Reducción del costo marginal por visita en un servicio público de datos legislativos: de la insolvencia operativa al costo acotado

Restauración de la cacheabilidad, compresión en el codo de la curva calidad–CPU, invalidación incremental por deltas y precómputo de la resolución de referencias en el servidor

Ariel Eitner

Observatorio de Leyes — observatorio-de-leyes-front.vercel.app · Buenos Aires, Argentina · Versión 1.0, agosto de 2026

RESUMEN

El Observatorio de Leyes publica el corpus normativo argentino —8.034 normas y 90.945 artículos al cierre de este trabajo— como ~49.000 páginas estáticas servidas desde una red de distribución de contenido, con regeneración estática incremental (ISR). Entre junio y agosto de 2026 el servicio fue *suspendido tres veces por costo*: dos pausas de la plataforma de frontend por agotamiento del crédito mensual y una suspensión del backend por 12,24 GB de transferencia saliente en un ciclo. Este trabajo documenta el diagnóstico y las cuatro intervenciones que llevaron el costo por visita de un régimen proporcional al tamaño del corpus a uno acotado por constantes: (i) la *restauración de la cacheabilidad*, anulada en silencio por tres mecanismos compuestos de la semántica de caché del framework (un `force-dynamic` en el layout raíz que arrastraba a todo el árbol, rutas dinámicas sin función de parámetros estáticos y la regla del mínimo entre los `revalidate` de un segmento); (ii) la *compresión* del backend en el codo medido de la curva calidad–CPU de Brotli (×6,8–9,1 por respuesta, a 27 ms); (iii) el reemplazo de la *invalidación total* del sitio —49.342 páginas vencidas por cada actualización mayor a 400 rutas— por una invalidación *incremental por deltas* que distingue altas de ediciones (una carga de 300 normas pasa de invalidar 49.342 páginas a 305, ×162 menos); y (iv) el *precómputo en el servidor de la resolución de referencias cruzadas*, que eliminó del HTML de cada página un diccionario embebido de 2,4 MB (el 85–95 % del peso de la página, presente porque el navegador resolvía contra él el destino de cada cita normativa). La verificación se hizo sobre un banco de pruebas local reproducible comparando la *huella completa de enlaces internos* de cada página por diferencia de conjuntos —no por conteo—, lo que capturó dos regresiones invisibles a la inspección manual. Resultados medidos sobre 12 páginas representativas: 47.523 kB → 14.924 kB de HTML (−68,6 %); la página de artículo, de 2.796 kB a 65 kB (×42,7); el recorrido completo del sitio por un rastreador, de ≈57 GB a ≈3 GB estimados; el subsistema de navegación que viaja en cada página, de 2.492.565 a 7.147 bytes (×349). Ninguna página perdió un enlace interno.

Palabras clave: regeneración estática incremental · caché HTTP · invalidación de caché · compresión Brotli · precómputo · desnormalización · costo marginal · serverless · SEO · datos públicos · legislación argentina · legal tech

1 Introducción

Un servicio público de datos tiene una restricción que los servicios comerciales no suelen tener: debe poder operarse con ingresos cercanos a cero. El Observatorio de Leyes publica de forma gratuita el texto, la explicación en lenguaje claro y el grafo de relaciones del corpus normativo argentino; su infraestructura corre sobre los niveles gratuitos o mínimos de tres proveedores (frontend en CDN con funciones serverless, backend en un contenedor de 256–512 MB, base de datos gestionada). En ese régimen, el costo no es una

molestia contable: es la *condición de existencia* del servicio. Cuando el consumo supera el tope, el proveedor no degrada el servicio — lo **apaga**.

Eso ocurrió tres veces en el período estudiado. El 29 de junio de 2026 la plataforma de frontend pausó el proyecto por consumo; el 6 de agosto el operador debió suspender el backend tras 12,24 GB de transferencia saliente en un ciclo; el 19 de agosto el frontend fue pausado otra vez al agotarse el 95 % del crédito mensual (18,98 de 20 dólares) con desglose conocido por rubro. La serie de incidentes convirtió una pregunta difusa ("¿por qué gasta tanto?") en un problema de ingeniería con función objetivo medible: **minimizar el costo marginal de servir una página, sin sacrificar el contenido servido ni la indexabilidad**.

La tesis de este trabajo es que ese costo no era una función del tráfico sino del *diseño*: (a) páginas declaradas estáticas que en realidad se regeneraban en cada visita, por composición de reglas de caché que se anulan entre sí sin aviso; (b) respuestas sin comprimir, un multiplicador de ×6,8–9,1 sobre toda la transferencia; (c) una política de invalidación que, superado un umbral, marcaba como vencido *el sitio entero*; y (d) un diccionario de resolución de 2,4 MB embebido en el HTML de cada una de las ~49.000 páginas, necesario solo porque una decisión de arquitectura había delegado al navegador un cómputo que el servidor podía precalcular. Ninguna de las cuatro intervenciones "hace lo mismo más rápido": todas cambian la *clase* del costo — de proporcional al tamaño del corpus o al del sitio, a constante o proporcional al cambio — siguiendo las reglas clásicas de precomputación y evaluación perezosa [6].

El artículo se organiza así: §2 describe el sistema y formaliza su modelo de costo; §3 presenta el diagnóstico experimental de los tres episodios; §4 detalla las intervenciones y sus algoritmos; §5 la metodología de verificación, que es en sí misma una contribución (la *huella de enlaces* por diferencia de conjuntos); §6 los resultados; §7 el análisis asintótico consolidado; §8 limitaciones y amenazas a la validez; §9 el trabajo futuro.

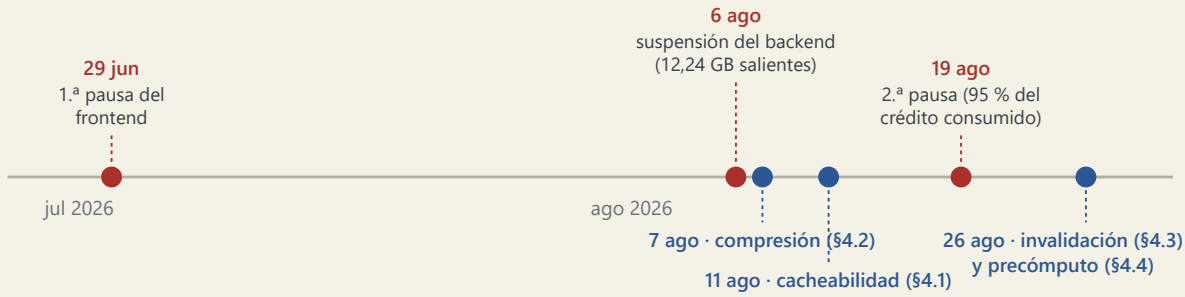


Figura 1. Línea de tiempo del período estudiado. En rojo, los tres incidentes de suspensión por costo; en azul, las intervenciones. Las de §4.3 y §4.4 se diseñaron el 13 de agosto y se implementaron y verificaron el 26; al cierre de esta versión esperan su despliegue, deliberadamente pospuesto para el reinicio del ciclo de facturación (§8.3).

