# Why pay more for AI calls, and why pay before the user asks?

*Optimize the cache; put Human-AI estimates behind Calculate so the model comes first.*

**Mike Idengren**






## Synopsis

In our JarviSIM process optimization product, a user reused the same document from their library to build a process model, and we were billed for AI work every time. The cache was on and configured correctly. We analyzed the problem and found two causes: (1) unintended gold-plating inserted by AI agents: four identifiers in the request that nobody had asked for, so the cache never matched; and (2) a Human-AI estimate that ran inside every BPMN model build, so the user paid for it and waited for it whether they wanted it or not.

Two fixes, two savings. We removed the identifiers, and the cache now answers a reused document. We moved the Human-AI estimate behind a Calculate button: the user gets the finished BPMN model first and decides when, and whether, to spend on the Human-AI estimate. We show how our jSwarm AI devops methods and tools found and fixed both, how `/jRightSize` prevents gold-plating and over-assurance that nobody asked for, and how a UX decision can cut cost and latency at the same time. JarviSIM is the product that supplies the real example.

**Audience:** AI Solution Product Managers · CIOs and CTOs · AI Developers

## The business story

JarviSIM turns a written procedure into a process model. Users keep their documents in a library and reuse them: the same purchasing procedure, built again later the same day in a new session. The AI work is the same, so the answer should come from the cache and the bill should be close to nothing. It was not. Every reuse was billed as if it were new, and the largest single cost was a Human-AI estimate that ran inside every BPMN model build, so the user paid for it and waited for it whether or not they wanted it. Paying again for the same AI work looks like pennies in a demo and becomes the budget in production.


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

COST · IN FOCUS · PRODUCT RUNTIME

### Two ways to spend less

Before: every reuse of a document paid for two things again. The BPMN model build missed the cache every time, about $0.14 a run. A Human-AI estimate ran inside every build, wanted or not, about $0.19 a run. Together that is **$0.33 a run**, or about **$396,000 a year** at our modeled volume.

✓ **AFTER · $0.05 a run, about $56,400 a year.** The cache answers most BPMN model builds, and the Human-AI estimate runs only when the user presses Calculate. That saves about $339,600 a year (modeled). Worked out in Part IV.

SPEED · IN FOCUS · PRODUCT RUNTIME

### The model first, in under a minute

Before: after pressing Build, the user waited **about 6 minutes** for the finished model. That was about 5 minutes (304 s) for a BPMN model build that missed the cache on every AI request, plus about 1 more minute for a Human-AI estimate they had not asked for (best guess).

✓ **AFTER · about 47 s (measured).** The cache answers the BPMN model build's AI requests, and the Human-AI estimate is no longer part of the wait. The user sees the finished model first.

QUALITY · HELD CONSTANT

### Same information, before and after

The user gets the same BPMN model, and the same Human-AI estimate when they ask for it, before and after the changes. Testing required quality to stay the same, so this case study measures only two non-functional requirements (NFRs): cost and speed.

## Terms used in this article

| TERM | WHAT IT MEANS HERE |
|---|---|
| **BPMN model build** | The JarviSIM capability that turns the user's document into a BPMN process model, including the correction pass and the follow-on AI steps that take the model as input. |
| **Human-AI estimate** | The JarviSIM capability that estimates the human effort and AI tokens the modeled process would take, shown as Process Metrics on the revision card. |
| **Cause1/Cache · Cause2/Estimate** | Shorthand for the two causes of the bill. Cause 1 (cache miss): four identifiers made every BPMN model build request new, so the cache missed. Cause 2 (forced Human-AI estimate): the Human-AI estimate ran inside every BPMN model build, whether the user wanted it or not. |


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

## more terms

| TERM | WHAT IT MEANS HERE |
|---|---|
| **Cache** | A store of recent AI requests and their answers, reused instead of running the model again. This article has two layers of it. Our AI gateway's response cache answers an exact repeat of a request (same model, same instructions, same content) for free. The AI provider's prompt cache reuses the unchanged beginning of a request at a low rate, even when the rest of the request is new. |
| **Cache hit** | A request the cache could reuse. At our gateway: an exact repeat of a stored request, answered without a new model run. At the provider: a request whose beginning matches a stored one; that beginning is billed at the low rate and the rest runs normally. |
| **Test run** | One reuse of the same library document: the user sends it to a BPMN model build, and the product does everything that follows. The before figures come from six test runs. |
| **On-demand calculation** | The Calculate button on the revision card runs the Human-AI estimate only when the user presses **calculate**, not automatically after every BPMN model build. |
| **jSwarm · JarviSIM** | jSwarm is our method: the loop, principles, practices, skills and scripts. JarviSIM is our product. It supplies the real development example in this article. |
| **BPMN** | Business Process Model and Notation: a standard way to draw a process diagram. |
| **LLM** | Large language model: the AI model that reads and generates text. |
| **Schema · nonce · request-local handles** | A schema defines the required data shape; a nonce is a one-time value; request-local handles are short names valid only inside one request. |
| **Billed call** | A request that did not match, so the model ran and we paid for it. |
| **AI gateway** | A proxy we run in front of the AI provider. It records every request and its spend, and keeps the response cache. |
| **GWT** | Given/When/Then acceptance scenario. |
| **NFR (non-functional requirement)** | A written rule about how the product must behave, such as "repeated builds reuse the cache". Part of what we call intent. |
| **Given/When/Then scenario** | An acceptance test written in the architect's own words: the starting state, the action, and what must be true afterwards. |
| **Agent Acceptance Test** | A walk through the real product by an AI agent, judged against the Given/When/Then scenario. |
| **jQATester** | The agent that walks the product like a user and judges each step against the acceptance scenario. |


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

| TERM | WHAT IT MEANS HERE |
|---|---|
| `/jRightSize` | Our jSwarm skill that controls gold-plating and over-assurance. It asks of every added mechanism what outcome or failure warrants it, and what smaller alternative works. |
| **Modeled volume** | Our stated assumption for scale figures: 1,000 users × 10 documents × 10 reuses a month × 12 months = 1,200,000 document reuses a year; 80% of reuses hit the cache within one day (assumed); the user presses calculate after 50% of reuses, because some users do not care about the process's Human-AI estimate (assumed). Ours, not a JarviSIM forecast. |
| **jTestEngineer** | The agent that designs and runs measured tests, such as the rebuild measurement in this article. |

PART I · INTRODUCTION

## Two causes, one expensive miss.

> *We are taking a multi-prong approach to solving the problem: LLM cache optimization techniques, as well as UX trade-off - creating calculation button instead of doing it automatically, incurring the AI cost when the user wants the information. LLM cache optimization is important, but cannot solve everything.*


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

Those are the (human) architect's words. This is a real JarviSIM development example, used to illustrate jSwarm principles, practices, skills and scripts. The before figures come from our AI gateway's records of six test runs, each reusing the same library document before the fix; the after figures come from measured rebuilds of that same document after it.

