al ajustar, y 150 en español (tres colecciones públicas etiquetadas por personas). Se mira el acierto y, sobre todo, cuántas de las respuestas que da con un 90 % de seguridad o más resultan erróneas:

En las 560 preguntas en inglés	Kev-0.8B	Kev-4B
Acierto, antes y después del ajuste	83 % y 82 %	87 % y 86 %
Respuestas con ≥ 90 % que fallan, antes del ajuste	4 de 298 (1 %)	0 de 338 (0 %)
Después del ajuste	88 de 530 (17 %)	53 de 512 (10 %)
Después de recalibrar	30 de 425 (7 %)	11 de 399 (3 %)

El acierto no cambia más de lo que cambia por azar. Lo que cambia es la seguridad: el ajustado dice «90 %» a casi todo, también cuando se equivoca. Si se hubiera puesto en producción con la regla del umbral del apartado anterior, habría dejado pasar como seguras muchas respuestas erróneas. En español pasa lo mismo, y más acusado: con el 4B, fallan 7 de 58 respuestas seguras antes del ajuste, 35 de 132 después y 11 de 81 recalibrado.

La recalibración. Se hizo como se explicó en el apartado «Probabilidad, calibración y umbrales», con 400 casos que no se usan para entrenar ni para medir: 140 textos del BOE de septiembre,

200 preguntas generales y 60 en español. La temperatura que mejor funcionó es 4, bastante más alta que la del modelo publicado:

- **En el 4B funciona.** En inglés, las respuestas seguras que fallan bajan del 10 % al 3 % (el original tenía un 0 %), y en la prueba del BOE fallan 3 de 289. En español mejora mucho, pero sigue peor que antes del ajuste.
- **En el 0.8B no basta.** Necesitaría una temperatura de más de 7, por encima del máximo que prueba la herramienta de Kev, y cada tipo de pregunta pide una distinta: unos 4 el BOE, 7 las preguntas generales y 18 el español. Una sola temperatura no puede corregir las tres a la vez. No se le deberían fijar umbrales.

La lección es general: **el ajuste no hace olvidar cómo responder, pero sí cuándo dudar**, y eso solo se ve midiendo la calibración fuera de la tarea ajustada, no solo en ella. La primera versión de este experimento midió el olvido con 12 preguntas y concluyó que no había ningún problema; con 710 apareció.

Y lo que pasa con pocos datos. En otra prueba, con 100 ejemplos de los que solo 17 eran de la opción difícil, el ajuste no se distinguió del azar: el modelo aprendió a responder casi siempre la opción mayoritaria. La diferencia entre un caso y otro no está en el modelo ni en la máquina, sino en tener **unos cientos de ejemplos de cada opción**.

Límites de esta medida. Es un solo periodo de prueba (dos meses) y una sola pregunta. Aún no se ha fijado ningún umbral: debería salir de otro conjunto aparte, distinto del de calibración y del de prueba. El control en español es pequeño (150 preguntas). Y los textos del BOE están más normalizados que los escritos que llegan a un registro, que serían más variados.

Fuente de los datos: Agencia Estatal Boletín Oficial del Estado (boe.es), reutilizados según sus condiciones de datos abiertos.

Qué tareas hacen bien

Todo lo que se pueda formular como una pregunta cerrada sobre un texto:

- **Clasificar:** qué tipo de documento es, a qué materia pertenece.
- **Encaminar:** a qué unidad, cola o persona debe ir.
- **Comprobar:** si falta un documento, si se cumple una condición, si se menciona un requisito.
- **Detectar:** urgencia, reclamación, queja, plazo, datos personales.
- **Priorizar:** puntuar para ordenar una cola de trabajo.
- **Verificar a otro sistema:** comprobar si la respuesta que ha dado otro programa (o un chat) es correcta o está sustentada en el texto.
- **Abstenerse:** responder «no consta» cuando el texto no dice nada, en lugar de suponer.