2 El sistema y su modelo de costo

2.1 Arquitectura

El sistema tiene tres capas. Una **base de datos** PostgreSQL 17 gestionada guarda el corpus (normas, artículos, segmentos de explicación, relaciones tipadas entre normas, categorías temáticas). Un **backend** NestJS mantiene en memoria un índice liviano del corpus ("registro": identificador, número, nombre, ruta

pública, alias, estado de vigencia de cada norma) y sirve las normas completas bajo demanda. Un **frontend** Next.js genera las páginas con regeneración estática incremental [9]: cada URL se renderiza una vez, el resultado queda en la caché de la CDN y las visitas siguientes lo reciben sin ejecutar código; cuando el corpus cambia, el backend notifica al frontend qué rutas invalidar y estas se regeneran recién en la próxima visita.

El contenido tiene una particularidad que estructura todo el problema: es un *hipertexto denso*. El texto de una norma cita otras normas constantemente ("conforme el artículo 20 de la Ley 24.240…"), y el sistema convierte cada cita en un enlace interno con estado (vigente, derogada, no cargada). Esos enlaces son, además, la columna vertebral de la indexabilidad: el rastreador de un buscador solo descubre y pondera el grafo interno del sitio si los enlaces están presentes en el HTML que el servidor entrega [8].

2.2 El modelo de costo

La factura mensual del frontend se descompone en rubros medidos por el proveedor; los dominantes en el episodio de agosto fueron *escrituras de ISR* (cada regeneración escribe la página en la caché), *transferencia de origen* (los bytes que el origen entrega a la CDN) y *CPU de funciones*. Los tres son proporcionales a la misma suma:

$$T = \Sigma_{p \in \text{páginas}} w_p \cdot r_p \tag{1}$$

donde w_p es el peso de la página p y r_p la cantidad de veces que se regenera en el mes. Un sitio sano de ~49.000 páginas estáticas tiene $r_p \approx 0$ para casi todas (se regeneran solo las que cambian) y w_p acotado por el contenido real. El diagnóstico de §3 muestra que este sistema violaba los dos factores a la vez: r_p era proporcional a las visitas (nada se cacheaba, §3.2) o al tamaño del sitio (invalidación total, §3.3), y w_p contenía un término proporcional al tamaño del corpus (el diccionario embebido, §3.3):

$$w_p = w_{\text{contenido},p} + |D| \quad \rightarrow \quad w_p = w_{\text{contenido},p} + c_{\text{nav}} + \Sigma_{ref \in p} |t_{ref}| \tag{2}$$

con $|D|$ el diccionario completo (2.434 kB con el corpus actual, y *creciente con cada norma cargada*), c_{nav} una constante de 7 kB y t_{ref} el destino precalculado de cada referencia presente en la página (§4.4). La transformación de (2) es el corazón del trabajo: el término dependiente del corpus desaparece del peso de *todas* las páginas y es reemplazado por un término proporcional a las referencias *de esa página*.

Para dimensionar $|D|$: la mediana mundial de peso *total* de una página de escritorio ronda los 2,3 MB [10]. Solo el diccionario embebido de este sistema ya la igualaba — antes de contar un byte de contenido.

3 Diagnóstico experimental

3.1 Episodio 1 (junio): rastreadores × imágenes generadas por solicitud

La primera pausa combinó tres factores multiplicativos: (a) la política de robots permitía explícitamente 13 rastreadores de inteligencia artificial, incluidos dos de los más agresivos y sin tráfico de referencia hacia el sitio; (b) el mapa del sitio expone el corpus completo (decenas de miles de URL); y (c) once rutas de imágenes para redes sociales (*Open Graph*) se rasterizaban *en cada solicitud* —composición tipográfica con Satori en tiempo de función, sin directiva de revalidación—, de modo que cada visita de cada rastreador a cada URL pagaba CPU y memoria de una generación de imagen. La corrección fue declarativa: revalidación semanal en las once rutas y exclusión de los dos rastreadores sin valor. La lección estructural —un costo por solicitud donde debía haber un costo por contenido— reaparece en los tres episodios.

3.2 Episodio 2 (agosto): la jerarquía de caché anulada en silencio

La verificación más simple posible —pedir dos veces seguidas la misma URL y mirar el encabezado de caché de la CDN— dio el resultado más grave: `MISS` seguido de `MISS`, con `age = 0`, en las páginas de mayor tráfico. **El sitio entero se renderizaba en cada visita**, pese a declarar caché permanente. Tres mecanismos compuestos lo causaban, y ninguno emite advertencia alguna:

- **Un `force-dynamic` en el layout raíz arrastra a todo el árbol.** Estaba puesto por un motivo menor (evitar un menú vacío durante la compilación) y anulaba el `revalidate = false` de todas las páginas descendientes.
- **Una ruta con segmento dinámico y sin función de parámetros estáticos nunca entra a la caché de ISR.** Las cinco rutas de artículo (~49.000 URL, el grueso del sitio) no la tenían. El hallazgo útil es que la función puede *devolver la lista vacía*: el framework solo exige que exista; cuántas rutas se prerenderizan es una decisión independiente — y prerenderizar miles resultó contraproducente (véase el recuadro).
- **El período de revalidación efectivo de un segmento es el mínimo entre los del layout, la página y cada `fetch` que ejecuta.** Dos funciones de acceso a datos llevaban un valor por defecto de 300 s, una de ellas en el layout raíz: un techo de cinco minutos sobre el caché "permanente" de todo el sitio. Como corolario, un solo `fetch` con `no-store` vuelve dinámica la ruta completa — así, por sus estadísticas, la portada era la última página pública dinámica del sitio.

> **RESULTADO NEGATIVO: PRERENDERIZAR EL CORPUS NO ESCALA**
>
> Antes de adoptar la lista vacía se intentó prerenderizar las fichas en compilación. Tres fracasos en escalera el mismo día: 3.365 fichas superaron el tiempo límite de 60 s por página contra un backend de 0,2 vCPU; con 107, el backend no devolvió error sino la respuesta *truncada*, y el error de JSON incompleto abortó el despliegue; hasta con 4 códigos hubo páginas rescatadas solo por el reintento del compilador. Con cero prerenderizadas: 416 páginas en 21 s. La regeneración bajo demanda no es una concesión: es la única política compatible con un backend de recursos mínimos.