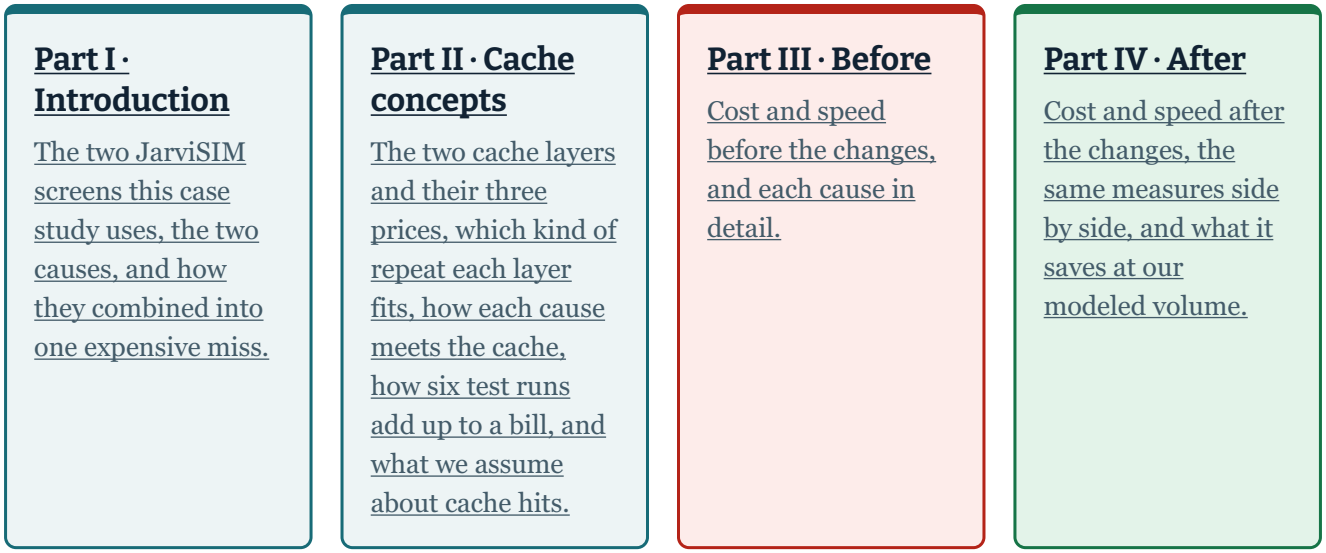
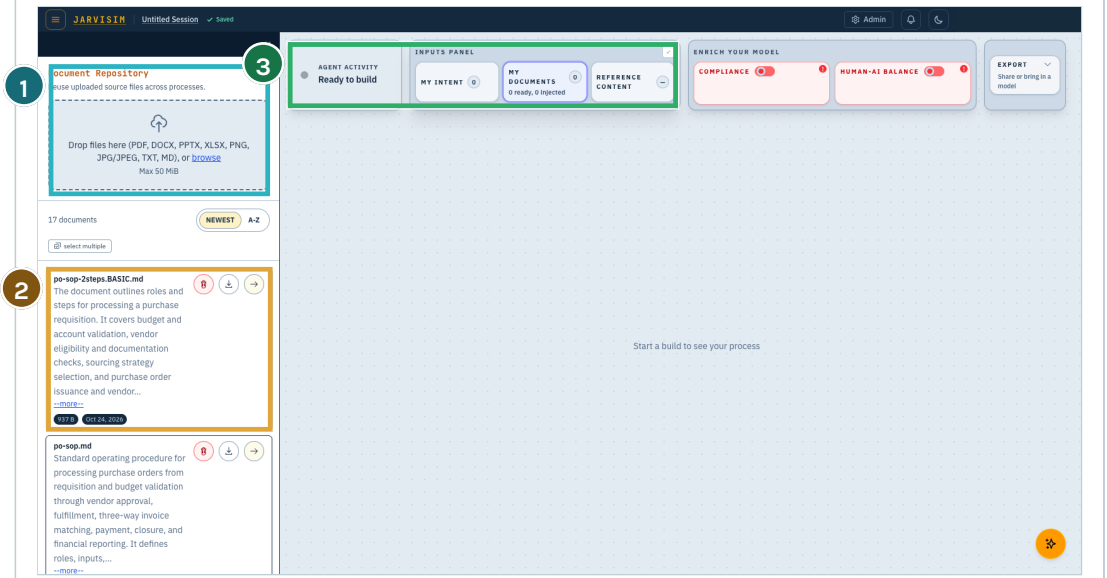



**Diagram 1.** The four Parts of this article (boxes, not steps).

## The two JarviSIM screens in this case study

**The Document Repository: where a document is reused.** The library keeps uploaded source files so they can be reused across processes. The user picks the same purchasing procedure they built from before and sends it to a BPMN model build, with no new upload. Same document, same intent: the AI work should be the same as last time.




**Reuse from the library.** Current build.


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

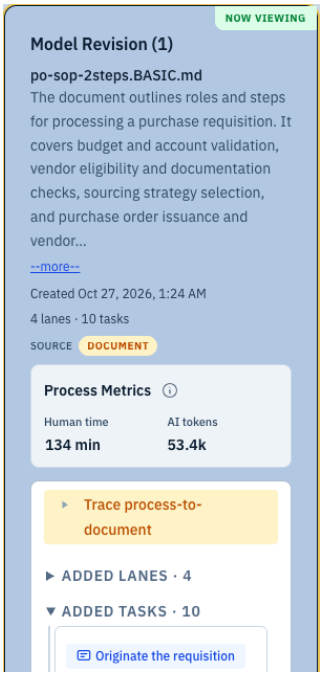
1 **The library exists so documents are reused**
Intent · Reuse is the normal path
Reusing a document is what the library is for. That is why a full bill on every reuse was a surprise.

2 **The same document, reused**
Cost · The 937-byte document the measured sessions used
The arrow sends this document to the BPMN model build with no new upload.

3 **Ready to build, nothing sent yet**
Starting point
Agent activity reads Ready to build and the inputs panel shows nothing injected. This is the moment before the reused document arrives.

Nothing on this screen shows Cause1/Cache. The reuse looks free; only the AI bill showed otherwise.

**The revision card: where the Human-AI estimate appears.** After a BPMN model build, the revision card shows Process Metrics: the human time and AI tokens the modeled process would take. Before the change, the product filled these in automatically, inside the BPMN model build.




Before the change: Human time and AI tokens were calculated automatically, without the user asking; there was no Calculate button. This is Cause2/Estimate. Illustration made from the current screen.


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

## The two causes

- **Cause 1 (cache miss), Cause1/Cache for short: four identifiers, so the cache never matched.** The BPMN model build request carried four identifiers that changed every session. Nobody had asked for them. Each reuse therefore looked like a new request, and the cache could not answer it.
- **Cause 2 (forced Human-AI estimate), Cause2/Estimate for short: the Human-AI estimate inside every BPMN model build.** The Human-AI estimate ran inside every BPMN model build, whether or not the user wanted it, and the user waited for it before seeing the model.

They call for two different kinds of fix. **A cache hit** makes repeated work free: the request is the same as before, so a stored answer is returned, and the user sees nothing different except speed. **A UX improvement** changes when work runs: work the user does not ask for never runs, and that saves money even when the work would have been new. Cause1/Cache needs a cache fix. Cause2/Estimate needs a UX fix, because a cache can only answer work it has seen before, and a Human-AI estimate of a model the user has just edited is new work.

