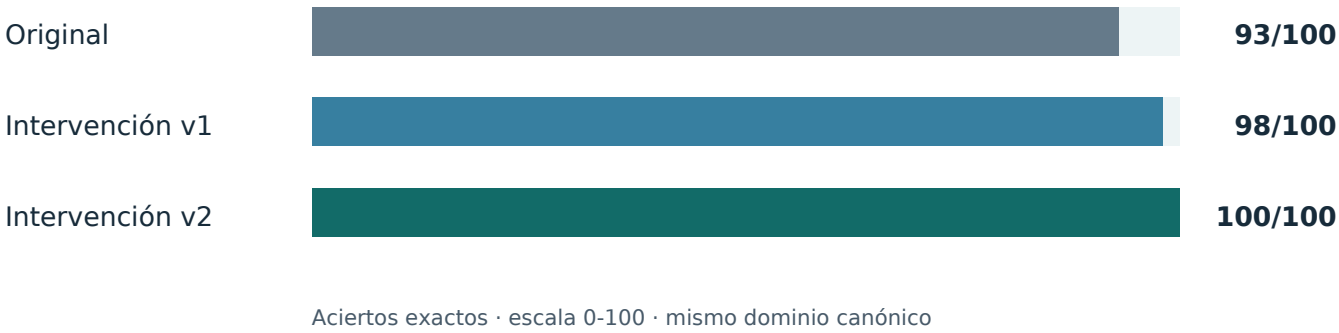
Reparación de MiniSum mediante edición directa de pesos

De 93/100 a 100/100, sin entrenamiento por gradientes

Informe técnico integrado de las intervenciones v1 y v2



Modelo de Edgar
MiniSum-Research · semilla 42 · época 210
26.432 parámetros · 609 valores editados · misma arquitectura

Diagnóstico, hipótesis, pruebas rechazadas, parche final y controles de reversión. Incluye el catálogo de 106 variantes y las 100 respuestas finales.

4 de noviembre de 2026
Agente editor: GPT-6 Astra en T3 Code
Versión documental 1.0

Contenido

1. Resultado, alcance y lectura del informe 3

2. Procedencia, restricciones y calidad de la evidencia 4

3. Arquitectura, representación y ejecución 6

4. Diagnóstico inicial y estados de referencia 8

5. Primera intervención: del diagnóstico a 98/100 9

6. Segunda intervención: señales más selectivas hasta 100/100 12

7. Verificación de la reparación y evidencia causal 15

8. Parche completo, tamaño e integridad 16

9. Coste, entorno y operaciones de entrenamiento 17

10. Qué demuestran los resultados y qué queda abierto 18

11. Archivos, fuentes y trazabilidad 19

12. Reproducción sin alterar el laboratorio histórico 20

13. Procedimiento de referencia para futuras intervenciones 21

Anexo A. Catálogo de las 106 variantes 23

Anexo B. Respuestas de las 100 sumas 27

Anexo C. Contenido del paquete de referencia 31

Los apartados 5 y 6 reconstruyen la exploración en orden. El apartado 8 contiene el parche completo; el 13 propone un procedimiento para futuras reparaciones. Los anexos conservan las pruebas y respuestas, incluidos los resultados descartados.

1. Resultado, alcance y lectura del informe

El checkpoint original producía siete respuestas incorrectas dentro de las 100 sumas de dos operandos de una cifra. La primera intervención corrigió cinco sin perder aciertos; la segunda corrigió los dos restantes. Se conservó la arquitectura y se cambiaron directamente valores de tres tensores.

Estado	Aciertos exactos	Pesos distintos del original	Nuevos pasos de entrenamiento
Original B0	93/100	0	0
Intervención v1, B3	98/100	576	0
Intervención v2, B5	100/100	609	0

El modelo tiene 26.432 parámetros. La primera edición afecta al 2,1791768 %; la segunda añade 33 parámetros, un 0,1248487 % del total. La diferencia final acumulada es del 2,3040254 %. No se añadieron capas, neuronas, adaptadores ni tablas de resultados. El porcentaje describe cuántos valores cambian, no cuán pequeños son sus cambios: algunos se multiplican por cuatro y uno cambia de signo.

Resultado comprobado: con generación greedy, formato d+d= y operandos de 0 a 9, la v2 guardada y recargada produce los dígitos correctos y el token EOS en los 100 casos. El código de inferencia histórico obtiene las mismas respuestas. La implementación del navegador también coincide en las secuencias evaluadas.

Alcance de la afirmación: reparación funcional completa del dominio canónico conocido. No demuestra una regla aritmética general, funcionamiento con operandos de dos cifras, otros formatos, cuantización o muestreo aleatorio. Tampoco constituye un test reservado: las 100 entradas se observaron durante el diagnóstico y la selección de cambios.

Para reproducir sólo la reparación, leer los apartados 8 y 12. Para comprender cómo se llegó a ella, leer los apartados 4 a 7 en orden. Para trasladar el método a otro modelo, usar el apartado 13. El anexo A enumera las 106 variantes exploradas, no sólo las que se conservaron.

2. Procedencia, restricciones y calidad de la evidencia

2.1 Archivo recibido e identificación

Edgar adjuntó `MiniSum-Research-seed42-epoca210.pt`, de 115.829 bytes. Se calculó su SHA-256 y se confirmó que coincidía con el checkpoint Research histórico de la máquina. El objeto es un `state_dict` de PyTorch, no un programa de inferencia autónomo. Su carga requiere reconstruir la arquitectura y usar el tokenizador correcto.

El laboratorio histórico está en `/home/edgar/Proyectos/MiniSumAI`. Se consultaron su arquitectura, tokenizador, inferencia e informes. Las intervenciones se realizaron sobre copias del adjunto en carpetas de resultados separadas. No se reemplazó el Research histórico ni el Deploy, no se modificaron sus servicios y no se cargaron pesos del modelo Deploy para obtener la reparación.

2.2 Restricciones aplicadas

• Sin reentrenamiento ni fine tuning convencional: no se calcularon gradientes, no se ejecutó `backward`, no se usó un optimizador y no se minimizaron pérdidas mediante actualizaciones automáticas.

• Los scripts de diagnóstico desactivan gradientes; los modelos cargados para inferencia usan `eval()` y `requires_grad_(False)`. La ruta de generación emplea `torch.inference_mode()`.

• Cada candidata parte de una base declarada. Se modifica una copia y se comprueban las 100 sumas, evitando acumular cambios accidentales entre candidatas.

• Inferencia exclusivamente neuronal. La aritmética para generar etiquetas o comprobar respuestas está en el evaluador y no decide qué devuelve el modelo.

• Original, v1, v2 y resultados rechazados se conservan como evidencia. No se instalaron dependencias para las intervenciones ni se apagaron servicios.

Es importante precisar el vocabulario: no se realizó entrenamiento mediante un optimizador, pero **sí hubo búsqueda empírica y selección manual guiada por resultados**. En un sentido amplio, se trata de optimización sin gradientes dirigida por un agente. No sería correcto describir el procedimiento como ajeno a los datos, libre de ajuste a las pruebas o completamente deducido en un único intento.

2.3 Qué está preservado y qué no

Se preservan los checkpoints, scripts de edición final, arquitectura, tokenizador, respuestas finales con tokens y logits, activaciones seleccionadas, resultados de las candidatas y hashes. La auditoría para este informe comprobó 71 entradas de los manifiestos de ambas rondas: todas coinciden con sus archivos.

En v1, varias exploraciones se ejecutaron como bloques de Python dentro de la sesión. Sus resultados y configuraciones están en JSON, pero no todos esos bloques tienen un archivo `.py` independiente. En v2 se conservaron scripts de las familias de pruebas. La narración integra los registros y el código visible en la conversación; **no se presenta como un diario instrumental completo con duración, consumo de tokens o tiempo por candidata**. Esos costes no se midieron aquí de forma exhaustiva.

No se guardaron los checkpoints de las 106 candidatas. Se conservaron sus especificaciones y listas de errores; para los estados original, v1 y v2 sí existen pesos y respuestas detalladas. El anexo de variantes identifica el archivo de origen y la posición de cada registro. Su numeración T001-T106 es editorial, añadida para este informe siguiendo el orden de ejecución de las familias y sus filas.