Medido tras la corrección: la ficha pasó de `Cache-Control: s-maxage=300` a `s-maxage=31536000`, con `MISS` en la primera visita y `HIT` después. El contexto de tráfico hace evidente la magnitud del desperdicio previo: el backend recibía ~1.600 solicitudes/hora generadas por renderizados de páginas "estáticas", contra **46 sesiones humanas por día**. El análisis de visitantes del período mostró además que el 55,8 % (3.165 de 5.669) eran procesos en centros de datos que se presentaban con agente de usuario de navegador legítimo:

0,95 páginas por visitante, 6,1 s de sesión promedio (2.745 de 3.113 con exactamente 0 s) y cero interacciones. Con caché, ese tráfico cuesta ancho de banda de CDN; sin caché, costaba una regeneración completa por visita.

3.3 Episodio 3 (agosto): el diccionario embebido y la invalidación total

Con la cacheabilidad restaurada, el crédito igual se agotó (18,98 de 20 dólares el 13 de agosto; Figura 6). Las dos causas restantes eran de diseño:

(a) El diccionario en el HTML. Para convertir cada cita normativa en un enlace con estado, el navegador resolvía el destino contra el registro completo del corpus, y para eso el registro viajaba serializado dentro del HTML de *cada* página. Medido contra producción (corpus de 3.110 normas): una página de artículo pesaba 1.163 kB, de los cuales 1.100 kB —el **94,6 %**— eran las entradas del registro. Sobre el banco de pruebas local con el corpus actual (8.034 normas listadas) la proporción se agrava: el registro serializa 2.434 kB, de los que el campo de normas son 2.427 kB (**99,7 %**); lo que el navegador necesita de verdad para el armazón de la página —categorías temáticas, alias de rutas, normas referenciadas no cargadas— son **6,9 kB** (Figura 2). El término $|D|$ de la ecuación (2) crecía además con cada norma cargada: el éxito editorial del proyecto encarecía cada página servida.

(b) La invalidación total. El protocolo de notificación entre backend y frontend tenía una válvula de escape: por encima de 400 rutas cambiadas, enviaba la orden "invalidar todo", que el frontend ejecutaba como una invalidación desde la raíz — **las 49.342 páginas del sitio marcadas como vencidas de una vez**. Una sola ley de 400 artículos alcanzaba para dispararla. Tras cada disparo, cada rastreador que recorriera el mapa del sitio pagaba una regeneración de ~1,16 MB por página: la combinación (a) × (b) explica el rubro dominante de la factura (escrituras de ISR) y el pico de transferencia de origen. Una pasada completa del sitio en ese estado se estima en ≈57 GB.

3.4 El backend: transferencia sin comprimir

La suspensión del backend (12,24 GB salientes en un ciclo) tuvo una causa desproporcionadamente simple: el servidor HTTP no montaba compresión alguna, y ningún elemento intermedio la agregaba. Todas las respuestas —incluido el registro de ~1,6 MB y normas completas de hasta 2,3 MB— viajaban crudas. Con compresión, la misma transferencia se estima en ~1,5 GB. El mismo ciclo de correcciones incorporó la carga *perezosa* del corpus (memoria residente del contenedor: 69 % → 33 %; arranque: 7 min → 18 s), a costa de trasladar lecturas a la base de datos — un intercambio explícito de RAM por transferencia interna que luego exigió su propia ronda de optimización.

4 Intervenciones

4.1 Restauración de la cacheabilidad

Es la intervención de mayor rendimiento por línea de código cambiada, y la más difícil de *ver*: consiste en quitar tres anulaciones (el `force-dynamic` del layout raíz; los `revalidate: 300` por defecto en las funciones

de acceso a datos, reemplazados por caché permanente con invalidación por evento; el `no-store` en los datos de la portada) y en agregar a las once rutas de norma una función de parámetros estáticos que devuelve la lista vacía. La regla operativa que queda para el proyecto: *la cacheabilidad de una página no se deduce leyendo su archivo* — es una propiedad compuesta de todo su árbol de layouts y de cada acceso a datos que ejecuta, gobernada por la regla del mínimo; solo la verificación externa (dos solicitudes, encabezados de caché) da fe de ella. La caché HTTP obedece la semántica estándar [3]; el punto ciego está en la capa de composición del framework.

4.2 Compresión en el codo de la curva calidad–CPU

El backend montó compresión negociada por `Accept-Encoding` (Brotli [1] con reserva gzip [2]). La calidad de Brotli se eligió midiendo la curva completa sobre el mayor payload del sistema (1,34 MB), no por defecto (Figura 3): la calidad 5 comprime a 195 kB en 27 ms; la calidad máxima (11) solo mejora a 167 kB pero cuesta 2.297 ms de CPU — 85 veces más tiempo por un 14 % menos de bytes. La calidad 5 es el codo de la curva y la elección para comprimir en línea; la calidad 11 solo se justifica para artefactos comprimidos una vez fuera de línea. Sobre los endpoints reales: registro 1.660 → 206 kB (×8,1), norma completa 2.320 → 254 kB (×9,1), grafo 1.038 → 159 kB (×6,5). Dos advertencias de medición que costaron falsas alarmas: las herramientas de alto nivel (Invoke-WebRequest, fetch de Node) *descomprimen de forma transparente y suprimen el encabezado*, haciendo creer que nada comprime — hay que medir con curl y bytes en cable; y `Vary: Accept-Encoding` es obligatorio, o un caché intermedio sirve la variante comprimida a un cliente que no la soporta.

