

✦ PLAYBOOK · COST OPTIMIZATION

AI Cost & Latency Playbook

Practical levers for making LLM features cheaper and faster without losing quality: token budgets, caching, model routing, batching, streaming and measuring cost per outcome.

FORMAT	LEVEL	READ TIME	UPDATED
Playbook	Intermediate	14 min	Apr 2026

OVERVIEW

What this playbook covers

Cost and latency in LLM systems are driven by a short list of things: how many tokens you send, how many you generate, which model processes them, how many sequential calls you make, and how much work you repeat. This playbook covers each lever, in the order we usually apply them, with the measurement you need to know whether a change actually helped.

KEY TAKEAWAYS

- Measure cost per successful outcome, not cost per call — a cheaper call that fails more often costs more.
- Output tokens and sequential round-trips dominate latency; input tokens and model choice dominate cost.
- Trim and structure prompts before reaching for a cheaper model; most prompts carry context nobody uses.
- Use provider prompt caching for stable prefixes and exact-match caching for repeated requests; treat semantic caching with care.
- Route easy requests to small models and reserve large models for what needs them — gated by evals.
- Set a latency budget per stage and stream the answer so users see progress early.

WHO IT'S FOR

Engineering leads owning AI spend

AI and backend engineers

Product managers setting SLAs

Finance partners modelling unit economics

HOW TO USE IT

Start with section 01 to instrument; you cannot optimize what you do not measure. Then work through 02–05 in order — each lever is cheaper to apply than the next.

CONTENTS

01 Measure before you optimize
02 Token budgets and prompt trimming
03 Caching at three layers
04 Model routing and batching
05 Latency budgets, streaming and parallelism
06 The cost and latency review checklist

01 / MEASUREMENT

Measure before you optimize

Every optimization in this playbook needs a before-and-after number. Instrument cost and latency per request, per stage and per outcome first.

METRIC	DEFINITION	WHY IT MATTERS
Cost per request	Input, cached and output tokens × price, plus tool and infra cost	Baseline for every change
Cost per outcome	Total cost ÷ requests that achieved the user's goal	Catches cheap-but-worse changes
Time to first token	Request start to first streamed token	What users perceive as responsiveness
End-to-end latency p50 / p95	Request start to final token, by percentile	Tail latency drives complaints and timeouts
Calls per request	Number of model and tool calls per user request	Sequential calls multiply latency

PRINCIPLE

Quality is a constraint, not a variable

Run your eval suite on every cost or latency change. An optimization that lowers quality below the agreed bar is not a saving — it moves the cost to support, rework or churn.

02 / TOKENS

Token budgets and prompt trimming

The cheapest token is the one you never send. Give every request a token budget per component and enforce it in code.

A token budget, component by component

COMPONENT	TYPICAL WASTE	FIX
System prompt	Accumulated instructions nobody removed	Review quarterly; delete rules evals show are unused
Few-shot examples	Many long examples on every call	Fewer, shorter examples; select dynamically by similarity
Retrieved context	Top-20 chunks when top-5 suffice	Rerank and cut to a fixed context budget
Conversation history	Full transcript resent every turn	Sliding window plus a running summary
Output	Verbose prose when a field is needed	Structured output, max tokens, concise instructions

01

Cap output length

Set max tokens per call to what the task needs. Output tokens are usually priced higher than input and generated sequentially.

02

Ask for structure

JSON with defined fields is shorter, easier to validate and avoids the model restating the question.

03

Compress tool results

Strip large API responses to the fields the model needs before they enter the context.

04

Keep stable content first

Order prompts as static instructions, then semi-static context, then the user turn — this also maximizes prompt-cache hits.

✦ 03 / REUSE

Caching at three layers

Repeated work is the easiest waste to remove. Different caches suit different kinds of repetition.