2.4 Relación con el experimento que motivó la revisión

La inspiración fue el experimento 8 de Sparks of Weight-Space Programming, publicado por investigadores de MIT CSAIL el 2 de noviembre de 2026 [R1]. Se tomó el principio de inspeccionar pesos y activaciones, formular hipótesis, escribir intervenciones directas y verificar sus efectos sin entrenamiento por gradientes.

No se reprodujo su protocolo exacto. En Qwen la intervención descrita corregía razonamientos repetitivos mediante una dirección interna. MiniSum terminaba sus respuestas y elegía dígitos erróneos. Aquí se permitieron ediciones en la cabeza de salida; la reparación final combina escalados de componentes y un cambio de coeficiente. Tampoco se impuso el presupuesto de consultas de aquel experimento ni se mantuvo un benchmark oculto durante la selección.

3. Arquitectura, representación y ejecución

3.1 Estructura exacta

El fichero `modelo_minisum_v2.py` define un Transformer causal con dos bloques pre-LayerNorm, cuatro cabezas de atención reales por bloque y una dimensión residual de 32. No confundir el sufijo histórico «v2» de la arquitectura con «edición v2» de esta reparación: la arquitectura es la misma en los tres checkpoints comparados.

Componente	Dimensiones o configuración	Parámetros
Embedding de tokens `wte`	15 × 32	480
Embedding de posiciones `wpe`	8 × 32	256
Atención por bloque	QKV: 96 × 32; proyección: 32 × 32; sin sesgos	4.096
MLP por bloque	32 → 128 → 32, GELU, con sesgos	8.352
Dos LayerNorm por bloque	32 ganancias y 32 sesgos cada una	128
Total de los dos bloques	2 × 12.576	25.152
LayerNorm final	32 ganancias + 32 sesgos	64
Cabeza de salida	15 × 32, sin sesgo	480
Total del modelo	**26.432 parámetros**	**26.432**

Las dos máscaras causales son buffers de 8 × 8, 128 valores adicionales que no se cuentan como parámetros y no se editaron. Dropout es 0. La dimensión de cada cabeza es 8. Los pesos evaluados son float32 en CPU; las operaciones de las ediciones conservan ese tipo.

Para una posición y un bloque, la estructura conceptual es:

```
x = embedding_token + embedding_posición
u = x + atención(LayerNorm1(x))
a = GELU(W_in · LayerNorm2(u) + b_in)
y = u + W_out · a + b_out
h = LayerNorm_final(salida_del_segundo_bloque)
logits = W_vocab · h
token_siguiente = argmax(logits)
```

La atención usa `softmax(QKᵀ / sqrt(8))` y máscara causal. En PyTorch, una capa lineal almacena su matriz como `[salidas, entradas]`: la salida en notación vectorial columna es `W · x + b`. Esta orientación es decisiva para entender por qué se editaron columnas en las proyecciones y filas en la cabeza de vocabulario.

3.2 Tokenizador y ausencia de ambigüedad en los índices

Token	ID	Token	ID	Token	ID
PAD	0	0	5	5	10
BOS	1	1	6	6	11
EOS	2	2	7	7	12
+	3	3	8	8	13
=	4	4	9	9	14

4+7= se convierte en `[BOS, 4, +, 7, =]` simbólicamente y en `[1, 9, 3, 12, 4]` como IDs. Para responder 11, la red debe generar `[6, 6, 2]`: los dos dígitos y EOS.

Todos los índices de tensores del informe empiezan en cero. `bloques.1` es el segundo bloque; la fila 9 de `cabeza.weight` es el dígito 4; la fila 13 es el dígito 8. La columna interna 8 de una matriz no representa necesariamente el número ocho. La neurona MLP 8 y la coordenada residual 8 son objetos distintos, aunque tengan el mismo índice.

3.3 Inferencia y criterio de corrección

El generador parte de cinco tokens y permite hasta tres tokens adicionales, completando el contexto máximo de ocho. En cada paso ejecuta el modelo y toma el `argmax`; no usa temperatura ni muestreo. Al emitir EOS termina la respuesta correspondiente.

El evaluador exige coincidencia de la lista completa de tokens con los dígitos esperados más EOS. Una cadena que parece correcta pero no termina adecuadamente no satisface ese criterio. El generador por lotes sigue calculando los pasos de otros elementos aunque uno ya haya terminado; deja de registrar tokens del elemento acabado. Por eso esos cálculos adicionales no alteran su respuesta.

La sintaxis admitida por la herramienta permite espacios y ausencia del = final, que normaliza. La afirmación exhaustiva del informe se refiere a los 100 prompts canónicos. El código del generador histórico produce el mismo texto en las verificaciones independientes; la comprobación estricta de EOS corresponde al evaluador nuevo.

4. Diagnóstico inicial y estados de referencia

4.1 De dos errores recordados a siete errores observados

Edgar recordaba dos fallos del TEST histórico, que contenía 20 ejemplos. La evaluación exhaustiva del adjunto identificó siete fallos entre las 100 entradas. Esto es compatible con el registro histórico: 60/60 de entrenamiento, 15/20 de validación y 18/20 de test suman 93/100. No se abrieron archivos sellados para ajustar esta reparación; las sumas canónicas se enumeraron en un evaluador nuevo.

Entrada	Correcto	Original	v1	v2
0+0	0	1	1	0
0+4	4	3	4	4
1+9	10	9	10	10
7+7	14	15	14	14
7+9	16	17	16	16
9+0	9	8	8	9
9+7	16	17	16	16

Los dos errores del test histórico eran 0+4 y 9+7. Los restantes cinco completan los errores de validación. Las 93 entradas correctas permanecen correctas en las dos versiones aceptadas. El anexo B conserva todas las respuestas, incluidas las que nunca fallaron.

4.2 El fallo era de elección de dígitos

Los siete fallos originales terminaban con EOS. En 7+7, 7+9 y 9+7, el primer dígito 1 era correcto y el error aparecía en el segundo. En 1+9 se elegía 9 y luego EOS, perdiendo el acarreo. No se observó la repetición larga que motivó la reparación de Qwen.

Las puntuaciones internas no son probabilidades. A modo de diagnóstico, en 0+0 el original asignaba 4,89299 al dígito 1 y 0,85664 al 0; en 9+0 asignaba 7,30701 al 8 y 4,42843 al 9. No bastaba con resolver un empate numérico diminuto. Era necesario modificar la competencia entre salidas conservando la generación de los demás casos.

4.3 Bases usadas durante la exploración

Base	Construcción	Aciertos	Errores pendientes
B0	Adjunto original	93	los siete iniciales
B1	B0 + cabeza 2 ×1,25 y cabeza 3 ×0,5 del segundo bloque	96	0+0, 0+4, 7+7, 9+0
B2	B1 + fila de salida del dígito 4 ×1,25	97	0+0, 0+4, 9+0
B3	B2 + columna MLP 20 ×1,5; edición v1	98	0+0, 9+0
B4	B3 + columna MLP 8 ×4	99	9+0
B5	B4 + coeficiente de salida [13,8] menos 1; edición v2	100	ninguno

Las modificaciones de cada fila se acumulan sólo en esta cadena aceptada. Las candidatas de una familia se reinician desde su base, como indica el anexo A. Es incorrecto interpretar las 106 filas como 106 actualizaciones consecutivas de un único modelo.

5. Primera intervención: del diagnóstico a 98/100

5.1 Paso I.1: ablaciones y duplicaciones de componentes completos

Se comenzó con 20 variantes sobre B0. Para cada una de las ocho cabezas se multiplicaron sus ocho columnas de proyección de salida por 0 o por 2. Esto suprime o duplica su contribución después de concatenar las cabezas; no se alteran directamente Q, K, V ni la máscara causal. Para cada uno de los dos MLP se multiplicaron tanto la matriz de salida como su sesgo por 0 o por 2.