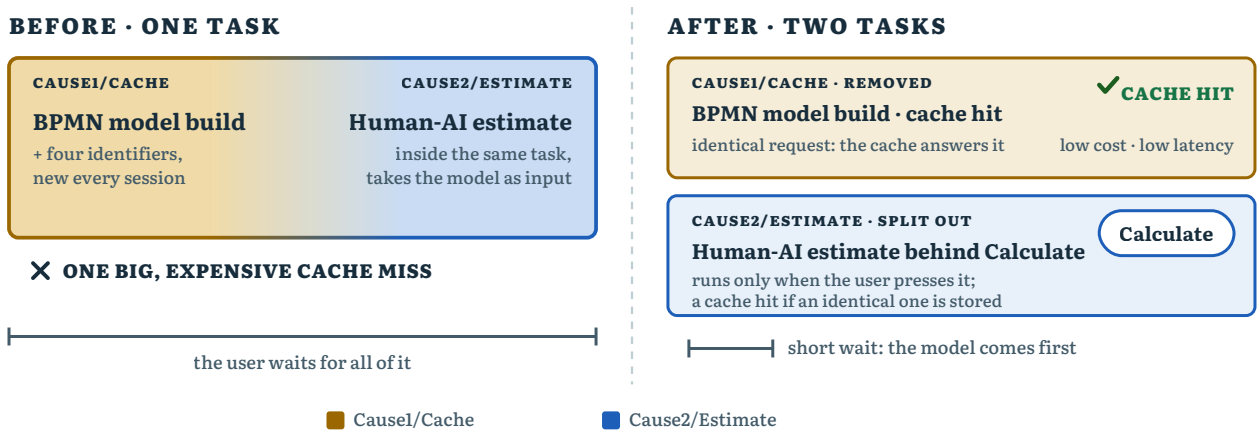



**Diagram 2.** How the two causes combined, and how we split them apart. Before: one long task, the BPMN model build with the Human-AI estimate inside it; one big, expensive cache miss, and the user waits for all of it. After: two tasks; the BPMN model build is a cache hit, and the Human-AI estimate runs only when the user presses Calculate.

In the cost tables and charts of Parts III and IV, one color marks Cause1/Cache and another marks Cause2/Estimate, so each saving can be traced back to its cause.


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

PART II

# Cache concepts.

The cache was switched on and configured correctly, yet a reused document was billed almost as if it were new. To see why, it helps to know that our AI requests pass through two caches, that each answers a different kind of repeat, and that the two causes meet them in different ways.

**What a cache does here.** Our AI requests pass through two caches before any model runs.

- Layer 1: our **AI gateway**, a proxy we run in front of the AI provider, keeps a response cache. When a request is an exact repeat of a recent one (same model, same instructions, same content), it is answered from the stored answer, with no new request to the provider and no token charge. Free means no new provider request and no token charge; running the gateway still uses some resources, but that is negligible compared to AI LLM costs.
- Layer 2: the **AI provider's own prompt cache**. When a request is new but its beginning is unchanged from a recent one, the provider reuses that beginning and bills it at a low rate, up to a 95% discount from the normal input price. The rest of the request is billed at the normal input price, and the answer at the output price.

So a request that misses at both layers runs in full at the high price. To simplify, we represent this as free, low and high.

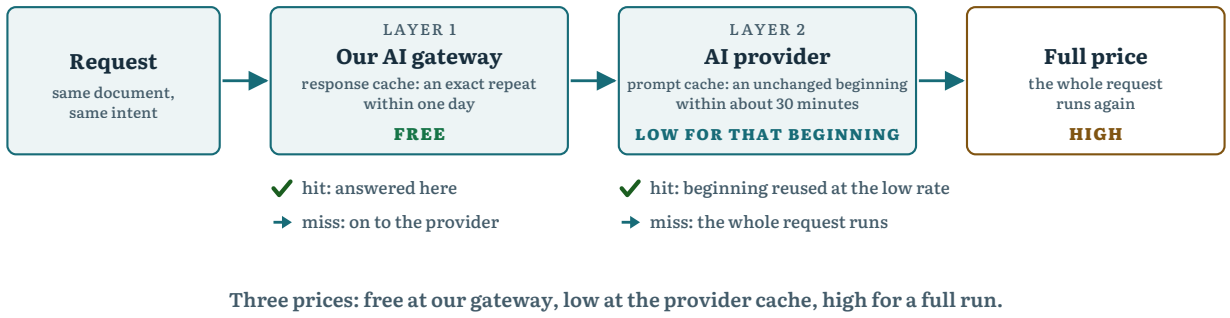



**Diagram 3.** Two cache layers, three prices: free at our gateway, low for an unchanged beginning at the provider, high for a full run.

### Which cache fits which kind of repeat

The two layers are not a better and a worse version of the same thing. Each fits a different shape of repeat, and a good design matches each AI step to the layer its repeats fit.


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

| KIND OF WORK | HOW IT REPEATS | LAYER THAT FITS | WHY |
|---|---|---|---|
| **Chatbot conversation** | Every turn resends the whole transcript plus one new message, seconds to minutes apart | Provider prompt cache | The history is an unchanged beginning, billed at the low rate; only the new message and the answer pay in full. No two turns are exact repeats, so the gateway never hits. |
| **Many steps inside one task** | The same instructions and tools are sent many times within seconds | Provider prompt cache | The shared instructions are reused every time; each step's own content is new. |
| **Same document, same intent, rebuilt later the same day (our case)** | **The exact same request, hours apart, and the right answer has not changed** | **Gateway response cache** | **The whole answer is reused for free for a day. A 30-minute window would usually expire before the user comes back.** |
| **"Latest news", today's prices, a live calendar** | The same request, but the right answer changes over time | Neither, for the answer | A stored answer would be stale. The fixed instructions can still use the provider's low rate. |
| **"Try again": the user wants a different attempt** | The same request, but the user asks for a new answer | Must skip the gateway cache | Serving the stored answer would ignore the request. That is the one place a "fresh draw" marker belongs. |
| **A one-time request** | Never repeats | Neither | Writing a provider prefix costs about 1.25 times normal input on current models and is never read back. |

**Why our case fits the gateway cache.** A BPMN model build reads a library document that does not change, for an intent that does not change, and nothing it depends on changes during the day. So the right answer for a repeat is the stored answer, and serving it is also more consistent: the same document gives the same model. That makes the gateway's free, one-day, whole-answer reuse the layer to design for. The provider's cache is the complement inside a run, where many requests share the same instructions; that is where the 1 in 8 discounted prompt tokens in Part III came from. For the gateway layer to work, the request must contain everything that should change the answer (model, instructions, document, intent) and nothing that should not, such as a session identifier. That is exactly what Cause1/Cache broke.

## How each cause meets the cache

**Cause1/Cache.** A cache answers a request only if it matches a stored one. An identifier that changes every session makes every BPMN model build request different, so it misses the free layer; and because the identifiers sat inside the request, the low layer could reuse only the part before the first one. Worse, every step after the first BPMN model build request takes its answer as input. Once that request is new, everything after it is new too, so one stray identifier makes a whole chain of requests miss.

**Cause2/Estimate.** The Human-AI estimate takes the built model as its input. Before the fix it missed because the BPMN model build missed (Cause1/Cache). With Cause1/Cache fixed, the Human-AI estimate of an unchanged model can be answered by the cache, but once the user edits the model, the Human-AI estimate is new work that no cache has seen. The cache cannot remove that cost; only not running the Human-AI estimate until the user asks for it can.


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

## How six test runs add up to a bill

Every test run reused the same document, so every run asked for the same AI work and was billed about the same. Add the six runs together and you get the billed total our gateway recorded. Divide that total among the six runs and you get the cost of one average run. Part III shows both: the billed total of six runs, and one run as the total's sixth share, so that six times one run comes back to the total. A run is not a cache hit: before the fix, every run missed the cache and paid in full, which is exactly why the six runs cost six times one.