Posibles usos en una administración

Algunos ejemplos, escritos como la pregunta que se le haría al modelo:

Uso	Pregunta	Opciones
Registro de entrada	¿Qué tipo de escrito es?	solicitud · alegación · recurso · consulta · otro
Reparto	¿Qué unidad debe tramitarlo?	la lista de unidades
Completitud	¿Aporta la documentación obligatoria?	sí · no · no consta
Plazos	¿Se presentó dentro de plazo según el texto?	sí · no · no consta
Urgencia	¿Con qué urgencia hay que atenderlo?	escala del 1 al 5
Consultas	¿La respuesta a esta consulta está en la documentación publicada?	sí · no
Calidad	¿Cita la resolución la norma aplicable?	sí · no
Revisión	¿Contiene datos personales que haya que anonimizar?	sí · no · no consta

En todos ellos el modelo no sustituye a quien tramita: **ordena, filtra y señala**, y deja la decisión a una persona cuando hay duda o cuando la decisión tiene efectos sobre alguien.

También encajan bien **delante de un modelo de lenguaje**: el modelo de decisión, barato y rápido, clasifica todas las entradas, y solo las que lo necesitan pasan a un chat, más caro, para redactar un borrador.

Qué no hacen, y con qué cuidado usarlos

- **No redactan, no resumen, no explican.** Solo eligen entre opciones.
- **Dependen de cómo se escriban las opciones.** Opciones ambiguas o que se solapan dan resultados pobres.
- **Fallan con la ironía, los dobles sentidos y los razonamientos largos** (contar días hábiles, encadenar varias condiciones). Ahí conviene que el modelo dude, y medir si lo hace.
- **Hay que medirlos con casos propios.** Las cifras de quien publica un modelo se obtienen con sus datos, no con los tuyos.
- **Ajustarlos tiene coste y riesgos.** Con pocos ejemplos aprenden poco, y el ajuste puede estropear la calibración aunque el acierto no baje.
- **Las decisiones con efectos sobre personas requieren intervención humana.** El Reglamento General de Protección de Datos (artículo 22) limita las decisiones basadas únicamente en tratamiento automatizado, y la Ley 40/2015 (artículo 41) exige que la actuación administrativa automatizada tenga un órgano responsable definido. Usados como apoyo a la tramitación, estos límites se respetan con facilidad; usados para resolver, no.

Qué modelos existen

La idea la popularizó **Jev**, de la empresa TypeSafe AI (2026). Es un modelo cerrado: solo se puede usar como servicio en la nube, enviando los textos a sus servidores. Es muy barato por consulta, pero implica que los datos salen de la organización.

Poco después aparecieron **modelos abiertos** con el mismo planteamiento. Uno de ellos es **Kev**, con licencia Apache-2.0 y varios tamaños (de 0,8 a 27 mil millones de parámetros). Los pequeños funcionan en un servidor corriente o en un ordenador de sobremesa con buena memoria, sin conexión a internet. Eso permite:

- que **ningún dato salga** de la propia infraestructura;
- **no depender de un proveedor** ni de su precio;
- **ajustarlos con los propios casos**, con herramientas libres.

Son proyectos recientes y cambian rápido: conviene tratarlos como tecnología en evaluación, no como producto maduro.

Cómo empezar

1. **Pruébalo.** Clona el repositorio, arranca el laboratorio y un servidor de Kev (el README explica cómo) y hazle tus propias preguntas con el formulario o desde código.
2. **Elige una sola decisión** concreta y frecuente, que hoy se tome a mano, y escribe la pregunta y sus opciones exactamente como las usarías.
3. **Mide el modelo sin ajustar** con 100 o 200 casos ya resueltos por personas: acierto en cada opción y calibración.
4. **Si no basta, ajústalo.** Reúne unos cientos de casos de cada opción, repártelos en entrenamiento, calibración y prueba sin fugas, fija los ajustes antes de mirar, entrena, recalibra y mide, también fuera de tu tarea.
5. **Fija un umbral** con casos apartados y haz un piloto en el que el modelo solo propone y una persona revisa.
6. **Decide con los datos del piloto** si merece la pena seguir.