En el primer bloque, las variantes de atención quedaron entre 45 y 70 aciertos. Suprimir la salida del primer MLP llevó a 0/100; duplicarla, a 63/100. Las intervenciones eran demasiado amplias para conservar el comportamiento aprendido. En el segundo bloque apareció información más útil: suprimir la cabeza 3 daba 95/100, con tres rescates pero una regresión, y duplicar la cabeza 2 mantenía 93/100 mediante cuatro rescates y cuatro regresiones.

Estas observaciones no justificaban conservar esas variantes, pero identificaban componentes cuyo cambio podía rescatar errores. La puntuación agregada por sí sola era insuficiente: una candidata con el mismo 93/100 podía haber sustituido cuatro aciertos por otros cuatro.

Fuente: `diagnostico_componentes.json`, T001-T020. La ablación de un MLP completo incluye su sesgo; las posteriores intervenciones de una neurona sólo escalan su columna y dejan ese sesgo intacto.

5.2 Paso I.2: moderar las dos cabezas prometedoras

Se probaron seis pares de factores para las cabezas 2 y 3 del segundo bloque. Las combinaciones `(1,5; 0,5)` y `(1,5; 0,25)` alcanzaban 95/100, pero perdían dos aciertos. Las combinaciones `(1,25; 0,5)` y `(1,25; 0,25)` alcanzaban 96/100 sin regresiones.

Se conservó `(1,25; 0,5)` como B1. El cambio refuerza la tercera cabeza en un 25 % y reduce a la mitad la cuarta. Se corrigieron 1+9, 7+9 y 9+7. En un espacio residual de 32 componentes, cada cabeza ocupa ocho columnas; por eso se modifican `[:,16:24]` y `[:,24:32]`, 256 valores cada una.

```
state['bloques.1.attn.proy.weight'][:, 16:24] *= 1.25
state['bloques.1.attn.proy.weight'][:, 24:32] *= 0.5
```

Fuente: `diagnostico_atencion_parcial.json`, T021-T026. La existencia de dos pares con igual cobertura no demuestra un óptimo global; se eligió una intervención útil y se siguió diagnosticando.

5.3 Paso I.3: activaciones MLP y lectura aproximada de contribuciones

Sobre B1 se capturaron activaciones para diez contextos: 0+0, 0+1, 1+0, 0+4, 4+0, los prefijos 7+7=1, 6+8=1, 8+6=1, y 9+0, 0+9. Se buscaban diferencias entre errores y contextos cercanos correctos, incluyendo la posición exacta del dígito que fallaba.

Para comparar un token correcto c y uno incorrecto e, se utilizó el indicador:

```
d = (W_vocab[c,:] - W_vocab[e,:]) ⊙ gamma_final
contribución_aproximada(j) = activación_MLP[j] · (dᵀ W_out[:,j])
```

⊙ significa producto componente a componente. El indicador orienta qué neuronas pueden escribir a favor de un token. No es una descomposición causal exacta de los logits: omite el centrado y la escala dependiente del estado de LayerNorm y, para el primer bloque, las transformaciones posteriores. Se usó como heurística para proponer intervenciones, que después se evaluaron con la red completa.

Se probaron 11 cambios de columnas MLP. Multiplicar por 2 o por 3 la neurona 20 del segundo bloque corregía algunos casos relacionados con el 4, pero dejaba 95/100 con regresiones. Varias otras intervenciones dejaban exactamente los mismos cuatro errores. Eliminar la columna 121 del segundo

bloque añadía un fallo y también se descartó.

Fuente: `activaciones_diagnostico.json` y `diagnostico_neuronas.json`, T027-T037. Aún no existía una edición de neurona aceptada en esta etapa.

5.4 Paso I.4: posición de operandos y ganancias de lectura

Se probaron dos formas de acercar los embeddings de las posiciones 1 y 3, que contienen los operandos: igualarlos a su media o recorrer la mitad del camino hacia ella. No equivalen a imponer conmutatividad, porque permanecen la atención causal y las demás transformaciones. Obtuvieron 92 y 93 aciertos, por lo que se descartaron.

Después se ensayaron ganancias de filas de la cabeza de salida. El token 4 con factor 1,5 corregía dos errores de B1 pero introducía tres; el token 0 con factor 6 dejaba 62/100; el token 9 con factor 1,75 dejaba 92/100. En cambio, multiplicar la fila del 4 por **1,25** producía 97/100 sin regresiones: corregía 7+7 y definía B2.

```
state['cabeza.weight'][9] *= 1.25
```

Como la cabeza de salida no tiene sesgo, esa operación multiplica el logit del token 4 por 1,25 para un estado fijo. No «añade un 25 % de probabilidad»: la probabilidad dependería del softmax de todos los logits. Tampoco afecta necesariamente sólo a respuestas cuyo resultado contiene un 4; de ahí la necesidad de evaluar todo el dominio.

Fuente: `diagnostico_posicion_lectura.json`, T038-T043. La variable `four_partial` identifica la candidata aceptada.

5.5 Paso I.5: direcciones contrastivas que no funcionaron

Sobre B2 se capturó la salida de LayerNorm final. Para cada error se construyó una dirección como diferencia entre el estado objetivo y la media de dos controles, y se normalizó a longitud uno:

```
u = normalize(h_objetivo - mean(h_controles))
W_vocab[token_correcto,:] += intensidad · u
```

Para cero se contrastó 0+0 frente a 0+1 y 1+0; para cuatro, 0+4 frente a 0+3 y 0+5; para nueve, 9+0 frente a 8+0 y 9+1. Se probaron intensidades 1 y 2. Ninguna mejoró B2 conservando sus aciertos.

La dirección del cero con intensidad 2 dio 0/100 bajo el criterio estricto: varias respuestas terminaban con ceros adicionales y no con EOS. La candidata no se puede considerar correcta porque empiece por el número esperado. Las direcciones del cuatro y del nueve podían rescatar el caso objetivo mientras dañaban muchas otras sumas.

Este resultado mostró que una diferencia de activaciones entre pocos ejemplos puede mezclar identidad del dígito, posición, longitud de respuesta y terminación. No todo contraste útil para describir un estado es seguro para editar globalmente una fila de salida.

Fuente: `contrastes_activaciones.json` y `diagnostico_contrastes.json`, T044-T049.

5.6 Paso I.6: volver a una columna MLP y obtener v1

Se evaluaron siete variantes adicionales sobre B2. Entre ellas, factores mayores para neuronas que favorecían el cero, la combinación de tres neuronas, cambios dirigidos al nueve y un refuerzo moderado de la neurona 20.

La columna 8 del segundo MLP multiplicada por 6 corregía 0+0, pero dañaba cinco aciertos y dejaba 93/100. La columna 27 por 6 dejaba 90/100. La columna 75 por 6 daba 98/100 sin perder aciertos de B2, pero 0+0 emitía una respuesta vacía: ese caso ya era incorrecto, de modo que no aparece como regresión de exactitud aunque cambia su modo de fallo.

La variante elegida fue **columna 20 por 1,5**, T056: corrigió 0+4 sin perjudicar los 97 aciertos de B2, y mantuvo respuestas completas. La relación con I.3 es esencial: aquella neurona no había funcionado al amplificarla más sobre otra base. En B2 la lectura del token 4 ya estaba reforzada, y bastó una intervención menor.

```
state['bloques.1.mlp.2.weight'][:, 20] *= 1.5
```

Se obtuvo B3, la v1 entregada: 98/100, con 0+0 → 1 y 9+0 → 8 pendientes. Las cuatro operaciones finales afectan a 576 valores en tres tensores. Fuente: `diagnostico_neuronas_ampliado.json`, T050-T056, y `editar_pesos.py`.

5.7 Paso I.7: intentos adicionales sobre los dos errores restantes

Se hicieron cinco pruebas sobre B3. Reducir la fila completa del dígito 8 a un factor 0,6 corregía 9+0, pero dañaba 1+7 y 7+1, que pasaban a 7: el balance era 97/100. Con factor 0,5 aparecían siete regresiones y se quedaba en 92/100. Con 0,75 no se corregía 9+0 y se perdía un acierto.