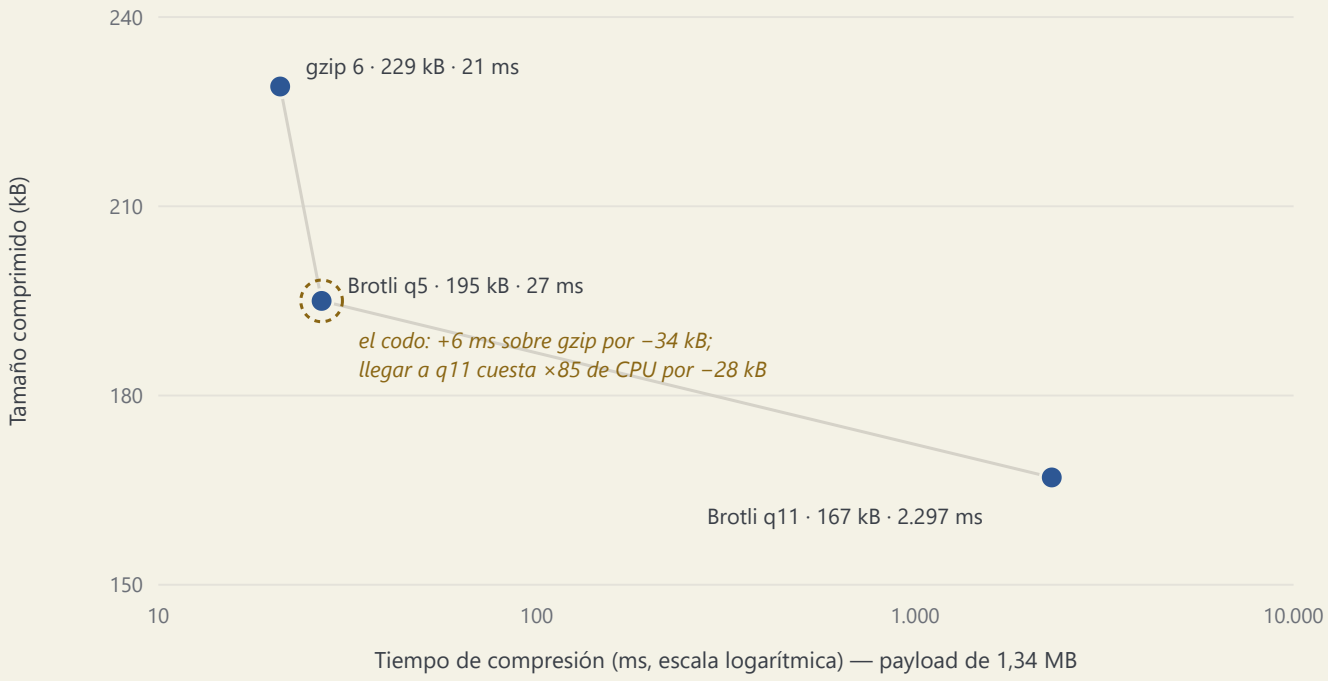


Figura 3. Curva calidad–CPU medida sobre el payload más pesado del sistema (1,34 MB). El eje horizontal es logarítmico (posición, no longitud, codifica el valor). Brotli calidad 5 domina: casi todo el beneficio de compresión a costo de CPU marginal. La calidad por defecto "máxima" habría multiplicado por 85 el costo de CPU de cada respuesta no cacheada.

4.3 Invalidación incremental por deltas

La invalidación total se eliminó del protocolo. La observación que lo permite es que **alta y edición no cuestan lo mismo**: una norma *nueva* no tiene ninguna página generada todavía — invalidar sus artículos no destraba nada, porque no hay nada cacheado que destrabar; lo único que puede haber quedado viejo es un 404 de su ficha, cacheado antes de que la norma existiera. Una norma *editada*, en cambio, sí tiene páginas cacheadas con el texto anterior: ficha y artículos deben invalidarse. El protocolo nuevo enumera exactamente eso y lo envía en lotes acotados (500 rutas por solicitud, con el receptor alineado al mismo límite para no descartar rutas en silencio):

ALGORITMO 1 — INVALIDACIÓN POR DELTAS (REEMPLAZA A LA INVALIDACIÓN TOTAL)

```
entrada: C = normas cambiadas, E = normas eliminadas, N ⊆ C = altas (sin sello previo)
// rutas afectadas de verdad — no "todo por las dudas"
R ← ∅
para cada norma n ∈ C:
    R ← R ∪ { ficha(n) }                          // alta o edición: la ficha siempre
para cada norma n ∈ C \ N:                         // SOLO ediciones: sus artículos tienen cac
    R ← R ∪ { ficha(n)/articulo/k : k ∈ artículos(n) }   // una única consulta agregada a
para cada norma n ∈ E:  R ← R ∪ { ficha(n) }
R ← R ∪ Índices                                    // portada, listados por tipo, mapa del si
enviar R en lotes de ≤ 500 rutas                   // tamaño alineado con el tope del receptor
// invariante: |R| = O(|C| + Σ artículos de las editadas + k) — independiente del tamaño de
```

El costo pasa de N_{sitio} páginas vencidas por actualización a la ecuación (3):

$$N_{\text{vencidas}} = 49.342 \quad \rightarrow \quad N_{\text{vencidas}} = |\text{altas}| + \Sigma_{n \in \text{editadas}}(1 + a_n) + k \tag{3}$$

Para el caso de uso típico —una tanda de carga de 300 normas nuevas— la invalidación cae de 49.342 páginas a 305 (300 fichas + 5 índices): **×162 menos** (Figura 4). El algoritmo elimina de paso la consulta más cara del ciclo de actualización (la enumeración de números de artículo), que ahora solo se ejecuta para las editadas. La orden "invalidar todo" sobrevive únicamente como herramienta manual, para el único caso en que es verdad (un cambio de plantilla global).

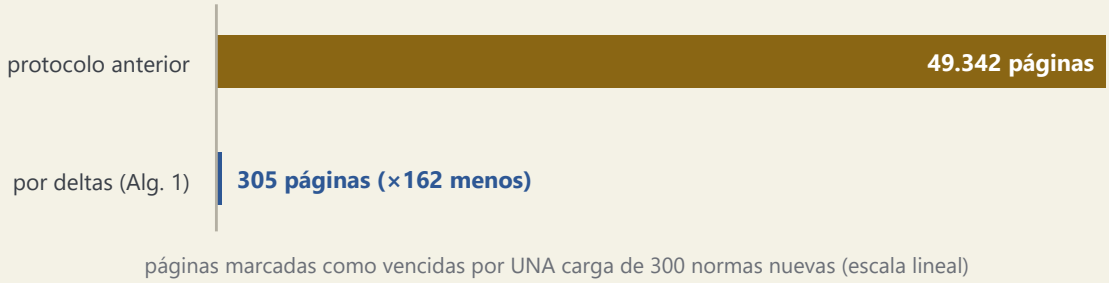


Figura 4. El mismo evento editorial —cargar 300 normas— bajo los dos protocolos. La barra inferior es real: a escala lineal, 305 páginas apenas se distinguen del eje. Cada página vencida se convierte, con la próxima visita de un rastreador, en una regeneración pagada (escritura de ISR + transferencia de origen + CPU).

4.4 Precómputo de la resolución de referencias en el servidor

La intervención central. El backend ya analizaba sintácticamente las citas normativas de cada texto (un analizador de expresiones regulares con grupos nombrados, espejado en el cliente); lo que delegaba al navegador era la *resolución*: convertir "Ley 24.240" en un destino navegable con estado. El rediseño adjunta a cada cita su destino resuelto —un objeto t_{ref} con ruta, nombre completo de la norma, disponibilidad, marca de derogada, ficha mínima si la norma no está cargada, y marca de autorreferencia— y reduce lo que viaja en cada página al armazón de navegación (7.147 bytes: categorías, alias, normas referenciadas no cargadas).