Veredicto provisional

El repositorio que acompaña a este texto es un **toolkit de evaluación de modelos de decisión**: un laboratorio para comparar modelos con los mismos casos, la prueba del BOE, los controles de olvido y la recalibración. La idea es repetir las mismas pruebas con cada modelo de decisión nuevo que salga, así que esta valoración es **provisional**: refleja el estado de la tecnología a 28 de septiembre de 2026, con los modelos probados hasta ahora (Kev-0.8B y Kev-4B, sin ajustar y ajustados; las réplicas abiertas de Jev de 800 y 2.000 millones de parámetros; y un modelo de chat general, Qwen3.5 9B, como referencia). El Jev original no se ha probado, porque exige enviar los textos a sus servidores.

En una frase: los modelos de decisión abiertos **ya son usables para clasificar y encaminar documentos con revisión humana**, en una máquina propia, siempre que se ajusten con unos cientos de ejemplos de cada opción y se recalibren antes de fiarse de su seguridad. **No lo son** sin ajustar para tareas con convenciones propias, ni para decidir solos.

Aspecto	Valoración	Por qué
Integración en una aplicación	**Buena**	Devuelven datos, no texto; no pueden salirse de las opciones; la API es sencilla y estable
Acierto sin ajustar en una tarea propia	**Insuficiente**	Del 49 % al 73 % en el BOE: no conocen las convenciones propias de un dominio
Acierto ajustados, con datos suficientes	**Muy bueno**	95 % y 96 % en el BOE con 1.400 ejemplos, por encima de modelos diez veces mayores
Con pocos datos	**No funciona**	Con 17 ejemplos de la opción difícil, el ajuste no se distinguió del azar
Fiabilidad de su seguridad	**Frágil**	El ajuste la estropea fuera de la tarea; recalibrando se recupera en el 4B, no en el 0.8B
Coste y soberanía	**Muy favorable**	Un ordenador de sobremesa, sin enviar nada fuera; ajuste en horas; respuesta en décimas de segundo
Español	**Aceptable, poco medido**	Funcionan, pero se ha medido menos y su calibración empeora más que en inglés
Madurez	**Baja**	Proyectos de semanas (la versión de Kev probada es del 24 de septiembre de 2026); sus herramientas tienen límites que no avisan

Sobre la madurez, dos ejemplos encontrados por el camino: el evaluador de Kev descartaba en silencio los textos largos (182 de 350 en la prueba del BOE), y su herramienta de calibración tiene un tope que el 0.8B ajustado supera. Nada de eso impide usarlos, pero obliga a medir con cuidado y a no dar por buenas las cifras de nadie, incluidas las de este texto.

Qué modelo elegir hoy. Kev-4B, ajustado con tus casos y recalibrado. Kev-0.8B solo si la máquina no da para más, y sin fijarle umbrales. Para una primera prueba sin ajustar, la réplica abierta de Jev de 2.000 millones es la que mejor rinde.

Qué falta para pasar de provisional a firme:

- probarlo con un lote real de expedientes con la respuesta conocida;
- fijar un umbral con casos apartados y medir cuánto trabajo ahorra y cuántos errores deja pasar;
- repetir las pruebas con cada modelo nuevo que salga, con los mismos datos, para ver si la tecnología madura.

Glosario

Modelo de decisión. Modelo que, dado un texto y una pregunta con opciones cerradas, devuelve una probabilidad por opción. No genera texto.

Modelo de lenguaje (LLM). Modelo que genera texto *token* a *token*: los chats como ChatGPT o Claude.

***Token*.** Trozo de palabra con el que trabajan los modelos de lenguaje.

***Prompt*.** Texto libre con instrucciones y contenido que se le escribe a un chat. Los modelos de decisión no lo usan: reciben campos fijos (texto, pregunta, opciones).

Cabeza. Pieza pequeña que se añade a un modelo de lenguaje para que, en vez de escribir, puntúe cada opción.

***Logit*.** Puntuación sin normalizar que el modelo da a cada opción. Softmax los convierte en probabilidades.

Softmax. Función que convierte una lista de números en probabilidades que suman 1: eleva *e* a cada número y divide por la suma.

Log-loss. Menos el logaritmo de la probabilidad dada a la respuesta correcta, en media. Cuanto más baja, mejor; castiga mucho equivocarse con seguridad.