EXACT AND PROMPT CACHING

- Exact cache: same normalized input, same answer
- Provider prompt cache: reuses a stable prefix
- Safe and predictable
- Gains depend on how stable prompts are

SEMANTIC CACHING

- Similar question returns a stored answer
- Higher hit rate on FAQ-style traffic
- Risk of serving a subtly wrong answer
- Needs tight thresholds and scoping

CACHE	USE FOR	KEY INCLUDES
Embedding cache	Re-embedding identical text	Text hash, embedding model and version
Retrieval cache	Repeated searches over a static index	Normalized query, filters, permission scope, index version
Response cache	Deterministic tasks like classification and extraction	Prompt version, model, inputs, user or tenant scope
Provider prompt cache	Long shared system prompts and documents	Byte-identical prefix; managed by the provider

✦ 04 / MODELS

Model routing and batching

Not every request needs your largest model. Route by task and difficulty, and move work that is not interactive off the real-time path.

01 Classify task & difficulty → 02 Pick smallest adequate model → 03 Validate output → 04 Escalate on failure → 05 Log route & result

ROUTE

Small models for simple tasks

Classification, extraction, routing and short rewrites often run well on small, fast models.

ROUTE

Cascades

Try the cheap model first; escalate to a larger one only when validation or confidence checks fail.

BATCH

Batch APIs for offline work

Enrichment, backfills and nightly evals tolerate delay; provider batch endpoints are typically discounted.

✦ FIELD NOTE

Validate the route, not just the model

A router is a model too. Evaluate it on a labelled set: how often does it send hard requests to the small model, and what does that cost in quality?

05 / LATENCY

Latency budgets, streaming and parallelism

Users experience the slowest sequential path. Set a budget per stage, then remove stages from the critical path.

Example latency budget for a RAG answer (illustrative)

STAGE	BUDGET	LEVERS
Auth, routing, guardrails	≤ 100 ms	Cache identity; run input checks in parallel
Query rewrite	≤ 400 ms	Small model or skip for simple queries
Retrieval + rerank	≤ 500 ms	Parallel keyword and vector search; limit rerank candidates
Generation, first token	≤ 1 s	Shorter prompt, prompt caching, faster model
Generation, full answer	Stream throughout	Cap output length; stream to the client

01 **Stream every user-facing response**

Time to first token matters more to perceived speed than total time. Stream tokens and show progress for tool steps.

02 **Parallelize independent calls**

Issue independent tool calls, retrievals and guardrail checks concurrently instead of in sequence.

03 **Collapse sequential model calls**

Merge steps that do not need separate calls, or move them to deterministic code.

04 **Precompute where possible**

Summaries, embeddings and classifications that do not depend on the live request can be computed ahead of time.

05 **Set timeouts with fallbacks**

Each stage has a timeout and a defined degraded behavior, such as answering without reranking.

✦ 06 / OPERATIONS

The cost and latency review checklist

Run this review on each AI feature before launch and again every quarter.

MEASURE

- [ ] Tokens, cost and latency logged per call and per stage
- [ ] Cost per successful outcome tracked on a dashboard
- [ ] Eval suite run on every optimization

TOKENS

- [ ] Token budget per prompt component enforced in code
- [ ] Max output tokens set on every call
- [ ] Conversation history windowed or summarized

REUSE & ROUTING

- [ ] Prompts ordered for prefix caching
- [ ] Exact caches keyed with model and prompt version
- [ ] Router evaluated on a labelled set

LATENCY

- [ ] Per-stage latency budget agreed
- [ ] Responses streamed; independent calls run in parallel
- [ ] Alerts on p95 latency and daily spend

> **Optimize the outcome, then the call. The cheapest answer is the one that is right the first time.**
>
> SPLENDENSLABS ENGINEERING PRINCIPLE

✦ GET IN TOUCH

Let's Talk

We audit and tune AI systems for cost and latency without trading away quality. Bring us your traces, your bill and the SLA you need to hit.

- ✦ Token and cost instrumentation review
- ✦ Caching and model-routing design
- ✦ Latency budget and streaming plan

BOOK AN AI STRATEGY SESSION →

EMAIL
hello@splendenslabs.com

WEB
splendenslabs.com

OFFICE
Sona Incubation Foundation, Salem