## Not every request hits the cache

Even with the design fixed, our gateway's response cache answers only a request it has seen recently. A user can come back to the same session hours or days later and ask for another AI operation on earlier AI output; by then the stored answer may be gone, and that request pays full price again. A single miss costs cents, which is why cost is easy to ignore in a demo and hard to ignore in production: many AI demos never become economically feasible in production because they are priced per request, and nobody counted the requests. So the after state needs a volume and assumptions about how often the cache answers. We use these, each labeled:

- **Volume:** 1,000 users each reuse 10 documents 10 times a month: 1.2 million runs a year (modeled).
- **80%** of runs repeat an identical request within one day and are answered free by our AI gateway's response cache (assumed, one-day cache life). The rest miss and pay full price.
- **50%** of runs are followed by a calculate press (assumed). The other 50% have no press: the user does not need the process's Human-AI estimate on that occasion. We assume the same press rate in cached and uncached runs.
- A calculate press after a BPMN model build the cache answered is answered too, because we assume a previous identical calculation is already present in the warm cache (assumed). A press after a BPMN model build that missed pays full price.


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

PART III · BEFORE

# What one run cost, and why.

No product screen shows cache hits or dollars, so the cost figures come from our AI gateway's spend records for the six test runs. Here is every stage, grouped by cause.

## Cost and speed, before the changes

Cause1/Cache, four identifiers

Cause2/Estimate, forced Human-AI estimate

grey = not affected by either cause

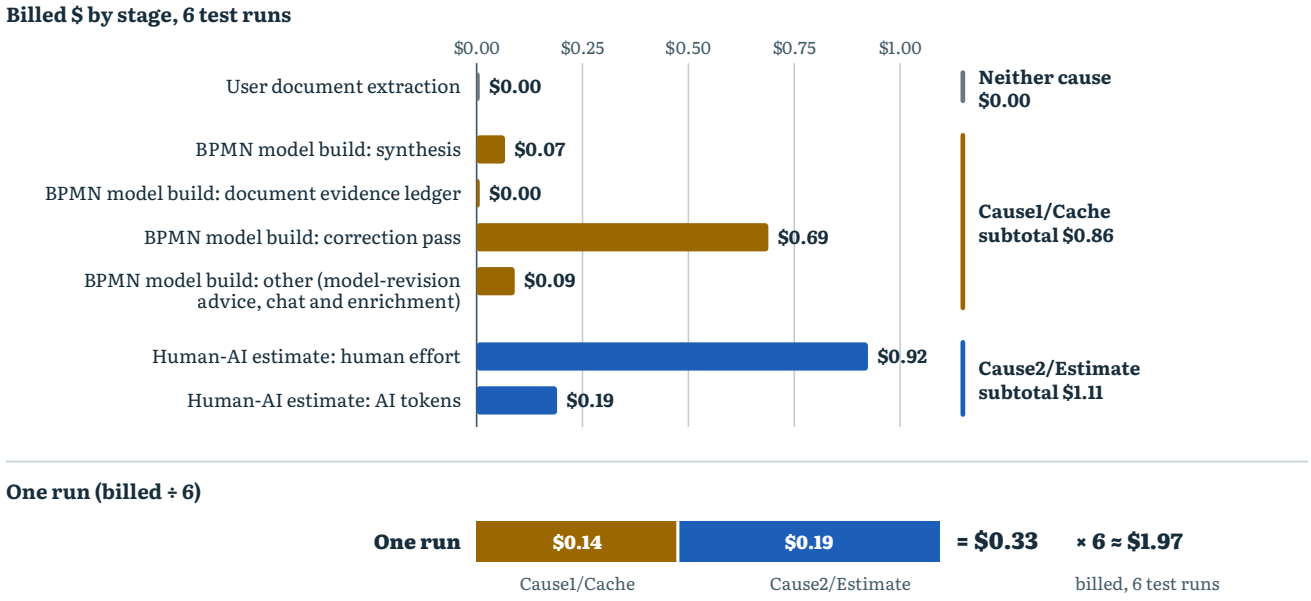
| STAGE | CAUSE | BILLED CALLS, 6 TEST RUNS | FREE-CACHE HITS (AI GATEWAY), 6 TEST RUNS | BILLED $, 6 TEST RUNS | ONE RUN (BILLED ÷ 6) |
|---|---|---|---|---|---|
| User document extraction | – | 5 | 38 | $0.00 | $0.00 |
| BPMN model build: synthesis | 1 | 11 | 0 | $0.07 | |
| BPMN model build: document evidence ledger | 1 | 6 | 0 | $0.00 | |
| BPMN model build: correction pass | 1 | 3 | 0 | $0.69 | |
| BPMN model build: other (model-revision advice, chat and enrichment) | 1 | 9 | 0 | $0.09 | |
| **Cause1/Cache: BPMN model build** | **1** | **29** | **0** | **$0.86** | **$0.14** |
| Human-AI estimate: human effort | 2 | 13 | 0 | $0.92 | |
| Human-AI estimate: AI tokens | 2 | 15 | 0 | $0.19 | |
| **Cause2/Estimate: Human-AI estimate** | **2** | **28** | **0** | **$1.11** | **$0.19** |
| **Total** | | **62** | **38** | **$1.97** | **$0.33** |

### The math, one run × 6 test runs ≈ billed

| | ONE RUN | × 6 TEST RUNS | BILLED |
|---|---|---|---|
| Cause1/Cache: BPMN model build | $0.14 | 6 × $0.14 | ≈ $0.86 |
| Cause2/Estimate: Human-AI estimate | $0.19 | 6 × $0.19 | ≈ $1.11 |
| **Total** | **$0.33** | **6 × $0.33** | **≈ $1.97** |


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

One run is rounded to the cent, so six times it lands within a few cents of the recorded billed amount (for example, 6 × $0.19 = $1.14 against $1.11 billed). Stage rows are rounded to the cent; subtotals and the total are the recorded values. The Human-AI estimate ran more than once in some runs, so $0.19 is an average per run, not the price of one Human-AI estimate.




The same chunks as the table, drawn to scale from the gateway's spend records: stage bars over the six test runs, by cause (grey for user document extraction), then one run as $0.14 + $0.19 = $0.33, and × 6 ≈ the $1.97 billed.

One average run cost **$0.33**: **$0.14** for Cause1/Cache in the BPMN model build plus **$0.19** for Cause2/Estimate.

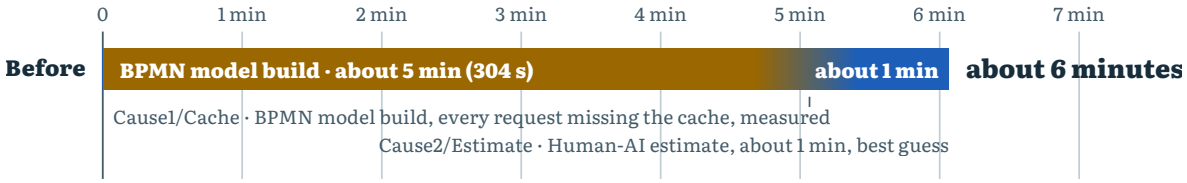
Dollars as recorded by our AI gateway for the models configured at the time: GPT-6 Luna for user document extraction and the BPMN model build, GPT-6 Sol for the Human-AI estimate and the correction pass. List price at the time, per million tokens (input / cached input / output): GPT-6 Luna $0.10 / $0.01 / $0.50; GPT-6 Sol $2.00 / $0.20 / $10.00. Users can select higher- or lower-cost models.