ALGORITMO 2 — RESOLUCIÓN DE UNA CITA EN EL SERVIDOR (ANTES: EN EL NAVEGADOR, CONTRA EL DICCIONARIO COMPLETO)

```
índices (se construyen una vez por versión del corpus, se descartan al cambiar el corpus):
    porSigla, porAlias, porId : código → {ruta, nombre, disponible, estado, alias[]}   // (
    stubPorNúmero : número → ficha mínima de norma no cargada

resolver(cita, artículo?, esPropia):
    e ← porSigla[cita] ‖ porAlias[normalizar(cita)] ‖ porId[derivarId(cita)]
    si esPropia:           devolver {isSelf, nombre: e.nombre, ruta: e.ruta}   // conserva
                                                                  // múltiples ("arts. 42–44"
    si e ∄ o no disponible: devolver stubPorNúmero[número(cita)] ‖ {no disponible}
    ruta ← artículo? e.ruta + "/articulo/" + slug(artículo) : e.ruta
    devolver {ruta, nombre: e.nombre, disponible, derogada si corresponde}

// REQUISITO DE PARIDAD: resolver() es un port exacto del resolutor del cliente (incluida l
// normalización de números "26485" → "26.485"). Si divergen, una cita enlaza en un lado y
// en el otro, sin error visible. La propiedad se verifica extensionalmente en §5.
```

Tres decisiones subordinadas completan el diseño:

- **Se resuelve todo texto que llega al HTML, no solo el articulado.** El resumen ejecutivo, el objetivo, las listas de obligaciones y derechos, las respuestas de preguntas frecuentes y las reglamentaciones de cada artículo también citan normas; omitirlos habría borrado los enlaces internos de las ~8.000 fichas. La verificación de §5 detectó exactamente esa omisión como regresión.
- **El diccionario completo sobrevive como carga perezosa.** Cinco componentes interactivos (el buscador de normas guardadas de la cuenta, el modal de conversión, la vista de impresión, entre otros) sí necesitan el catálogo; lo descargan del backend *después* de la hidratación, una sola vez, y solo cuando el usuario llega a esa interacción — el botón de guardar, presente en todas las páginas de norma, lo pide recién al abrirse su panel, porque pedirlo al montar habría reintroducido el costo entero por otra vía.
- **Un segundo ahorro estructural, invisible en el peso de página:** la caché de datos del framework descarta entradas mayores a 2 MB [9]. El registro (2,49 MB) no entraba, de modo que el layout lo descargaba del backend *en cada generación de página*. El armazón de 7 kB entra con holgura: la descarga repetida desaparece.

La restricción que prohíbe la alternativa obvia ("resolver en el cliente y no mandar nada") es la indexabilidad: los enlaces resueltos en el navegador no existen en el HTML que entrega el servidor, y la documentación del

propio buscador dominante recomienda no depender del renderizado de JavaScript para el enlazado interno crítico [8]. El precómputo en el servidor es el único punto del espacio de diseño que minimiza el peso *y* conserva el grafo de enlaces en el HTML.

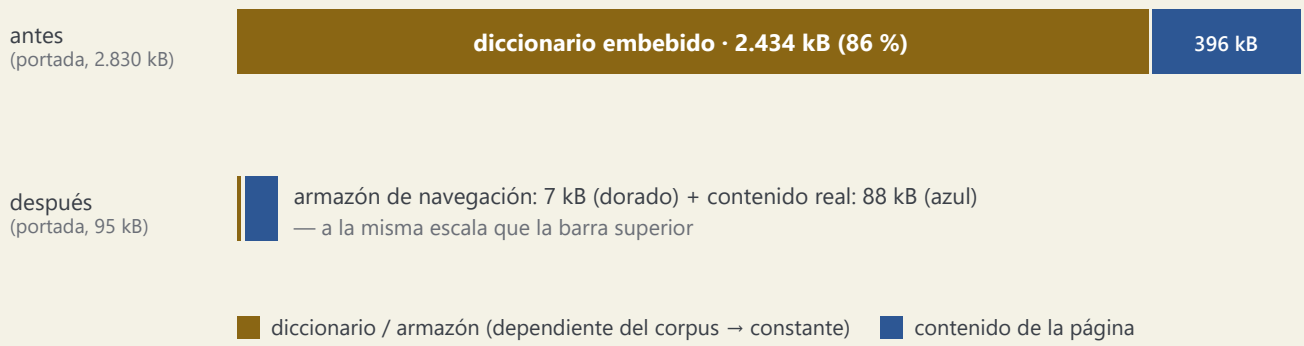


Figura 2. Composición del HTML de la portada, a escala, antes y después de §4.4. El bloque dorado de la barra superior —2.434 kB, del cual el 99,7 % es el catálogo de las 8.034 normas— viajaba en *todas* las páginas del sitio y crecía con cada norma cargada. Su reemplazo mide 7 kB y es independiente del tamaño del corpus.

5 Metodología de verificación

5.1 Banco de pruebas

Todas las mediciones de §6 se tomaron sobre un banco local reproducible: la pila completa en contenedores (PostgreSQL 17, backend en :4600, frontend en :4000, build de producción), con un corpus de 8.334 normas en base (8.034 públicas listadas) y 90.945 artículos. Protocolo por página: (i) solicitud con `Accept-Encoding: identity` para medir el HTML real —las herramientas de alto nivel descomprimen en silencio y mienten el tamaño por ~6×—; (ii) registro del código de estado; (iii) extracción de *todos* los `href` internos del documento, ordenados. Dos precauciones no obvias: tras cada reconstrucción de la pila se invalida el caché completo antes de medir, porque las páginas prerenderizadas durante la compilación se generan con el backend aún arrancando y salen sin datos (50 kB en lugar de 2.830: medir sin invalidar compara una página rota contra una sana); y las URL de muestra usan las rutas canónicas del registro, no las abreviadas — estas responden 308 y se termina midiendo la redirección.

5.2 La huella de enlaces: verificación por diferencia de conjuntos

El riesgo específico de esta refactorización es que su modo de falla es *silencioso*: si la resolución diverge, el HTML sigue siendo válido, la página pesa poco (¡mejor aún para la métrica ingenua!) y lo que desaparece son enlaces internos — exactamente lo que el rastreador de un buscador dejaría de ver, sin que ninguna prueba de tipo o de render lo detecte. La verificación adoptada define como invariante la **huella de enlaces** de cada página: el multiconjunto ordenado de sus `href` internos. La condición de aceptación es de igualdad de conjuntos, no de cardinalidad:

$$\forall p \in \text{muestra}:\ H_{\text{antes}}(p) \setminus H_{\text{después}}(p) = \emptyset \quad \text{(ningún enlace perdido; los ganados se auditan a mano)} \tag{4}$$

El conteo no alcanza: dos versiones de una página pueden tener 33 enlaces cada una y ser conjuntos distintos. La distinción no fue teórica — la verificación capturó dos regresiones reales que la inspección visual y la compilación en verde habían dejado pasar:

Tabla 1. Regresiones capturadas por la huella de enlaces durante el desarrollo de §4.4. Ambas invisibles para el compilador, las pruebas unitarias y la inspección visual rápida.

