

# LLM COST OPTIMIZER PROXY

## MVP Technical Specification

Version 0.1 — February 2026

Status: DRAFT — For Claude Code pairing session

**Language:** Go

**Architecture:** Reverse proxy with pipeline middleware

**MVP Scope:** Caching + Budget Enforcement + Fallback + Benchmarking

**Providers:** OpenAI, Anthropic, Google, DeepSeek, Groq, Together

# 1. MVP Goals

The MVP must achieve exactly four goals. Each has a measurable acceptance criterion. If any goal is not met, the proxy is not ready for use.

## GOAL 1: Negligible Proxy Overhead — < 10ms P99

The proxy must add less than 10ms of latency to any request compared to a direct API call. Measured as P99 overhead under load with 1,000 concurrent connections.

## GOAL 2: Streaming Parity

Streaming (SSE) requests must have identical time-to-first-byte (TTFB) overhead as non-streaming. The proxy acts as a transparent SSE relay — no buffering, no waiting. TTFB overhead < 10ms.

## GOAL 3: Demonstrable Token/Cost Reduction

The proxy must achieve measurable cost savings through exact-match caching, semantic caching, and output budget enforcement. Target: 20–40% cost reduction on realistic workloads with zero quality degradation on cache hits.

## GOAL 4: Intelligent Fallback

When a provider returns 429 (rate limit), 500/502/503 (server error), or times out, the proxy automatically retries with exponential backoff, then falls back to a configured secondary model/provider. Zero failed requests that could have been served.