Escalar el embedding del token 0 por 0,5 corregía 9+0 pero dañaba cinco sumas, dejando 94/100; escalarlo por 1,5 dejaba 95/100. Ninguna de estas candidatas reemplazó a B3. Se conservó una reparación parcial sin regresiones y se documentaron los dos fallos aún abiertos.

Fuente: `diagnostico_ceros_ochos.json`, T057-T061. Las 61 variantes son pruebas de edición, no épocas ni pasos de entrenamiento; no incluyen todos los cálculos auxiliares de activaciones y verificaciones.

6. Segunda intervención: señales más selectivas hasta 100/100

6.1 Paso II.1: inspección de todos los prefijos y corrección del cero

La segunda ronda partió de B3. Se construyeron **245 contextos de diagnóstico**: los 100 prompts antes de emitir la primera cifra, los prefijos siguientes de las 45 respuestas de dos cifras y los 100 contextos inmediatamente anteriores a EOS. Las 55 sumas de una cifra necesitan dos decisiones de salida y las 45 de dos cifras necesitan tres: `55 × 2 + 45 × 3 = 245`.

En estos contextos se introdujeron prefijos correctos para observar posiciones concretas, aunque el modelo original pudiera no alcanzarlas por sí mismo. Es una inspección de tipo teacher forcing, sin calcular pérdidas ni ajustar pesos. Se distingue de la evaluación de aceptación, que genera cada respuesta autoregresivamente desde el prompt y debe superar por sí sola cada decisión.

Se capturaron las entradas y preactivaciones de los dos MLP. Un indicador adicional comparó la activación de cada neurona en el objetivo con su mayor activación en contextos cuyo siguiente token correcto era diferente. El denominador se acotó por 0,01; se combinó con el signo de la proyección de escritura. Es una clasificación heurística, sensible a activaciones negativas de GELU y al conjunto de controles, no una medida universal de selectividad ni una prueba causal.

Se evaluaron 14 candidatas sobre B3. La neurona 27 por 4 corregía el cero, pero introducía 9+3 → 11. La neurona 8 por **4** corrigió 0+0 y conservó todos los demás aciertos: T064, B4, 99/100. El resto de la familia no mejoró esa cobertura.

```
state['bloques.1.mlp.2.weight'][:, 8] *= 4
```

Esta neurona también se activa en prefijos como 2+8=1, donde el siguiente dígito correcto es cero. No es un detector exclusivo de 0+0. El resultado muestra que una intensidad intermedia es suficiente y compatible con los otros casos; el fracaso con factor 6 en la primera ronda no demostraba que la neurona fuese inútil. Además, las bases B2 y B3 no son idénticas, por lo que no se debe atribuir toda la diferencia entre rondas sólo al factor.

Fuente: `inspeccionar.py`, `activaciones_prefijos.json`, `probar_selectivos.py` y `pruebas_selectivas.json`, T062-T075.

6.2 Paso II.2: asimetría de operandos y geometría del embedding del nueve

El caso restante era 9+0. Se compararon mapas de atención de 9+0, 0+9, 8+0, 9+1 y 1+9. Los mapas diferían al intercambiar operandos. Por ejemplo, en la cabeza 2 del segundo bloque, el estado de = en 9+0 atendía casi por completo a la posición del segundo operando, mientras que en 0+9 atendía casi por completo a la posición de =. Es una observación descriptiva, no una prueba de que ese mapa aislado fuese la causa del error.

Sobre B4 se probaron siete cambios del embedding del token 9: extrapolar desde 8 hacia 9 con factores 0,25, 0,5 y 1; extrapolar desde 7 con 0,25 y 0,5; y escalar directamente el vector del 9 por 0,75 o 1,25. No se conservó ninguna variante. El escalado por 1,25 dejaba 99/100 con el mismo fallo; otras variantes introducían errores adicionales.

La extrapolación se define como `e9' = e9 + alpha · (e9 - e8)` o su versión con e7. Que los símbolos sean números consecutivos no garantiza que sus embeddings formen una recta aritmética. Las pruebas mostraron precisamente que esa hipótesis no bastaba para reparar el modelo.

Fuente: `diagnostico_nueve.py`, `atencion_orden_operandos.json` y `pruebas_embedding_nueve.json`, T076-T082.

6.3 Paso II.3: eliminación de direcciones asociadas al orden

Se registraron, para 9+0, 0+9 y 8+0, la salida de atención del segundo bloque, la de su MLP, el residual final del bloque y el estado después de LayerNorm final. Para atención, MLP y residual se definió:

```
u = normalize(estado(9+0) - estado(0+9))
W[:, C] = W[:, C] - u · (uᵀ W[:, C])
```

La transformación elimina la proyección sobre u de las columnas seleccionadas `C`. Para cada una de las tres direcciones se probaron cinco destinos: las cuatro cabezas del segundo bloque, por separado, y las 16 columnas del MLP con mayor `abs(uᵀ W[:,j])`. Son 15 variantes, todas reiniciadas desde B4. Ninguna corrigió 9+0 manteniendo los aciertos.

Se añadieron tres variantes de lectura: la dirección normalizada `h(9+0) - h(8+0)` se sumó a la fila del token 9 con intensidades 1, 2 y 4. Las dos mayores corregían el objetivo, pero producían 19 y 39 regresiones respectivamente; los resultados eran 81/100 y 61/100.

La técnica de proyectar direcciones fue una prueba explícita de la familia que motivó la conversación. Se descartó por sus efectos medidos. No se incluyó una proyección de este tipo en la reparación final.

Fuente: `probar_direcciones.py`, `contrastes_orden.json` y `pruebas_direcciones.json`, T083-T100.

6.4 Paso II.4: inspección por coordenadas de la lectura final

Se pasó de intervenir filas completas a examinar sus coeficientes individuales. Se capturó el vector final h, de dimensión 32, para 9+0 y controles relevantes: 8+0, 1+7, 7+1, 0+8, 1+9, 0+9 y 0+0. Los casos 1+7 y 7+1 eran controles especialmente informativos porque habían empeorado al reducir toda la fila del 8.

Contexto en B4	Coordenada h[8]	Coordenada h[30]
9+0=	2,467	1,140
8+0=	1,679	0,856
1+7=	0,776	0,110
7+1=	0,672	0,318
0+8=	1,543	0,484

Estas diferencias motivaron seis intervenciones específicas en la fila del token 8. Todas dieron 100/100: restar 2,5 o 3 a la coordenada 30; restar 2,5 a la 28; restar 2 a la 9; restar 1,25 o 1 a la 8. No se había encontrado una única localización imprescindible; existían varias correcciones funcionales.

Se compararon también los márgenes de tres candidatas. Restar 1,25 en la coordenada 8 dejaba un margen mínimo de sólo 0,022577 en 3+5; restar 1 dejaba un margen mínimo de 0,151607. La resta de 2,5 en la coordenada 30 tenía el mismo mínimo global que la resta de 1 en la coordenada 8, pero requería una modificación absoluta mayor. Se eligió T106: **restar 1 en** `[13,8]`.

```
state['cabeza.weight'][13, 8] -= 1
```

Fuente: `lectura_dimensiones.json`, `probar_lectura.py` y `pruebas_lectura_selectiva.json`, T101-T106. Las comparaciones de márgenes constan en la ejecución de la sesión; los márgenes finales se conservaron en `margenes.json`.

6.5 Por qué esa única modificación cambia la respuesta

La cabeza de salida es lineal y no tiene sesgo. Si z8 es la puntuación del token 8 y h8 la coordenada interna 8, la intervención satisface, en aritmética real y para el mismo prefijo:

```
z8 = W_vocab[13,:] · h
W_vocab'[13,8] = W_vocab[13,8] - 1
z8' = z8 - h8
z_t' = z_t   para todo token t distinto de 8
```

El peso float32 pasa de `0.18949353694915771` a `-0.8105064630508423`. En 9+0, h8 es positivo y suficientemente grande para que el 8 deje de ganar al 9. Si h8 fuera negativo en otro contexto, la misma edición aumentaría el logit del 8; por eso no se interpreta como una penalización uniforme a ese dígito.