REGRESIÓN	SÍNTOMA EN LA HUELLA	CAUSA	CORRECCIÓN
Las preguntas frecuentes perdieron sus enlaces	`PERDIDO: /codigos/penal`	El acordeón de preguntas frecuentes no recibía citas resueltas y caía al analizador del cliente — que, sin diccionario, rinde texto plano. Efecto secundario: disparaba la descarga perezosa de 2,4 MB en cada ficha con preguntas.	Resolver también las respuestas de preguntas frecuentes en el servidor (`refsFicha.faq`).
La cita múltiple autorreferente perdió su destino	El enlace reaparecía pero como `href="#"`	La marca de autorreferencia descartaba la ruta; las citas de rangos ("arts. 42–44" dentro del propio Código Penal) la necesitan aunque no muestren el nombre de la norma.	La autorreferencia conserva la ruta y solo suprime el nombre redundante.

La segunda regresión motiva una nota metodológica: la primera corrida de la verificación reportó "1 página perdió 1 enlace". La reacción correcta no fue relajar el criterio sino perseguir ese único enlace hasta su causa — que resultó ser una regla semántica genuina del dominio (una cita múltiple a la propia norma *sí* se enlaza; una cita simple, no). Un criterio de aceptación por umbral ("99 % de los enlaces sobreviven") habría absorbido el error.

6 Resultados

6.1 Peso de página

Tabla 2. HTML por página (kB, sin comprimir en el cable), muestra de 12 páginas que cubre cada plantilla del sitio. "Enlaces" = huella de enlaces internos antes → después; la columna final es la condición (4).

PÁGINA	ANTES (KB)	DESPUÉS (KB)	FACTOR	ENLACES	PERDIDOS
/leyes (índice)	2.769	34	**×81,5**	`30 → 30`	0
Artículo 79, Código Penal	2.796	65	**×42,7**	`33 → 33`	0
Artículo 14, Constitución Nacional	2.799	73	×38,1	`40 → 40`	0
Portada	2.830	95	×29,8	`38 → 38`	0
/categorias	2.844	109	×26,1	`67 → 67`	0
Artículo 171, ley 18.345	2.891	185	×15,6	`62 → 62`	0
Artículo 217, ley 11.683 (224 citas)	3.008	298	×10,1	`130 → 130`	0
Ficha de la ley 24.240	3.126	401	×7,8	`26 → 26`	0
Ficha de la Constitución Nacional	3.449	758	×4,6	`46 → 46`	0
Ficha del Código Penal	3.722	1.024	×3,6	`28 → 28`	0
Ficha de la ley 11.683	3.858	1.185	×3,3	`26 → 26`	0
/corpus (catálogo completo)	13.432	10.696	×1,3	`8.055 → 8.055`	0
Total de la muestra	**47.523**	**14.924**	**×3,2 (−68,6 %)**		**0**

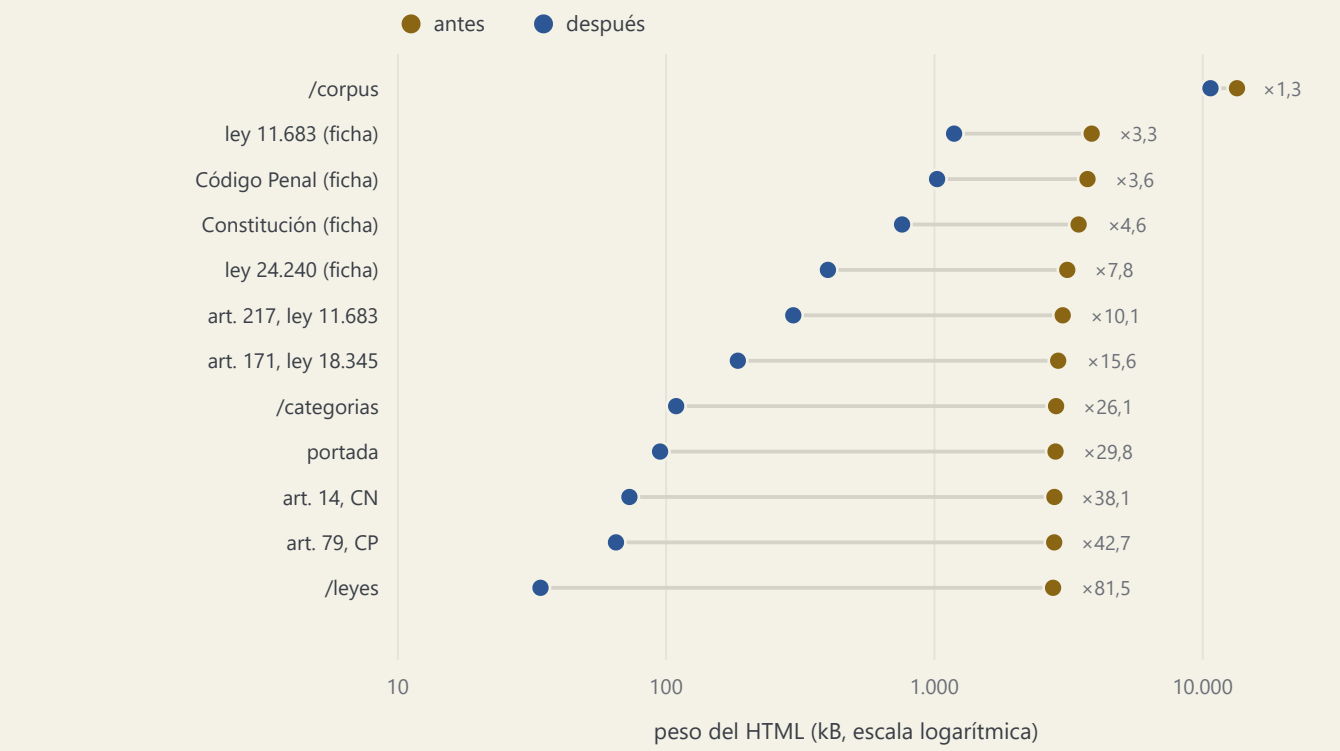


Figura 5. Peso del HTML por página, antes (dorado) y después (azul), escala logarítmica. El patrón es informativo: *todas* las páginas "antes" pesan lo mismo (~2,8 MB) sin importar su contenido — la firma del término constante $|D|$ de la ecuación (2). "Después", el peso vuelve a ser función del contenido: el índice pesa 34 kB, la ficha del Código Penal 1.024 kB. La excepción (/corpus, ×1,3) es la página cuyo contenido *es* el catálogo: allí el diccionario no era sobrecarga.