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

## Speed, from the user's point of view: Build pressed → finished model shown

| BEFORE THE CHANGES | WAIT |
|---|---|
| BPMN model build, every request missing the cache (Cause1/ Cache) | about 5 minutes (304 s), measured |
| Human-AI estimate inside the BPMN model build (Cause2/ Estimate) | about 1 more minute, best guess |
| **Total wait** | **about 6 minutes** |

**Wait, Build pressed → finished model shown**




The BPMN model build time is measured, one run, on the current build of the same document with the cache switched off, so that every request ran the model as it did before the fix. The Human-AI estimate was never timed inside a BPMN model build; about 1 minute is our best guess from its two parts: the AI-token half took about 13 s in one recorded call, and the human-effort half cost about five times as much per call.

Read against the cache concepts in Part II, four things stand out:

- **The free cache worked for document extraction.** The stages that read the document did not change between reuses, so they hit the free layer 38 times.
- **The BPMN model build missed the free cache, and everything after it missed too.** Cause1/ Cache: the BPMN model build request carried four identifiers that changed every session. Every later step takes the model as its input, so once the request differed, nothing after it could hit: 0 hits in every Cause1/ Cache and Cause2/Estimate row.
- **The provider's prompt cache took a little off the bill.** About 1 in 8 prompt tokens were read from the provider's cache at the low rate. They came from requests inside a run that share the same instructions, the kind of repeat that cache fits (Part II). That discount is already inside the billed dollars above; the free-cache hits column counts only our gateway's cache.
- **The largest single cost was the Human-AI estimate.** Cause2/Estimate: it ran inside every BPMN model build, whether or not the user wanted it: $0.19 of every $0.33.


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

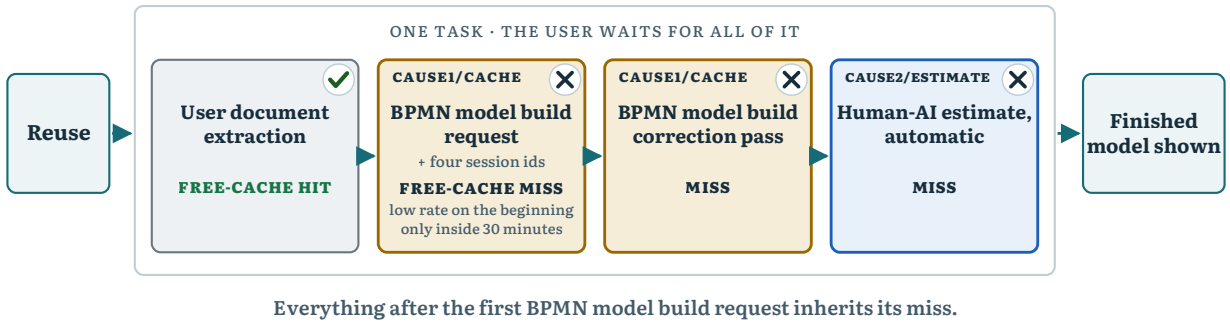



**Diagram 4.** Before: one long task. The user waits for all of it, and everything after the first BPMN model build request inherits its miss. Two boxes belong to Cause1/Cache, one to Cause2/Estimate; all three miss.

## Cause1/Cache: four identifiers nobody asked for

We compared two BPMN model build requests for the same document from two sessions, byte by byte. The instructions to the model, its tools, options and output format were identical. Every difference was an identifier that nobody had asked for:

- A per-session value mixed into the task and lane identifiers.
- The session's source-artifact identifier, copied into 24 lineage references.
- The BPMN model build job identifier, which included the session identifier.
- A one-time “fresh draw” (another model attempt) marker on the first attempt of the evidence ledger.

Mask those four, and the two requests are identical. The identifiers were accidental, so this is a defect, not a trade-off.

## Cause2/Estimate: inside every BPMN model build

The other half of the bill was a design choice. Inside the same BPMN model build, the product automatically ran the Human-AI estimate of the modeled process, and the BPMN model build waited for it before it was done. That is useful when the user wants it and expensive when they do not: it was the largest single cost, $0.19 of the $0.33 per run. This was a deliberate design choice, not unintended gold-plating. Part IV shows the UX improvement that changed it.

> ### how we found them, and how each became a defect
>
> We decrypted the two requests and compared every field. The system prompt, tools, options, schema and message count were identical. Every difference sat in the user message. Each identifier was recorded as its own cost defect under a written non-functional requirement: repeated builds reuse the cache unless there is a good reason not to. The requirement and its acceptance scenario are the worked example in Article 02. The fix removed each identifier from the request bytes rather than moving it elsewhere.


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

**`/jRightSize` and `/jFix`: our skills, methods & tools preventing and removing gold-plating and over-assurance.** Nobody asked for the four identifiers, which is what we mean by gold plating / over-assurance. Each addition felt reasonable to the human or AI agent adding it, but nobody weighed it against what the user needed.

Two jSwarm skills now do that weighing. `/jRightSize` decides how much to build. It asks of every added mechanism what outcome or failure warrants it, and what smaller alternative would do.

`/jFix` does the work to build/improve the feature. It diagnoses the problem, then checks GWT and NFR catalogs for intended product behavior. It writes a contract for the fix, makes the repair and proves it. The two work together: every `/jFix` contract carries a "necessity" section written with the `/jRightSize` method, and the contract check refuses a contract whose section is incomplete.

The Calculate button was a UX improvement delivered through `/jFix`: the acceptance scenario came first, then the change, then the judged walk in Part IV.

So now, the question “who asked for this?” is answered before the code is written, not after the bill arrives. Article 09 (coming) gives `/jRightSize` its worked example.

> ### where `/jRightSize` acts in the jSwarm cycle
>
> - **In the fix cycle.** The `/jFix` contract template has a necessity section bound to the `/jRightSize` method. The contract check runs before any reviewer sees the contract, and fails it when that section is missing or incomplete.
> - **At check-in.** The orchestrator's three-hourly review runs one necessity pass over the in-flight items. It records keep, narrow or cut, with one concrete cut or keep per flagged item.
> - **In a commander's lane.** The commander runs the method at each slice's contract point, and again whenever a change reaches its second repair or review round. It reuses the existing record rather than adding a gate.
> - **Six questions it asks of any addition:**
>   1. What wrong behavior would defeat it?
>   2. Which real caller uses the result?
>   3. Would a do-nothing version still pass?
>   4. Can the outcome be stated without naming the machinery?
>   5. Does it test the rule, or freeze today's shape?
>   6. Do its edits, false alarms and upkeep outweigh the protection?


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

PART IV · AFTER

# Identical requests, and a Human-AI estimate on request.

Two fixes, one for each cause. Cause1/Cache: with the four identifiers gone, the rebuilt request is identical, so our gateway answers it. Cause2/Estimate: the one long task is split in two, so the user gets the finished BPMN model first, and the Human-AI estimate runs only when they press Calculate.