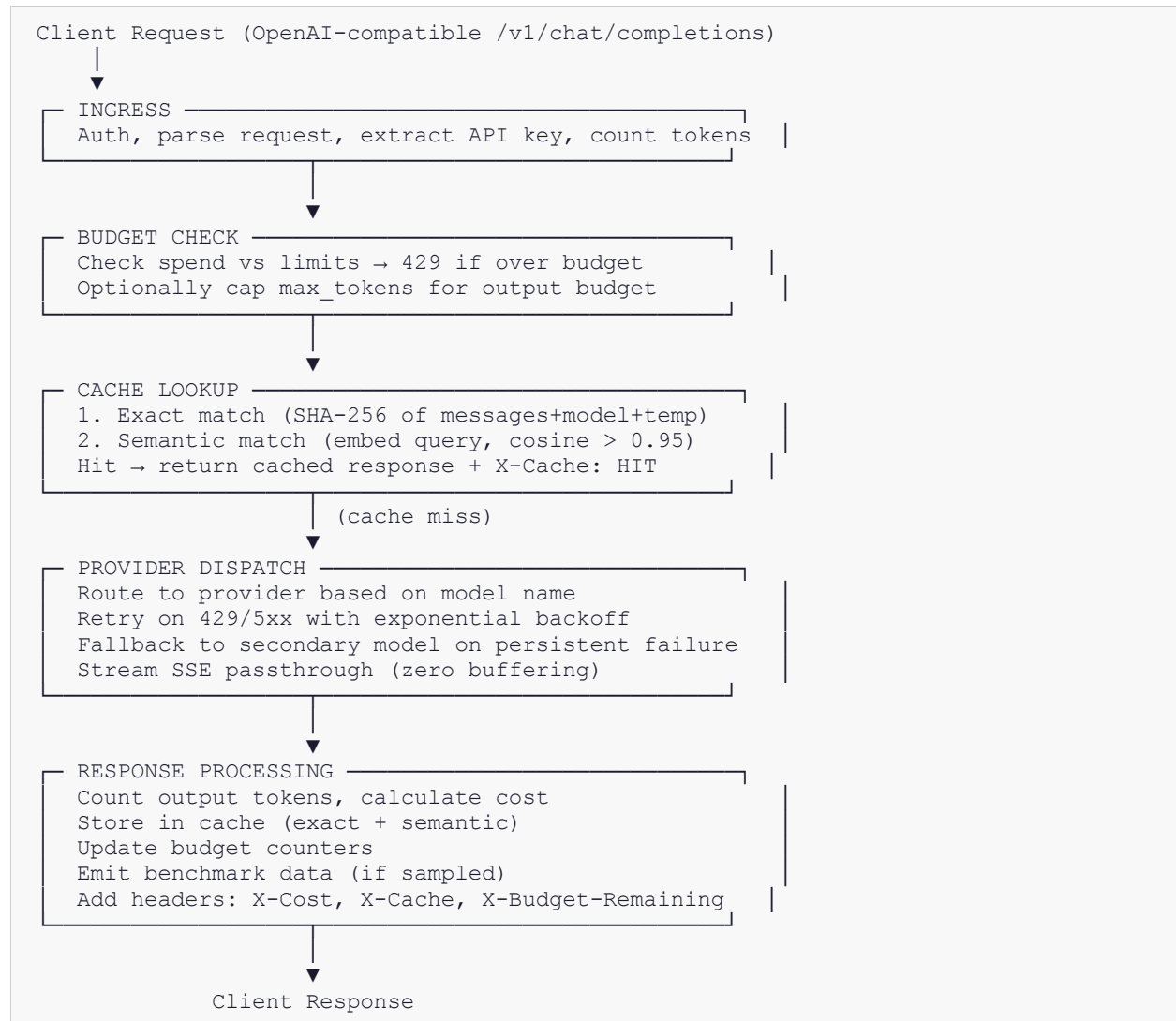
## Acceptance Criteria Matrix

| Goal               | Metric                                     | Target | How to Measure   |
|--------------------|--|--------|--|
| G1: Overhead       | P99 added latency                          | < 10ms | Benchmark: 1K requests, direct vs proxied                          |
| G2: Streaming      | TTFB overhead                              | < 10ms | Benchmark: streaming requests, measure first SSE chunk delta       |
| G3: Cost reduction | Cost savings on test suite                 | ≥ 20%  | A/B benchmark: same prompts, direct vs proxied, compare total cost |
| G3: Quality        | Semantic similarity on cache hits          | ≥ 0.95 | Embedding cosine similarity of cached vs fresh responses           |
| G4: Fallback       | Request failure rate under provider errors | 0%     | Inject 429/500 errors, verify all requests succeed via fallback    |

## 2. Architecture

### 2.1 Request Lifecycle

Every request flows through a pipeline of stages. Each stage can short-circuit (e.g., cache hit returns immediately) or modify the request before passing it forward.



### 2.2 Module Structure



|                     |  |
|---------------------|--|
| handler.go          | # /v1/chat/completions, /v1/embeddings |
| middleware.go       | # Auth, request ID, CORS, logging      |
| streaming.go        | # SSE proxy – transparent relay        |
| benchmark_api.go    | # /v1/benchmark/* endpoints            |
| pipeline/           |  |
| pipeline.go         | # Stage interface + orchestrator       |
| budget.go           | # Pre-request budget check             |
| cache.go            | # Exact + semantic cache lookup        |
| benchmark.go        | # Dual-path A/B execution              |
| provider/           |  |
| provider.go         | # Provider interface                   |
| openai_compat.go    | # OpenAI, Groq, Together, DeepSeek     |
| anthropic.go        | # Anthropic Messages API               |
| google.go           | # Gemini API                           |
| fallback.go         | # Retry + fallback chain logic         |
| cache/              |  |
| exact.go            | # Redis SHA-256 hash lookup            |
| semantic.go         | # Embedding + cosine similarity        |
| embedder.go         | # Embedding client (OpenAI API)        |
| budget/             |  |
| tracker.go          | # Atomic spend tracking per key        |
| policy.go           | # Budget rules & limits                |
| store.go            | # Redis-backed persistence             |
| benchmark/          |  |
| sampler.go          | # Probabilistic request sampling       |
| comparator.go       | # Quality metrics (similarity, tokens) |
| store.go            | # Benchmark result persistence         |
| reporter.go         | # Aggregate stats & summaries          |
| tokenizer/          |  |
| counter.go          | # tiktoken-compatible token estimation |
| pricing/            |  |
| pricing.go          | # Model price table + cost calc        |
| analytics/          |  |
| logger.go           | # Structured JSON request logs         |
| metrics.go          | # Prometheus /metrics endpoint         |
| config/             |  |
| config.yaml         | # Default configuration                |
| deploy/             |  |
| Dockerfile          | # Multi-stage Go build (~15MB)         |
| docker-compose.yaml | # Proxy + Redis + Qdrant               |