La igualdad anterior describe un paso con prefijo fijo. Una vez cambia el token elegido, el siguiente estado puede cambiar porque la generación es autoregresiva. En la reparación completa se verificaron todas las secuencias, no sólo el primer logit. La proximidad entre el índice de neurona 8 y la coordenada 8 no establece por sí misma una conexión exclusiva entre ambos.

7. Verificación de la reparación y evidencia causal

7.1 Guardar, recargar y comprobar rutas independientes

La candidata aceptada se guardó como v2, se recargó mediante `torch.load(..., weights_only=True, map_location='cpu')` y se evaluó desde el prompt. Se obtuvo 100/100 con coincidencia estricta de dígitos y EOS. Se verificó que las formas y tipos de los tensores fueran iguales y que sus valores fueran finitos.

El generador histórico confirmó las 100 respuestas de v2. En v1 había confirmado 200 respuestas, contando el original y la edición. El navegador confirmó 200 secuencias en cada ronda, original frente a la versión de esa ronda. No son cientos de problemas independientes adicionales: son comprobaciones de implementación sobre el mismo dominio.

La versión JavaScript usa los pesos float32 exactos exportados y reconstruye las operaciones de edición con redondeo float32. Ejecuta embeddings, atención, MLP, LayerNorm y salida; usa una aproximación de `erf` para GELU. La diferencia máxima de logits frente a PyTorch fue 0,0000159515 en v1 y 0,0000152738 en v2. Ninguna cambió los tokens evaluados.

7.2 Restaurar cada cambio para comprobar su efecto

Control sobre v2	Aciertos	Error observado
Restaurar sólo la columna MLP 8 a v1	99	0+0 → 1
Restaurar sólo la cabeza de salida a v1	99	9+0 → 8
Restaurar ambos cambios a v1	98	reaparecen ambos errores
Pesos aleatorios, semilla 42, misma ruta	0	control negativo

Los dos primeros controles demuestran la necesidad de cada edición adicional dentro de esta configuración concreta. No prueban que esas localizaciones sean la única forma de reparar los casos; la familia final encontró varias alternativas. La restauración doble recupera el comportamiento de v1 y el control aleatorio comprueba que la ruta no devuelve sumas correctas independientemente de los pesos.

7.3 Puntuaciones observadas de los dos casos finales

Se define aquí margen dirigido como `logit(correcto) - logit(error original)`. No siempre coincide con el margen frente al segundo candidato de todo el vocabulario.

Entrada	Estado	Logit correcto	Logit del error original	Margen dirigido
0+0	Original	0,856635	4,892992 (1)	-4,036357
0+0	v1	2,324410	5,255336 (1)	-2,930927
0+0	v2	4,293737	4,126544 (1)	+0,167193
9+0	Original	4,428428	7,307014 (8)	-2,878586
9+0	v1	5,142598	7,535220 (8)	-2,392622
9+0	v2	5,488431	4,895944 (8)	+0,592488

El menor margen entre ganador y segundo candidato fue 0,15160656, en 0+4. Es positivo en todos los tokens finales: no hubo empates. Esto no garantiza estabilidad ante cuantización, otros tipos de dato o cambios de backend.

8. Parche completo, tamaño e integridad

8.1 Las seis operaciones desde el original

```
with torch.inference_mode():
    # Primera intervención, B0 -> B3.
    s['bloques.1.attn.proy.weight'][:, 16:24] *= 1.25
    s['bloques.1.attn.proy.weight'][:, 24:32] *= 0.5
    s['cabeza.weight'][9] *= 1.25
    s['bloques.1.mlp.2.weight'][:, 20] *= 1.5
    # Segunda intervención, B3 -> B5.
    s['bloques.1.mlp.2.weight'][:, 8] *= 4
    s['cabeza.weight'][13, 8] -= 1
```

Este código sólo es válido para el checkpoint y arquitectura identificados. No se debe ejecutar dos veces sobre los mismos pesos editados: volvería a multiplicar o restar. El script de referencia adjunto comprueba el hash de entrada, reconstruye desde el original y compara sus tensores con la v2 oficial antes de permitir una nueva salida.

8.2 Diferencia por tensor

Tensor	v1	Cambio nuevo v2	Acumulado final
bloques.1.attn.proy.weight	512	0	512
bloques.1.mlp.2.weight	32	32	64
cabeza.weight	32	1	33
Total de valores diferentes	**576**	**33**	**609**

Las regiones son disjuntas: no se cuentan varias veces los mismos índices. Permanecen 25.823 parámetros idénticos al original. Embeddings, LayerNorm, sesgos, QKV, buffers causales y todo el primer bloque conservan sus valores. Los 609 índices y sus valores exactos antes/después figuran en los anexos CSV y JSON.

8.3 Hashes de los checkpoints entregados

Original:

`ef5b831ae582a3db3f3b74908ac2ab3c5d0eb0e0dd909e576c8153e0fbbeb159`

Edición v1:

`690ff127aa8ee2802c53edcc90abf0dcc443dcda75e6d7002ae5ec3d45b04764`

Edición v2:

`2677af4f2f1f6777c0815db2ebd2bdc3ea4651ebeefb9e3cae3f6443eb5048df`

Los hashes verifican los ficheros concretos. Volver a serializar tensores iguales puede producir otro hash de archivo por detalles del contenedor, versión de PyTorch o nombre de salida. Para reconstrucciones se comprueban también claves, formas, tipos y equivalencia exacta de cada tensor; no debe confundirse igualdad semántica de pesos con identidad de bytes del archivo.

9. Coste, entorno y operaciones de entrenamiento

Las intervenciones usaron CPU, un hilo, Python 3.14.4 y PyTorch 2.14.1+cpu. La ausencia de NumPy generó un aviso de PyTorch, pero los cálculos usados no lo requerían. El número de variantes registradas es 61 + 45 = 106. No equivale al número total de pases forward, porque cada evaluación genera varios tokens y hubo inspecciones, controles y comprobaciones adicionales.

No se midieron de forma completa las operaciones de inferencia, la energía, el tiempo de cada candidata ni el consumo del agente editor. Por tanto, el informe no afirma un ahorro cuantificado frente a entrenamiento ni que la reparación haya sido computacionalmente gratuita.

El nuevo entrenamiento realizado en v1, v2 y el conjunto de este experimento de edición es **cero pasos, 0 × 10⁰ operaciones, 0 billones**. Esa cifra no incluye el entrenamiento histórico que produjo el modelo ni el coste de las inferencias de diagnóstico.

Para el checkpoint original, los registros del piloto indican 60 secuencias por paso, longitud completa 8, siete símbolos de entrada procesados por secuencia, un paso por época, checkpoint elegido en época 210 y corrida terminada en 260. Aplicando la estimación usada por Edgar:

```
operaciones ≈ 6 × parámetros × pasos × lote × (largo - 1)
```

Alcance	Símbolos procesados	Estimación científica	Billones (10^12)
Hasta época 210	88.200	1,39878144 × 10^10	0,0139878144
Piloto completo, 260 épocas	109.200	1,73182464 × 10^10	0,0173182464
Nuevas operaciones de entrenamiento v1 + v2	0	0 × 10^0	0

Es una aproximación, incluye el padding procesado y no sustituye a una medición de FLOPs reales. El coste histórico de todas las demás corridas de MiniSumAI no se reconstruyó. No se identifica el coste del piloto con el de todo el proyecto histórico.

10. Qué demuestran los resultados y qué queda abierto

Demostrado: una edición directa de 609 parámetros convierte este checkpoint de 93/100 a 100/100 en su dominio canónico, sin cambiar la arquitectura ni la ruta de inferencia y sin entrenamiento por gradientes. El artefacto funciona después de guardarlo y recargarlo, y los dos cambios finales tienen controles de reversión individual.

Inferencia razonable pero limitada: algunas representaciones existentes permitían una lectura o contribución más favorable a los dígitos correctos. La reparación aprovechó esas señales. Esto no permite concluir que se haya identificado un «circuito de suma» completo o que cada neurona intervenida tenga una función única.