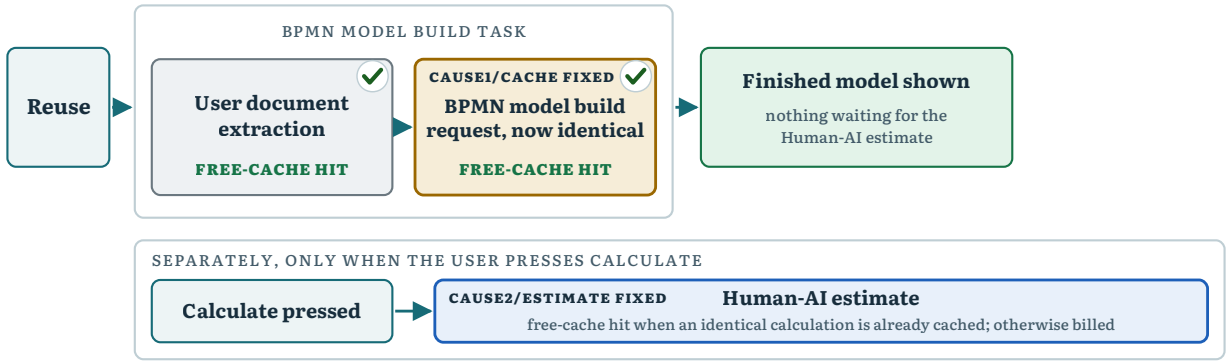
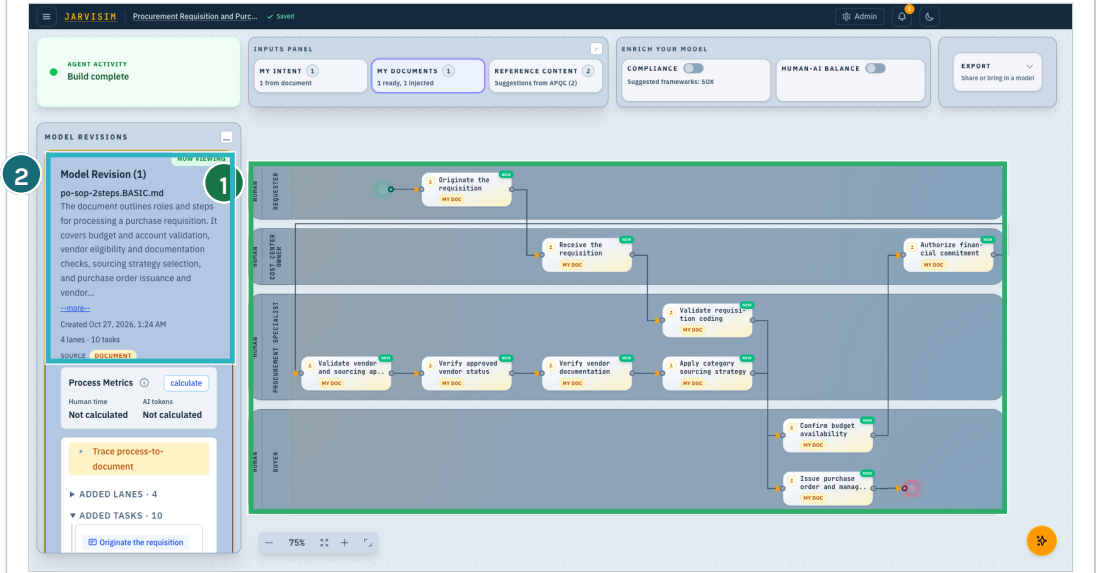



**Diagram 5.** After: two tasks. The model is shown with nothing waiting for the Human-AI estimate; the Human-AI estimate runs only when the user presses Calculate. One box is Cause1/Cache fixed, the other Cause2/Estimate fixed.

## Cause1/Cache fixed: identical requests

We removed each identifier from the request. The BPMN model build completes with four lanes and ten tasks from the one reused document.





© 2026 Lean Scaled Architects LLC | https://lsadigital.com

**Built from the one reused document.** Current build. The canvas has four lanes and ten tasks.

1 **Four lanes, ten tasks, each tagged MY DOC**
Build complete · 1 document ready, 1 injected
The BPMN model build is complete from the one reused document, and every task carries the MY DOC tag that says where it came from. Same input now means the same request, and the same request hits the cache.

2 **Same document, same intent**
Intent · Model Revision (1)
The revision card names the source document and shows 4 lanes · 10 tasks. The same document with unchanged intent is exactly the case a cache is for.

## Measured run after the fix: two rebuilds of the same document

| GATEWAY CACHE HITS | BILLED FOR THE BPMN MODEL BUILD |
|---|---|
| **26 of 26** | **$0.00** |
| Before: none | Before: $0.14 a run |

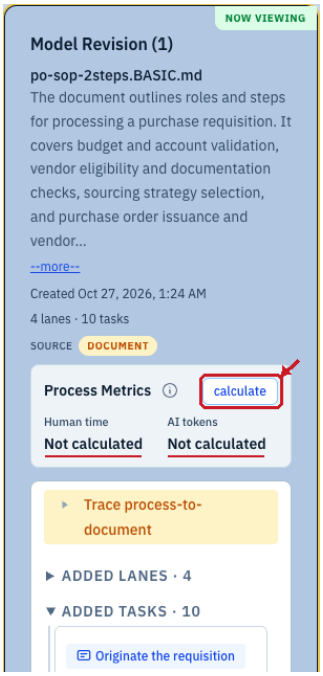
Measured by jTestEngineer in the development environment: the same document rebuilt twice, 13 requests each time, all answered at the first layer, our gateway. The requests never reached the provider, so there was nothing to bill.

This is a run the cache answers. Not every run will be: Part IV assumes the cache answers 80% of runs, and the other 20% pay the full $0.14. Averaged with the Calculate presses, that comes to **$0.05 a run**.

## Cause2/Estimate fixed: a Calculate button

Before the change, the Human-AI estimate ran inside the same BPMN model build, and the BPMN model build waited for it before it was done. The architect made a product decision, not a caching one: deliver the finished model first, show Not calculated, and let the user press Calculate when they want the Human-AI estimate. A Human-AI estimate of a model the user has changed is a new request by design; caching alone cannot remove that work.


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

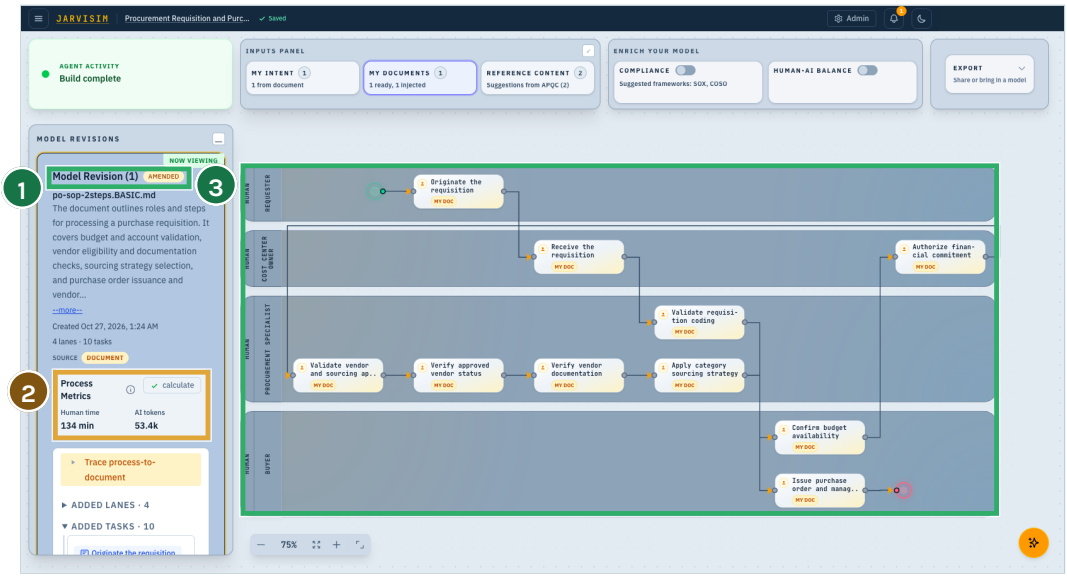



After the change: the model is finished; Human time and AI tokens remain Not calculated until the user asks. Red marks emphasize the new Calculate button and the waiting Human-AI estimate. Detail from the current screen. (Compare the before screen in Part I.)