Calibración. Grado en que la probabilidad que da el modelo coincide con su acierto real. Bien calibrado: cuando dice 80 %, acierta 8 de cada 10.

ECE. Error de calibración esperado: diferencia media entre la seguridad y el acierto, por tramos de seguridad.

Temperatura. Número por el que se dividen los *logits* antes de softmax. En un modelo de decisión corrige lo seguro que se muestra sin cambiar la respuesta; en un modelo de lenguaje controla la aleatoriedad al generar.

Umbral. Probabilidad a partir de la cual se acepta la respuesta del modelo sin revisión.

Post-entrenamiento. Cualquier entrenamiento que se hace sobre un modelo ya entrenado. El ajuste fino es un caso.

Ajuste fino. Seguir entrenando un modelo ya hecho con ejemplos propios, para que responda mejor en una tarea concreta.

LoRA. Técnica de ajuste fino que congela el modelo y entrena solo unas matrices pequeñas que se suman a las grandes. Reduce mucho la memoria y el tiempo necesarios.

Época. Una pasada completa por los datos de entrenamiento.

Ritmo de aprendizaje (*learning rate*). Cuánto se mueven los pesos en cada paso del entrenamiento.

Repaso (*replay*). Mezclar ejemplos del entrenamiento original con los nuevos para que el modelo no olvide lo que sabía.

Olvido catastrófico. Cuando un modelo, al aprender algo nuevo, empeora en lo que ya sabía hacer.

Fuga de datos (*data leakage*). Cuando información de la prueba se cuela en el entrenamiento y la medida sale mejor de lo que es.

***Bootstrap*.** Forma de estimar cuánto se debe al azar una cifra: se repite el cálculo sobre muchos sorteos con repetición de los mismos casos.

Intervalo de confianza del 95 %. Rango en el que cae la cifra en el 95 % de esos sorteos. Si una diferencia entre dos modelos no incluye el cero, no se explica por azar.

Pesos abiertos. Modelo cuyos ficheros se publican y se pueden descargar y ejecutar en máquinas propias.

GB y GiB. Dos unidades de memoria que se confunden a menudo. Un gigabyte (GB) son mil millones de bytes (10^9); un gibibyte (GiB), 1.073.741.824 bytes (2^{30}), un 7,4 % más. Los fabricantes de discos y macOS usan GB; Windows y muchos programas muestran GiB aunque escriban «GB». Las memorias de este informe están en GB decimales, salvo los «24 GB» del equipo, que son la cifra comercial de su memoria y equivalen a 24 GiB. La norma que distingue ambas unidades es la ISO/IEC 80000-13.

Créditos

Modelos de decisión y ajuste fino
Una introducción para programadores que ya han usado modelos de lenguaje

Joaquín Herrero Pintado
Creative Codeworks

Primera versión · 28 de septiembre de 2026
Última versión: creativecodeworks.com/toolkit-modelos-de-decision.html
Toolkit de evaluación de modelos de decisión: github.com/joakinen/toolkit-modelos-de-decision
creativecodeworks.com

Los datos de la prueba proceden de la Agencia Estatal Boletín Oficial del Estado (boe.es), reutilizados según sus condiciones de datos abiertos. Los modelos probados son de sus autores: Kev, de Jared Palmer; las réplicas abiertas de Jev, de chaoliangUNSW; Qwen3.5, de Alibaba. Jev es un modelo de TypeSafe AI, que no está relacionada con este trabajo.

Este informe y el toolkit que lo acompaña se han hecho con la asistencia de **Claude Code** (Anthropic). La herramienta se usó para escribir y ejecutar los programas de evaluación, lanzar los ajustes y las medidas, contrastar cada cifra con los datos de los que sale, revisar a mano muestras de resultados, redactar borradores del texto y maquetarlo. Las preguntas, el criterio sobre qué medir y qué dar por bueno, y la decisión de publicar también lo que corrige versiones anteriores son míos; míos son también los errores.

Este informe se actualizará cada vez que el toolkit evalúe un modelo de decisión nuevo.