No demostrado: edición mínima, unicidad de la solución, causalidad exhaustiva de todas las decisiones de v1, robustez a ruido en pesos, funcionamiento con otros modelos/semillas, mejoras fuera del dominio, ni superioridad de coste frente a otras técnicas. V1 no recibió un diseño factorial completo de reversión de sus cuatro operaciones; v2 sí recibió controles individuales de sus dos adiciones.

Riesgo metodológico: consultar repetidamente todas las sumas convierte el dominio en material de desarrollo. Aunque no se entrene con gradientes, las decisiones del agente contienen información de las respuestas correctas. Sería incorrecto presentar el 100/100 como un resultado de generalización sobre el antiguo TEST reservado.

Lección sobre la explicación: los fallos no desaparecieron por aumentar genéricamente la «inteligencia matemática» del modelo. Se modificaron decisiones concretas y se midió su efecto en cada entrada. La evidencia permite afirmar reparación funcional; una explicación mecanística global exigiría experimentos adicionales.

11. Archivos, fuentes y trazabilidad

Las carpetas históricas se abrevian así:

• **V1:** `../minisum-edicion-20261104/`.

• **V2:** `../minisum-edicion-20261104-v2/`.

• **IF:** carpeta de este informe, `minisum-informe-final/`.

Evidencia	Ubicación relativa	Uso
E01	V1/modelo_minisum_v2.py; minisum_tokenizer.py	Arquitectura e IDs
E02	V1/inferencia.py; diagnostico.py	Generación y evaluación
E03	V1/respuestas_original.json	Baseline, tokens y logits
E04	V1/diagnostico_*.json	61 variantes de la primera ronda
E05	V1/activaciones_diagnostico.json; contrastes_activaciones.json	Señales inspeccionadas
E06	V1/editar_pesos.py; verificacion.json	Parche y comprobaciones v1
E07	V2/inspeccionar.py; activaciones_prefijos.json	245 prefijos de diagnóstico
E08	V2/pruebas_.json; probar_.py; diagnostico_nueve.py	45 variantes de la segunda ronda
E09	V2/atencion_orden_operandos.json; contrastes_orden.json; lectura_dimensiones.json	Atención, estados y lectura
E10	V2/editar_pesos_v2.py; verificar_v2.py; verificacion.json	Parche final y controles
E11	V2/respuestas_editado.json; margenes.json	Tokens, logits y márgenes finales
E12	V1 y V2/verificacion-browser.json; inferencia-browser.js	Concordancia del navegador
E13	V1/pesos_modificados.json; V2/pesos_nuevos.json; pesos_modificados_total.json	Índices exactos modificados
E14	IF/anexos/fuentes-verificadas.json; auditoria.json	Auditoría documental nueva

R1. Rikiya Takehi, Jayden Teoh, Alex Zhang y Omar Khattab, MIT CSAIL. Sparks of Weight-Space Programming: LLMs can natively read & write neural network weights. Publicación de investigación en blog, 2 de noviembre de 2026. https://rikiyat.github.io/blog/weight-space-programming/

R2. Registros del experimento de referencia, experimento 8: https://github.com/RikiyaT/lm_understand-logs/tree/main/exp8_math. Se cita como antecedente metodológico; no es evidencia de los resultados de MiniSum.

Las fuentes principales de este informe son los archivos locales y la secuencia de trabajo de esta conversación. Para consultar qué se probó exactamente, el anexo A prevalece sobre una descripción resumida: incluye base, especificación de cada candidata, aciertos y cambios de corrección.

12. Reproducción sin alterar el laboratorio histórico

El paquete integrado conserva las carpetas V1, V2 e IF como hermanas. Así se mantienen las referencias relativas. Se necesita Python con PyTorch compatible para reconstruir pesos, y Node.js para comprobar el generador del navegador. Las intervenciones originales se ejecutaron con las versiones indicadas en el apartado 9.

El script nuevo `reconstruir_y_comprobar.py`, añadido para este informe, aplica las seis operaciones a una carga nueva del original y compara todos los tensores con los de la v2 entregada. Por defecto no escribe pesos. Es una herramienta de reproducción del parche conocido, no una repetición de la búsqueda exploratoria.

```
cd minisum-informe-final
python reconstruir_y_comprobar.py
```

Para guardar una copia reconstruida, usar un nombre que no exista:

```
python reconstruir_y_comprobar.py --salida ./reconstruido.pt
```

Para preguntar a la red, usar la inferencia portable de V2:

```
cd ../minisum-edicion-20261104-v2
python inferencia.py \
  MiniSum-Research-seed42-epoca210-edicion-directa-v2.pt \
  '0+0=' '9+0=' '9+9='
```

Para verificar el traslado al navegador:

```
node verificar-browser.cjs
```

Los scripts históricos `verificar.py` y `verificar_v2.py` escriben resultados en sus carpetas y usan una ruta absoluta al generador histórico para una comprobación adicional. Si se ejecutan de nuevo, hacerlo en una copia de trabajo y conservar los archivos originales. No son necesarios para usar el `.pt` ni para comprobar la igualdad del parche completo.

El comparador final está en `V2/minisum-comparador.html`; el documento no sustituye ni despliega los modelos de ningún servicio. Para trasladar la reparación a una aplicación existente hay que integrar explícitamente el nuevo checkpoint y repetir la verificación de su ruta de inferencia.

13. Procedimiento de referencia para futuras intervenciones

13.1 Definir qué significa reparar

Registrar el modelo exacto, hash, arquitectura, tokenizador, dominio admitido, forma de generación y criterio de éxito. Distinguir resultado completo de acierto parcial, y acierto de formato de acierto semántico. Definir si la prioridad es cero regresiones, una tolerancia acotada o un balance explícito entre tareas. No adoptar automáticamente el objetivo 100/100 cuando el nuevo dominio no es finito ni enumerable.

13.2 Congelar las entradas y separar responsabilidades

Mantener una copia inmutable del original y crear una carpeta por ronda. Separar inferencia, evaluación y edición. El generador no debe leer respuestas correctas. El evaluador sí puede crear o consultar etiquetas. El editor puede inspeccionar errores, pero debe reconocer que esa información convierte los casos consultados en datos de desarrollo.

En un problema más amplio, reservar un conjunto realmente oculto para la validación final y otro conjunto de regresión. No reutilizar como «reservados» los ejemplos que ya se emplearon para decidir cambios. Si se trabaja sobre todo un dominio finito, declarar cobertura exhaustiva en lugar de generalización.

13.3 Reproducir el baseline antes de editar

Comprobar que el archivo correcto produce el comportamiento reportado. Registrar respuestas crudas, tokens, terminación, márgenes y errores por posición. Distinguir fallos de cálculo, elección de token, formato, parada o bucle. Una hipótesis útil para un tipo de fallo puede ser inadecuada para otro.

Ejecutar un control negativo con pesos aleatorios o alterados de forma conocida. Si el sistema conserva artificialmente todas las respuestas correctas, revisar primero la ruta de inferencia: podría estar usando una tabla, una caché o un cálculo externo.

13.4 Localizar componentes con pruebas causales sencillas

Usar ablaciones o factores discretos sobre bloques, cabezas o columnas, siempre reiniciando desde una base conocida. Evaluar los rescates y las regresiones por ejemplo, no sólo la nota agregada. Un componente prometedor puede mostrar rescates aun cuando una ablación completa sea demasiado destructiva.

Acotar de antemano el número o coste de pruebas para que la exploración no se convierta en una búsqueda extensa sin trazabilidad. En MiniSum se registró el número de variantes después de ejecutarlas; una mejora para futuros experimentos sería fijar previamente el presupuesto e instrumentar cada llamada.

13.5 Formular contrastes y reducir el alcance del cambio

Comparar casos fallidos con controles que difieran en una propiedad relevante. Registrar la capa, posición y estado exactos que se comparan. Usar activaciones y proyecciones como generadores de hipótesis, sin asumir que cada dirección es una función aislada. Si una edición amplia corrige el objetivo pero daña controles, buscar una intensidad menor, una columna o un coeficiente específico.

No trasladar índices entre modelos: una neurona 8 de otro checkpoint no tiene por qué cumplir la misma función. Comprobar las dimensiones y la orientación de las matrices. Verificar si el cambio propuesto se aplica antes o después de una normalización y cómo afecta a los pasos posteriores de generación.