> ***Spend tokens and latency on primary outcome first. Let users ask for the extra features.***

When the user presses calculate, the revision card is marked **AMENDED** and the action shows a check. The metrics now read **Human time 134 min · AI tokens 53.4k**, the product's displayed Human-AI estimate for the modeled process. The canvas is unchanged: nothing was rebuilt to answer the question.


© 2026 Lean Scaled Architects LLC | https://lsadigital.com




**After the press.** Current build. The user asked for the Human-AI estimate, so it ran once; the model was already finished before they pressed.

### 1 The card records that the Human-AI estimate ran

Model Revision (1) · AMENDED

The revision is amended by the Human-AI estimate, not rebuilt. The source document and the ten tasks are the same.

### 2 The Human-AI estimate, paid for once, on request

Cost · 134 min · 53.4k · calculate checked

Human time 134 min and AI tokens 53.4k are the product's Human-AI estimate for the modeled process. This is the work the user asked for, and the only new AI cost on this screen.

### 3 The canvas is unchanged

Nothing rebuilt

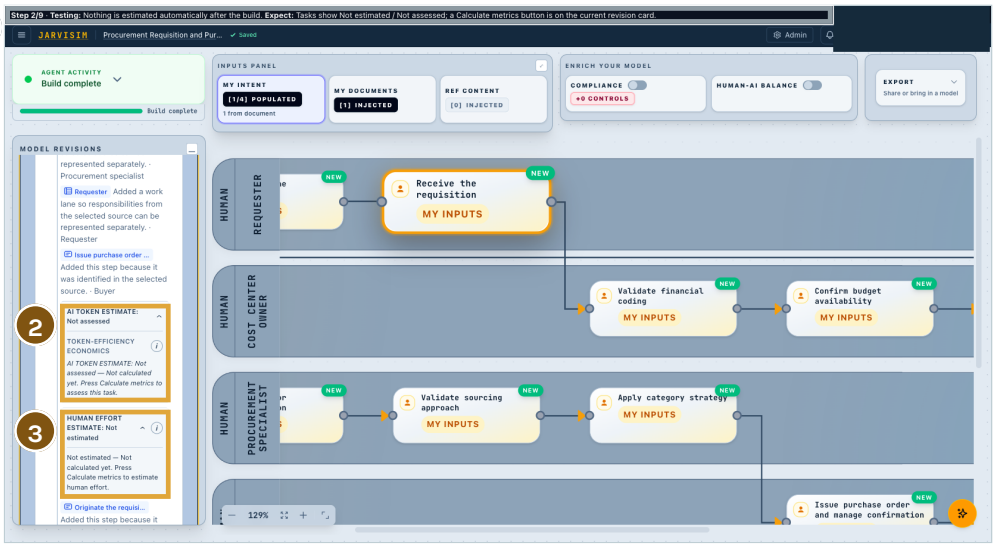
Answering the metrics question did not touch the model. The only new AI work was the Human-AI estimate the user asked for.

## how we develop and test the UX change

Why this picture: the acceptance scenario for the change was written first, in the architect's own words, and an agent walk then judged the product against it. The capture below is from that walk on an earlier build.

**The same rule at task level (an earlier build).** This capture was taken during an Agent Acceptance Test walk run by jQATester. It shows the same rule for each task: **AI token estimate: Not assessed** and **Human effort estimate: Not estimated**, each telling the user to press calculate. The dark strip at the top is the test-harness overlay that drives and judges the walk. It is not part of the product.


© 2026 Lean Scaled Architects LLC | https://lsadigital.com




**Honest placeholders (an earlier build).** Earlier layout. The top banner is the test harness, not the product.

1 **Test-harness banner (not the product)**

Earlier build · jQATester's walk controller

The banner states the step under test and what to expect: nothing is estimated automatically after the BPMN model build. It belongs to the walk controller, not to JarviSIM.

2 **AI token estimate: Not assessed**

Cost · Not calculated yet, by design

The task says Not assessed and tells the user to press Calculate metrics. A stale number here would have looked more finished and been wrong.

3 **Human effort estimate: Not estimated**

Cost · The $0.92 stage, not yet run

The same rule for human effort. Before the change, the Human-AI estimate was produced automatically inside the same BPMN model build.

The acceptance scenario for this behavior was written in the architect's own words first: the Human-AI estimate runs only when the user presses calculate, and until then the affected tasks show that metrics are not yet calculated instead of stale values. That sentence became the test. jQATester judged each step of the walk above against it.

## Cost and speed, after the changes

The same measures as Part III, side by side. Before is one measured average run; after applies the Part II assumptions to it: the cache answers 80% of runs, and the Human-AI estimate runs on 50% of runs.


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

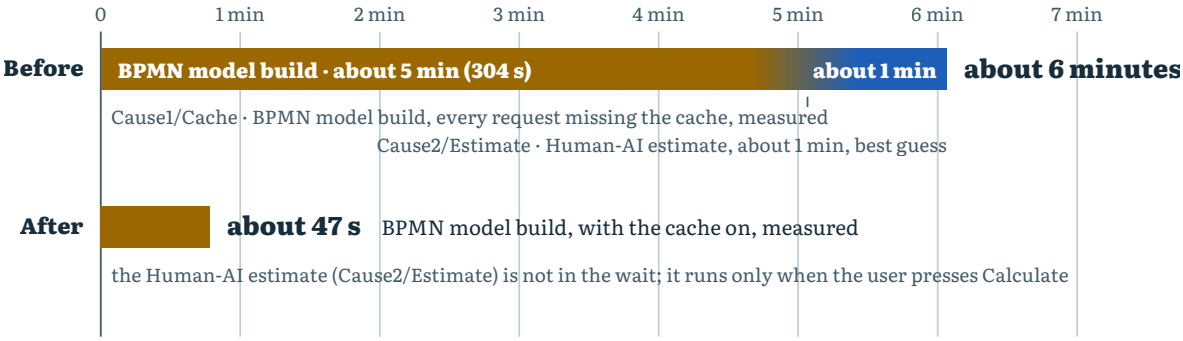
Cause1/Cache, four identifiers  Cause2/Estimate, forced Human-AI estimate
grey = not affected by either cause

| GROUP | CAUSE | FREE-CACHE HITS (AI GATEWAY), BEFORE | FREE-CACHE HITS (AI GATEWAY), AFTER | ONE RUN, BEFORE | ONE RUN, AFTER | HOW THE AFTER FIGURE IS WORKED OUT |
|---|---|---|---|---|---|---|
| User document extraction | — | ✓ hit | ✓ hit | $0.00 | **$0.00** | Unchanged. |
| BPMN model build | 1 | ✕ none | ✓ 26 of 26 measured; 80% of runs assumed | $0.14 | **$0.03** | The cache misses 20% of runs: 20% × $0.14. |
| Human-AI estimate | 2 | ✕ none | ✓ 80% of presses assumed | $0.19 | **$0.02** | Runs on 50% of runs, and the cache misses 20% of those: 50% × 20% × $0.19. |
| **Total** | | | | **$0.33** | **$0.05** | |

## Speed, from the user's point of view: Build pressed → finished model shown

| | BEFORE | AFTER |
|---|---|---|
| BPMN model build (Cause1/Cache) | about 5 minutes (304 s), every request missing the cache, measured | about 47 s, with the cache on, measured |
| Human-AI estimate (Cause2/Estimate) | about 1 more minute inside the wait, best guess | not in the wait; runs only when the user presses Calculate |
| **Total wait** | **about 6 minutes** | **about 47 s** |

**Wait, Build pressed → finished model shown**




The BPMN model build times are measured, one run each, on the current build of the same document, 13 minutes apart: first with the cache switched off, then with it on. The wait for each AI request the cache answers is about 0.2 s. The Human-AI estimate time is the same best guess as in Part III. Same scale as the chart in Part III.