6.2 El endpoint de armazón

Tabla 3. Lo que viaja hacia el HTML de cada página: endpoint anterior (registro liviano) contra el nuevo (armazón de navegación). Bytes exactos medidos contra el backend local.

ENDPOINT	CRUDO (B)	BROTLI (B)	CONTENIDO
`/laws/registry/light` (anterior)	2.492.565	342.732	8.034 normas (99,7 % del peso) + categorías + alias + referencias no cargadas
`/laws/registry/nav` (nuevo)	**7.147**	**2.520**	categorías (42) + alias (7) + referencias no cargadas (5)
Reducción	**×349**	**×136**	y el nuevo entra en la caché de datos (límite duro: 2 MB [9]); el anterior no entraba y se descargaba en cada generación

6.3 Proyección sobre el ciclo de facturación

La Figura 6 muestra el desglose real del episodio del 13 de agosto. Las intervenciones atacan directamente los tres rubros dominantes (78 % de la factura): las escrituras de ISR y la transferencia de origen son proporcionales a T de la ecuación (1) —reducida por §4.3 en el factor de regeneraciones y por §4.4 en el peso—, y la CPU de funciones al número de renderizados —reducido por §4.1—. La estimación de la pasada

completa del sitio por un rastreador (49.342 páginas × peso de página de artículo) cae de ≈57 GB a ≈3 GB. No se afirma un número de factura futuro: la validación final es empírica y queda para el ciclo posterior al despliegue (§8.3).

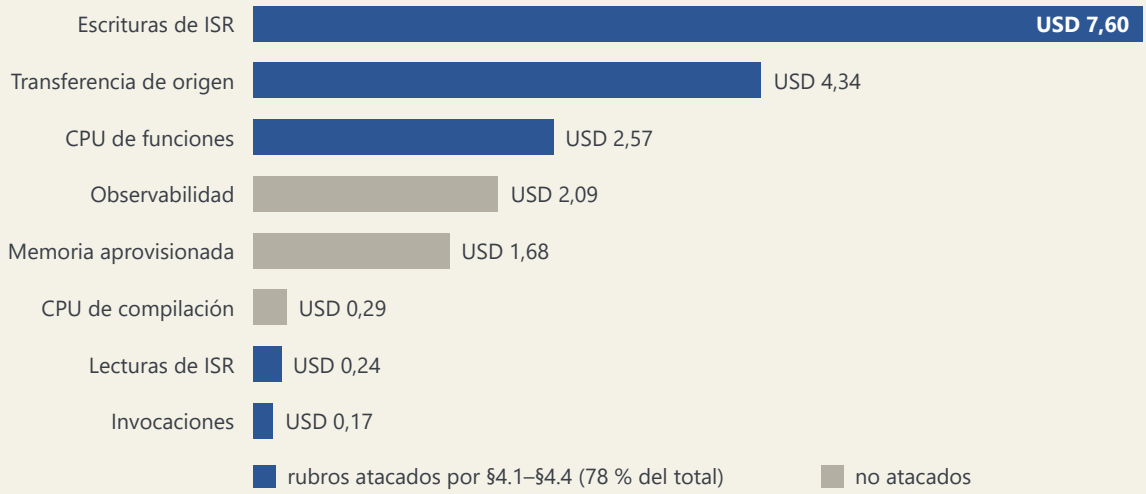


Figura 6. Desglose del consumo del ciclo de agosto de 2026 (total USD 18,98 sobre un tope de 20; al alcanzarlo, el proveedor pausa el sitio). Los tres rubros dominantes son funciones directas de la ecuación (1). La observabilidad es un interruptor del panel; la memoria aprovisionada y la compilación no dependen del tráfico. Las cifras se publican deliberadamente: la transparencia de costos es parte del proyecto (§10).

7 Análisis asintótico consolidado

Con N = páginas del sitio (~49.000), $|R|$ = normas del corpus (8.034 y creciente), V = visitas, $|\Delta|$ = normas cambiadas en una actualización, a = artículos de una norma editada, *refs(p)* = citas en la página p:

Tabla 4. Clase de costo de cada operación, antes y después. El patrón común: eliminar del costo por visita y por actualización todo término proporcional a N o a $|R|$.

OPERACIÓN	ANTES	DESPUÉS	INTERVENCIÓN
Renderizados del servidor por visita a página estática	O(1) por visita (siempre renderiza)	O(1) amortizado → 0 (HIT de CDN)	§4.1
Bytes de una respuesta del backend	tamaño crudo	crudo/6,8…9,1 (+27 ms CPU)	§4.2
Páginas invalidadas por actualización del corpus	O(N) — el sitio entero	O(|Δ| + Σa + k)	§4.3
Peso del HTML de una página	O(contenido + |R|)	O(contenido + Σ$\|t_{ref}\|$), refs de esa página	§4.4
Resolución de una cita	en el navegador, con |R| entregado antes	O(1) en el servidor (tres mapas por versión del corpus)	§4.4
Descarga del registro por generación de página	O(|R|) por generación (no entra en la caché de datos)	O(1): 7 kB cacheados	§4.4
Descarga del catálogo por el navegador	O(|R|) en cada página (embebido)	0 en lectura; O(|R|) una vez, solo al interactuar	§4.4
Costo editorial de cargar una norma nueva	hasta O(N) (invalidación total) + engorda todas las páginas	O(1): su ficha + índices	§4.3 + §4.4

La última fila es la síntesis del trabajo desde el punto de vista del proyecto: antes, cada norma cargada *encarecía el sitio entero* (más bytes en cada página, más páginas vencidas por actualización); después, el costo de crecer el corpus es local a la norma cargada. La sostenibilidad económica dejó de competir con la misión editorial.

8 Limitaciones y amenazas a la validez

8.1 Limitaciones del diseño

- **Unas ~15 páginas quedan fuera del precómputo** (páginas legales, guías, calculadoras): su texto no proviene del corpus, así que sus citas se resuelven en el navegador contra el catálogo perezoso, tras la hidratación — con un pre-test léxico barato para no descargar 2,4 MB por un párrafo que no menciona norma alguna. Sus enlaces internos no están en el HTML del servidor; si alguna vez importan para la indexabilidad, la salida es resolverlas también en el servidor.