13.6 Aceptar cambios por comportamiento completo

Para cada candidata guardar: ID, hash de la base, hipótesis, tensor, índices, operación, intensidad, código, coste observado y resultados. Medir el caso objetivo, todos los controles cercanos y el conjunto de regresión. Comprobar la secuencia completa, incluido EOS, en generación libre.

Cuando se acepte un cambio, congelar una nueva base y explicar qué corrige y qué queda pendiente. No reutilizar silenciosamente estados mutados. Si dos candidatas tienen la misma cobertura, comparar magnitud, número de valores alterados, margen y simplicidad sin afirmar un óptimo no demostrado.

13.7 Confirmar, empaquetar y declarar límites

Guardar y recargar el checkpoint, evaluar con una implementación independiente cuando sea viable, restaurar las ediciones individualmente y comprobar que los efectos atribuidos reaparecen. Verificar claves, dimensiones, tipos, valores finitos y diferencias exactas de pesos. Documentar si el cambio requiere otro decodificador o funciona con el mismo de antes.

Conservar un script de parche determinista que no dependa del dataset, un manifiesto de hashes, la tabla de regresiones, el conjunto completo de respuestas, los experimentos rechazados y una forma de prueba manual. Dejar separados el artefacto reparado y la afirmación científica: que una edición funcione no implica que su mecanismo esté completamente entendido.

13.8 Plantilla mínima para registrar una candidata

```
{
  "id": "T001",
  "base_sha256": "hash real del checkpoint de partida",
  "hipotesis": "componente y comportamiento que se quiere comprobar",
  "tensor": "nombre exacto",
  "indices": "selección explícita",
  "operacion": "transformación y constantes",
  "codigo_sha256": "hash del código ejecutado",
  "diagnostico": "contextos y activaciones consultados",
  "generacion": "greedy, límite de tokens, formato",
  "aciertos": 0,
  "rescates": [],
  "regresiones": [],
  "decision": "aceptar o descartar con motivo",
  "duracion_y_recursos": "medidos o declarados no medidos"
}
```

Anexo A. Catálogo de las 106 variantes

La tabla siguiente se genera directamente desde los once JSON de resultados, en el orden documentado. **R** son errores de la base que la candidata corrige; **G** son aciertos de la base que la candidata estropea. «Base» remite al apartado 4.3. Todos los aciertos tienen denominador 100. Una modificación del texto erróneo que sigue siendo erróneo no cuenta como regresión ni como rescate.

El CSV `anexos/106-variantes.csv` y el JSON equivalente incluyen además la lista íntegra de respuestas erróneas de cada variante, su fichero fuente y el número de fila. Los factores y direcciones se interpretan según la familia descrita en los apartados 5 y 6. El catálogo no cuenta las mediciones auxiliares ni las verificaciones de aceptación como nuevas candidatas.

ID	Familia base	Operación	Aciertos	R	G
T001	I.1/B0	L0 H0 ×0.0	50	0	43
T002	I.1/B0	L0 H0 ×2.0	54	0	39
T003	I.1/B0	L0 H1 ×0.0	45	2	50
T004	I.1/B0	L0 H1 ×2.0	56	2	39
T005	I.1/B0	L0 H2 ×0.0	47	0	46
T006	I.1/B0	L0 H2 ×2.0	53	2	42
T007	I.1/B0	L0 H3 ×0.0	56	3	40
T008	I.1/B0	L0 H3 ×2.0	70	1	24
T009	I.1/B0	L0 MLP salida y sesgo ×0.0	0	0	93
T010	I.1/B0	L0 MLP salida y sesgo ×2.0	63	2	32
T011	I.1/B0	L1 H0 ×0.0	92	1	2
T012	I.1/B0	L1 H0 ×2.0	92	1	2
T013	I.1/B0	L1 H1 ×0.0	43	1	51
T014	I.1/B0	L1 H1 ×2.0	50	1	44
T015	I.1/B0	L1 H2 ×0.0	84	0	9
T016	I.1/B0	L1 H2 ×2.0	93	4	4
T017	I.1/B0	L1 H3 ×0.0	95	3	1
T018	I.1/B0	L1 H3 ×2.0	81	0	12
T019	I.1/B0	L1 MLP salida y sesgo ×0.0	52	0	41
T020	I.1/B0	L1 MLP salida y sesgo ×2.0	67	1	27
T021	I.2/B0	L1 H2 ×1; H3 ×0.5	94	2	1
T022	I.2/B0	L1 H2 ×1.5; H3 ×1	94	2	1
T023	I.2/B0	L1 H2 ×1.5; H3 ×0.5	95	4	2
T024	I.2/B0	L1 H2 ×1.5; H3 ×0.25	95	4	2
T025	I.2/B0	L1 H2 ×1.25; H3 ×0.5	96	3	0

ID	Familia base	Operación	Aciertos	R	G
T026	I.2/B0	L1 H2 ×1.25; H3 ×0.25	96	3	0
T027	I.3/B1	L1 MLP j20 ×2.0	95	1	2
T028	I.3/B1	L1 MLP j20 ×3.0	95	2	3
T029	I.3/B1	L1 MLP j27 ×2.0	96	0	0
T030	I.3/B1	L1 MLP j8 ×2.0	96	0	0
T031	I.3/B1	L1 MLP j75 ×2.0	96	0	0
T032	I.3/B1	L1 MLP j6 ×0.0	96	0	0
T033	I.3/B1	L1 MLP j56 ×0.0	96	0	0
T034	I.3/B1	L0 MLP j79 ×0.5	96	0	0
T035	I.3/B1	L0 MLP j47 ×0.5	96	0	0
T036	I.3/B1	L1 MLP j46 ×2.0	96	0	0
T037	I.3/B1	L1 MLP j121 ×0.0	95	0	1
T038	I.4/B1	Posiciones 1 y 3 iguales a su media	92	0	4
T039	I.4/B1	Posiciones 1 y 3 a mitad hacia su media	93	0	3
T040	I.4/B1	Fila token 4 ×1,5	95	2	3
T041	I.4/B1	Fila token 0 ×6	62	1	35
T042	I.4/B1	Fila token 9 ×1,75	92	1	5
T043	I.4/B1	Fila token 4 ×1,25	97	1	0
T044	I.5/B2	Contraste zero; intensidad 1.0	95	0	2
T045	I.5/B2	Contraste zero; intensidad 2.0	0	0	97
T046	I.5/B2	Contraste four; intensidad 1.0	86	1	12
T047	I.5/B2	Contraste four; intensidad 2.0	85	1	13
T048	I.5/B2	Contraste nine; intensidad 1.0	92	1	6
T049	I.5/B2	Contraste nine; intensidad 2.0	87	1	11
T050	I.6/B2	L1 MLP j27 ×6	90	2	9
T051	I.6/B2	L1 MLP j8 ×6	93	1	5
T052	I.6/B2	L1 MLP j75 ×6	98	1	0
T053	I.6/B2	L1 MLP j27 ×3; L1 MLP j8 ×3; L1 MLP j75 ×3	93	2	6
T054	I.6/B2	L1 MLP j46 ×4	93	0	4
T055	I.6/B2	L0 MLP j79 ×0	95	0	2
T056	I.6/B2	L1 MLP j20 ×1.5	98	1	0
T057	I.7/B3	Fila token 8 ×0.5	92	1	7