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

## At our modeled volume: what it saves

The same groups, for a year at our modeled volume of 1.2 million runs. Start with what a year of runs cost before, take away each saving in turn, and see what is still billed.

| STEP | CAUSE | HOW IT IS WORKED OUT | YEARLY AI COST |
|---|---|---|---|
| **BPMN model build** | 1 | 1.2 million runs × $0.14 each. | **$168,000** |
| **Human-AI estimate, run automatically on every run** | 2 | 1.2 million × $0.19 each. | **+ $228,000** |
| **Before: total** | | $168,000 + $228,000. | **$396,000** |
| THE USER DECIDES **Human-AI estimates users don't ask for** | 2 | Half of runs have no calculate press: 50% × $228,000. | **– $114,000** |
| THE CACHE **BPMN model builds it answers** | 1 | The cache answers 80% of BPMN model builds: 80% × $168,000. | **– $134,400** |
| THE CACHE **Human-AI estimates it answers** | 2 | The cache answers 80% of the presses: 80% × ($228,000 – $114,000). | **– $91,200** |
| **After: what is still billed** | | BPMN model builds the cache misses, $33,600, plus presses it misses, $22,800. | **$56,400** |

**Total saving: $396,000 – $56,400 = $339,600 a year** (modeled).

- **Cause1/Cache, fixed by the cache: $134,400.**
- **Cause2/Estimate, fixed by the user deciding and the cache: $205,200**, the Human-AI estimates users don't ask for ($114,000) plus the presses the cache answers ($91,200).
- Check: $134,400 + $205,200 = $339,600.

### assumptions behind these figures

One-run costs: the six test runs cost $1.97 in total, $0.33 per run on average. The two Human-AI estimate stages cost $1.11 of that, about $0.19 a run; the BPMN model build and the rest cost $0.86, about $0.14 a run (derived averages from the six test runs). The Human-AI estimate ran more than once in some runs, so $0.19 is an average per run, not a measured price per press; we assume a press costs what the automatic Human-AI estimate cost on average, on GPT-6 Sol as configured at the time (list price at the time, per million tokens: $2.00 input, $0.20 cached input, $10.00 output). Volume: 1,000 users × 10 documents × 10 reuses a month × 12 months (modeled), with calculate pressed after 50% of runs (assumed).

Cache: with the four identifiers gone, the same document sent to the BPMN model build produces the same AI requests (measured: 26 of 26 hits across two rebuilds). We assume 80% of runs repeat within the gateway's one-day cache life and hit; the other 20% miss and pay full price, with no credit for the provider's prompt cache, which keeps the figures conservative. A calculate press on a rebuilt model the user has not changed sends the same request as an earlier press on that model, so a press after a BPMN model build the cache answered is assumed to hit too; a press after a BPMN model build that missed is a new request and pays full price. The same press rate applies in cached and uncached runs. The on-demand


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

saving is counted first, then the cache saving on what remains. After figures per run are rounded to the cent: $0.028 and $0.019, $0.047 in total. Users can choose a smaller model; one measured press on GPT-6 Luna cost about one cent (measured, one press).

Speed: the before and after BPMN model build waits are one measured run each of the same build of the same document. The before run had the cache switched off to stand in for the pre-fix misses; it did not include the Human-AI estimate, so the before wait adds a best-guess minute for it.

In plain terms: before the fix, every reuse of a document paid for the whole BPMN model build again and for a Human-AI estimate nobody had asked for. After the fix, most BPMN model builds cost nothing, because the cache answers them, and the Human-AI estimate is paid for only when a user presses calculate and the cache has no answer for it. The user gets the model in under a minute instead of about six, and the Human-AI estimate when they want it. Human judgment was redirected rather than removed: the architect decided what the user should pay for and when, and the user, not the pipeline, now decides when the Human-AI estimate runs.

## Optimized caching saves time and money – `/jFix` before you ship (and before you get the bill)

**The design lesson.** Article 01 argued for using the model for judgment and planning, and deterministic tools for execution. This is the cost side of the same rule. A cache is a deterministic tool, and each layer pays only for what it can reuse: our gateway's response cache needs an exact repeat of the request; the provider's prompt cache needs only an unchanged beginning. Run-unique identifiers, correlation ids and nonces stay out of prompt bytes unless a stated reason needs them there. And when a stage is expensive by nature, make it the user's choice: deliver the outcome they came for first, and put the rest behind an action. That is a product decision that saves both money and waiting.

**Catching a defective request before it costs money.** `/jFix` before you ship means running the fix cycle, diagnose, contract, repair, prove, before release rather than after the bill. For a defect in the request itself, the diagnosis starts with the request. The four identifiers were found by a byte diff of two requests after the bill arrived. Our DevOps practice now looks earlier. `/jDebug --prompt` evaluates the assembled first request, the "effective prompt" that the product actually sends, before another model attempt or a point fix. It reads the whole request in order, including the system message, tools, schema and options, and reports what in it is wrong, missing or accidentally variable, with the exact spans as evidence. It makes no model call to do so.

### what `/jDebug --prompt` checks

The check scores the assembled request on nine questions, each from 0 (clear) to 3 (a demonstrated contradiction or impossibility), and a 3 must point at the exact conflicting text: Who has authority when two instructions disagree? Can every mandatory, forbidden and schema rule be obeyed at once? Are the facts a decision needs usable where the decision is made? Does what the prompt teaches match what the consuming check accepts? Are examples kept separate from live identities? Is every required fact supplied, or owned by code? Is the model being asked for something code can derive, such as identifiers and counts? Does the request fit its allowance without truncating an obligation? Do corrections and variants keep the original obligations?


© 2026 Lean Scaled Architects LLC | https://lsadigital.com

> Two habits matter for cost. First, code derives identity, counts and coupled fields; the model is never asked to restate a fact code already knows, and long opaque identifiers are replaced by short request-local handles that code resolves afterwards. Second, every finding at the top severity becomes a test in the fixing contract, shown failing on the real pre-fix request and passing after the change, so the fix is proved on the request that was actually sent rather than on a template. The record is appended to the existing debugging receipt; no provider call, repair or retry is part of the check.

## For Claude Certified Architect–Professional readers

Anthropic's guidance on prompt caching (https://platform.claude.com/docs/en/build-with-claude/prompt-caching) explains that a cache hit needs a prompt prefix identical to one recently cached, and that varying content such as timestamps and per-request context belongs after the static prefix. We apply this on a real product. In JarviSIM the varying parts were inside the body of the BPMN model build request, so we removed them, and we let our own AI gateway answer an exact repeat before the provider is asked at all. We answered the cache question with per-stage hit and billed rows plus a byte diff of two misses, never with configuration alone. And where a stage is new work by design, we made it a product decision: run it when the user asks.

ARTICLE 03 · COST · PRODUCT RUNTIME

The cache was on, yet every reuse was billed: four accidental identifiers made each BPMN model build a new request, and the largest cost was a Human-AI estimate that ran inside every BPMN model build before anyone asked for it. We removed the identifiers and gave the user a Calculate button so the model comes first and the Human-AI estimate runs only when wanted; a measured run then rebuilt the same document twice at 26 of 26 cache hits and $0.00, a saving of about $339,600 a year at a modeled 1.2 million reuses.

HTML and PDF editions


© 2026 Lean Scaled Architects LLC | https://lsadigital.com