- **El catálogo perezoso sigue pesando O(|R|)** (2,4 MB crudos, 343 kB comprimidos) para los cinco componentes interactivos que lo usan. Es una descarga por sesión interactiva, no por página; su partición o paginación queda como trabajo futuro.
- **/corpus casi no mejora (×1,3), por construcción:** es la página cuyo contenido es el catálogo. No es un fracaso del método sino su caso degenerado.
- **El primer recorrido tras cada despliegue sigue pagándose:** un despliegue vacía la caché de ISR y la primera visita de cada página regenera. La reducción de peso convierte ese evento de ≈57 GB en ≈3 GB, pero no lo elimina.

8.2 Amenazas a la validez

- **Banco local vs. producción.** Las mediciones de §6 se tomaron en la pila local, cuyo corpus (8.034 normas) es mayor que el de producción al momento del último episodio (3.110). Esto *agranda* el término $|D|$ medido; la dirección del efecto favorece al "antes" en términos absolutos, pero el factor de mejora por página es robusto porque ambas versiones se midieron sobre el mismo corpus. Las cifras de facturación son del panel del proveedor, no reproducibles localmente.
- **Corridas únicas.** Cada página se midió una vez por versión. Para tamaños de HTML estático el determinismo es alto (el mismo build produce los mismos bytes salvo identificadores de build); no se reportan intervalos.
- **Muestra de 12 páginas.** Cubre cada plantilla del sitio, no cada página. La generalización a las ~49.000 descansa en que las páginas de una misma plantilla difieren solo en el contenido, y el término eliminado era idéntico en todas.
- **La estimación de ≈57 GB / ≈3 GB** es aritmética (páginas × peso medio de plantilla), no una medición de un rastreo real.

8.3 Estado de despliegue

Al cierre de esta versión, §4.1 y §4.2 están en producción desde el 7–11 de agosto; §4.3 y §4.4 están implementadas, verificadas en el banco local y confirmadas en dos commits, con el despliegue deliberadamente pospuesto: el sitio está pausado por el tope del ciclo, y desplegar vacía la caché de ISR — conviene hacerlo al reinicio del ciclo, con el peso nuevo ya activo para que la primera pasada de rastreadores cueste ≈3 GB y no ≈57.

9 Trabajo futuro

- **Particionar el catálogo perezoso** (por inicial, por tipo de norma, o un índice invertido servido por término de búsqueda) para que la interacción tampoco pague O($|R|$).
- **Comprimir los manejadores de ruta del frontend restantes:** el framework no comprime las respuestas de `app/api/**`; de 74, solo el del grafo legal comprime a mano (1,34 MB → 195 kB). Los demás son proxies pequeños, pero la regla debería ser uniforme.

- **Regla de firewall para tráfico de centro de datos:** el archivo de robots es consultivo; el 55,8 % del "tráfico" identificado en §3.2 no lo consulta. Con las páginas cacheadas el costo residual es ancho de banda de CDN, que también factura.
- **Apagar la observabilidad de pago** (USD 2,09/mes, un interruptor) y evaluar imágenes Open Graph pregeneradas.
- **Medición longitudinal:** publicar la serie mensual de consumo por rubro tras el despliegue, como validación empírica de la proyección de §6.3.

10 Conclusión

Un servicio público de datos que muere por su factura no es un problema presupuestario: es un problema de diseño con presupuesto como síntoma. Este trabajo mostró que el costo de servir un corpus legislativo de ~49.000 páginas estaba dominado por cuatro decisiones invisibles a la operación normal — caché anulada por composición de reglas, transferencia sin comprimir, invalidación total y un diccionario dependiente del corpus embebido en cada página — y que las cuatro admiten correcciones que cambian la clase del costo, no su coeficiente. El resultado agregado, verificado con una condición de aceptación extensional (ningún enlace interno perdido, por diferencia de conjuntos), es un sistema donde el peso de cada página vuelve a ser función de su contenido, el costo de una actualización es proporcional a lo que cambió, y —lo decisivo para la misión— **cargar la norma número 8.035 cuesta lo mismo que costó la 100**. La ciencia aplicada aquí no es exótica: es la disciplina de medir antes de opinar, de formalizar el modelo de costo antes de optimizar, y de definir la verificación por el modo de falla real del cambio — que en un hipertexto público no es "se ve mal", sino "el grafo de enlaces se adelgaza en silencio".

Las cifras de consumo y facturación se publican deliberadamente. El Observatorio de Leyes sostiene que la infraestructura de un bien público debe poder auditarse como se audita una norma: con el texto completo a la vista.

11 Referencias

1. Alakuijala, J. y Szabadka, Z. (2016). "Brotli Compressed Data Format". *RFC 7932*, IETF.
2. Deutsch, P. (1996). "GZIP file format specification version 4.3". *RFC 1952*, IETF.
3. Fielding, R., Nottingham, M. y Reschke, J. (2022). "HTTP Caching". *RFC 9111*, IETF.
4. Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Tesis doctoral, University of California, Irvine. (El modelo de caché como restricción arquitectónica de REST.)
5. Dean, J. y Barroso, L. A. (2013). "The Tail at Scale". *Communications of the ACM*, 56(2), 74–80. (Variabilidad de latencia en sistemas servidos a escala; motiva medir percentiles, no promedios.)
6. Bentley, J. L. (1982). *Writing Efficient Programs*. Prentice-Hall. (Las reglas de precomputación y de evaluación perezosa aplicadas en §4.3–§4.4.)

7. Grigorik, I. (2013). *High Performance Browser Networking*. O'Reilly Media. (Costo real de la transferencia en la web; compresión y caché como palancas de primer orden.)
8. Google Search Central. "Understand JavaScript SEO Basics". Documentación para desarrolladores, developers.google.com. (Los enlaces críticos deben estar en el HTML servido.)
9. Vercel. "Incremental Static Regeneration" y "Caching and Revalidating: Data Cache". Documentación de Next.js, nextjs.org/docs. (Semántica de revalidación por segmento; límite de 2 MB por entrada de la caché de datos.)
10. HTTP Archive (2022). *Web Almanac*, capítulo "Page Weight". almanac.httparchive.org. (Mediana de peso total de página de escritorio ≈ 2,3 MB.)

Reproducibilidad: el arnés de medición (`scratchpad/registry-refactor/medir.ps1`, `comparar.ps1`, `revalidar.ps1`) y el código del sistema (`laws.service.ts`, `inline-refs.util.ts`, `InlineRefText.tsx`, `RegistryProvider.tsx`, `app/api/revalidate/route.ts`) acompañan al repositorio del proyecto. Mediciones de agosto de 2026 sobre la pila local en contenedores (frontend :4000, backend :4600, PostgreSQL 17) con corpus de 8.334 normas / 90.945 artículos; cifras de facturación tomadas del panel del proveedor el 13 de agosto de 2026. Este documento no contiene datos personales. Contacto: Observatorio de Leyes.