ID	Familia base	Operación	Aciertos	R	G
T058	I.7/B3	Fila token 8 ×0.6	97	1	2
T059	I.7/B3	Fila token 8 ×0.75	97	0	1
T060	I.7/B3	Embedding token 0 ×0.5	94	1	5
T061	I.7/B3	Embedding token 0 ×1.5	95	0	3
T062	II.1/B3	L1 MLP j27 ×3	98	0	0
T063	II.1/B3	L1 MLP j27 ×4	98	1	1
T064	II.1/B3	L1 MLP j8 ×4	99	1	0
T065	II.1/B3	L1 MLP j98 ×4	97	0	1
T066	II.1/B3	L1 MLP j98 ×6	97	0	1
T067	II.1/B3	L1 MLP j35 ×6	91	0	7
T068	II.1/B3	L1 MLP j112 ×12	97	0	1
T069	II.1/B3	L1 MLP j92 ×6	98	0	0
T070	II.1/B3	L1 MLP j46 ×3	95	0	3
T071	II.1/B3	L1 MLP j23 ×6	98	0	0
T072	II.1/B3	L1 MLP j44 ×6	96	0	2
T073	II.1/B3	L1 MLP j4 ×6	96	0	2
T074	II.1/B3	L1 MLP j1 ×12	97	0	1
T075	II.1/B3	L0 MLP j120 ×6	97	0	1
T076	II.2/B4	e9 + alpha(e9 - e8); alpha=0.25	97	0	2
T077	II.2/B4	e9 + alpha(e9 - e8); alpha=0.5	92	0	7
T078	II.2/B4	e9 + alpha(e9 - e8); alpha=1.0	88	0	11
T079	II.2/B4	Embedding token 9; alpha=0.75	97	0	2
T080	II.2/B4	Embedding token 9; alpha=1.25	99	0	0
T081	II.2/B4	e9 + alpha(e9 - e7); alpha=0.25	98	0	1
T082	II.2/B4	e9 + alpha(e9 - e7); alpha=0.5	95	0	4
T083	II.3/B4	u=attention; destino head0	99	0	0
T084	II.3/B4	u=attention; destino head1	95	0	4
T085	II.3/B4	u=attention; destino head2	90	0	9
T086	II.3/B4	u=attention; destino head3	96	0	3
T087	II.3/B4	u=attention; destino mlp16	97	0	2
T088	II.3/B4	u=mlp; destino head0	96	0	3
T089	II.3/B4	u=mlp; destino head1	76	0	23

ID	Familia base	Operación	Aciertos	R	G
T090	II.3/B4	u=mlp; destino head2	93	0	6
T091	II.3/B4	u=mlp; destino head3	96	0	3
T092	II.3/B4	u=mlp; destino mlp16	95	0	4
T093	II.3/B4	u=residual; destino head0	98	0	1
T094	II.3/B4	u=residual; destino head1	77	0	22
T095	II.3/B4	u=residual; destino head2	80	0	19
T096	II.3/B4	u=residual; destino head3	94	0	5
T097	II.3/B4	u=residual; destino mlp16	94	0	5
T098	II.3/B4	u=nine_vs_eight; intensidad 1.0	91	0	8
T099	II.3/B4	u=nine_vs_eight; intensidad 2.0	81	1	19
T100	II.3/B4	u=nine_vs_eight; intensidad 4.0	61	1	39
T101	II.4/B4	Fila token 8, coordenada 30: delta -2.5	100	1	0
T102	II.4/B4	Fila token 8, coordenada 30: delta -3.0	100	1	0
T103	II.4/B4	Fila token 8, coordenada 28: delta -2.5	100	1	0
T104	II.4/B4	Fila token 8, coordenada 9: delta -2.0	100	1	0
T105	II.4/B4	Fila token 8, coordenada 8: delta -1.25	100	1	0
T106	II.4/B4	Fila token 8, coordenada 8: delta -1.0	100	1	0

Anexo B. Respuestas de las 100 sumas

La tabla se genera desde `anexos/100-respuestas.csv`. Incluye los pares ordenados: 0+9 y 9+0 son entradas distintas. «O» es original, «V1» primera reparación y «V2» reparación completa. Las etiquetas correctas sólo califican las respuestas; no intervienen en su generación.

Entrada	Correcto	O	V1	V2
0+0=	0	1	1	0
0+1=	1	1	1	1
0+2=	2	2	2	2
0+3=	3	3	3	3
0+4=	4	3	4	4
0+5=	5	5	5	5
0+6=	6	6	6	6
0+7=	7	7	7	7
0+8=	8	8	8	8
0+9=	9	9	9	9
1+0=	1	1	1	1
1+1=	2	2	2	2
1+2=	3	3	3	3
1+3=	4	4	4	4
1+4=	5	5	5	5
1+5=	6	6	6	6
1+6=	7	7	7	7
1+7=	8	8	8	8
1+8=	9	9	9	9
1+9=	10	9	10	10
2+0=	2	2	2	2
2+1=	3	3	3	3
2+2=	4	4	4	4
2+3=	5	5	5	5
2+4=	6	6	6	6
2+5=	7	7	7	7
2+6=	8	8	8	8
2+7=	9	9	9	9
2+8=	10	10	10	10

Entrada	Correcto	O	V1	V2
2+9=	11	11	11	11
3+0=	3	3	3	3
3+1=	4	4	4	4
3+2=	5	5	5	5
3+3=	6	6	6	6
3+4=	7	7	7	7
3+5=	8	8	8	8
3+6=	9	9	9	9
3+7=	10	10	10	10
3+8=	11	11	11	11
3+9=	12	12	12	12
4+0=	4	4	4	4
4+1=	5	5	5	5
4+2=	6	6	6	6
4+3=	7	7	7	7
4+4=	8	8	8	8
4+5=	9	9	9	9
4+6=	10	10	10	10
4+7=	11	11	11	11
4+8=	12	12	12	12
4+9=	13	13	13	13
5+0=	5	5	5	5
5+1=	6	6	6	6
5+2=	7	7	7	7
5+3=	8	8	8	8
5+4=	9	9	9	9
5+5=	10	10	10	10
5+6=	11	11	11	11
5+7=	12	12	12	12
5+8=	13	13	13	13
5+9=	14	14	14	14
6+0=	6	6	6	6
6+1=	7	7	7	7

Entrada	Correcto	O	V1	V2
6+2=	8	8	8	8
6+3=	9	9	9	9
6+4=	10	10	10	10
6+5=	11	11	11	11
6+6=	12	12	12	12
6+7=	13	13	13	13
6+8=	14	14	14	14
6+9=	15	15	15	15
7+0=	7	7	7	7
7+1=	8	8	8	8
7+2=	9	9	9	9
7+3=	10	10	10	10
7+4=	11	11	11	11
7+5=	12	12	12	12
7+6=	13	13	13	13
7+7=	14	15	14	14
7+8=	15	15	15	15
7+9=	16	17	16	16
8+0=	8	8	8	8
8+1=	9	9	9	9
8+2=	10	10	10	10
8+3=	11	11	11	11
8+4=	12	12	12	12
8+5=	13	13	13	13
8+6=	14	14	14	14
8+7=	15	15	15	15
8+8=	16	16	16	16
8+9=	17	17	17	17
9+0=	9	8	8	9
9+1=	10	10	10	10
9+2=	11	11	11	11
9+3=	12	12	12	12
9+4=	13	13	13	13

Entrada	Correcto	O	V1	V2
9+5=	14	14	14	14
9+6=	15	15	15	15
9+7=	16	17	16	16
9+8=	17	17	17	17
9+9=	18	18	18	18

Anexo C. Contenido del paquete de referencia

• `INFORME-FINAL.pdf`: versión de lectura, con índice y marcadores.

• `INFORME-FINAL.md`: versión editable del mismo contenido.

• `reconstruir_y_comprobar.py`: reconstrucción de las seis operaciones y comparación exacta de tensores.

• `auditar_evidencias.py`: auditoría documental de manifiestos, variantes, respuestas y diferencias.

• `anexos/106-variantes.csv` y `.json`: catálogo completo de candidatas y errores.

• `anexos/100-respuestas.csv`: resultados de las tres versiones para todo el dominio.

• `anexos/609-pesos.csv` y `.json`: cada índice modificado y sus valores anterior/nuevo.

• `anexos/fuentes-verificadas.json`: hashes de los 71 archivos auditados.

• `anexos/auditoria.json`: cifras y verificaciones derivadas para este informe.

• Las carpetas históricas V1 y V2 en el paquete integrado: pesos, scripts y registros originales de ambas intervenciones.

La versión documental describe lo ejecutado y ofrece un procedimiento transferible. Sus identificadores, índices y constantes son específicos de este checkpoint; en futuros modelos se debe repetir el diagnóstico, no copiar ciegamente el parche.