## 2.3 Pipeline Interface

Every processing stage implements a single interface. This allows adding new stages (routing, compression) without changing existing code.

```

type Stage interface {
    // Process handles a request. Return:
    //   (req, nil, nil) → pass modified request to next stage
    //   (nil, resp, nil) → short-circuit, return response to client
    //   (nil, nil, err) → error, abort pipeline
    Process(ctx context.Context, req *Request) (*Request, *Response, error)
    Name() string
}

type Pipeline struct {

```

```

    stages []Stage
}

func (p *Pipeline) Execute(ctx context.Context, req *Request) (*Response, error) {
    for _, stage := range p.stages {
        modified, resp, err := stage.Process(ctx, req)
        if err != nil { return nil, err }
        if resp != nil { return resp, nil } // short-circuit
        req = modified
    }
    return nil, errors.New("no stage produced a response")
}

```

## 3. Provider Support

Six providers, three adapter implementations. OpenAI-compatible providers (Groq, Together, DeepSeek) share a single adapter with configurable base URLs.

| Provider  | Adapter          | API Format        | Models (MVP)                        | Streaming |
|-----------|------------------|-------------------|-------------------------------------|-----------|
| OpenAI    | openai_compat.go | OpenAI native     | gpt-4o, gpt-4o-mini, gpt-4.1-nano   | SSE       |
| Groq      | openai_compat.go | OpenAI-compatible | llama-3.3-70b, mixtral-8x7b         | SSE       |
| Together  | openai_compat.go | OpenAI-compatible | meta-llama/Llama-3.3-70B            | SSE       |
| DeepSeek  | openai_compat.go | OpenAI-compatible | deepseek-chat, deepseek-reasoner    | SSE       |
| Anthropic | anthropic.go     | Messages API      | claude-sonnet-4-5, claude-haiku-4-5 | SSE       |
| Google    | google.go        | Gemini API        | gemini-2.5-flash, gemini-2.5-pro    | SSE       |

### 3.1 Provider Interface

```

type Provider interface {
    // Chat sends a non-streaming completion request
    Chat(ctx context.Context, req ChatRequest) (*ChatResponse, error)

    // ChatStream sends a streaming request, writing SSE chunks to the writer
    ChatStream(ctx context.Context, req ChatRequest, w SSEWriter) error

    // Name returns the provider identifier
    Name() string
}

```

```
// Models returns the list of supported model IDs
Models() []string
}
```

## 3.2 Fallback Chain

When a provider fails, the proxy walks a configured fallback chain. Each step gets retry attempts before moving to the next fallback.

```
Fallback logic (per request):

1. Call primary provider
2. If 429 → wait (Retry-After header or exponential backoff), retry up to N times
3. If 500/502/503/timeout → retry once with backoff
4. If still failing → check fallback chain config for this model
5. Call fallback[0] provider with equivalent request
6. If fallback[0] fails → try fallback[1], etc.
7. If all fallbacks exhausted → return 503 to client with diagnostic headers

Headers on fallback response:
X-Original-Model: gpt-4o
X-Fallback-Model: gpt-4o-mini
X-Fallback-Reason: primary_rate_limited
```

# 4. Caching

## 4.1 Exact-Match Cache

Hash-based lookup in Redis. Zero quality risk — only returns responses for identical requests.

| Parameter          | Value                                  | Rationale                               |
|--------------------|--|---|
| Hash algorithm     | SHA-256                                | Fast, collision-resistant               |
| Hash inputs        | model + messages + temperature + top_p | All params that affect output           |
| Excluded from hash | max_tokens, stream, user               | Don't affect content                    |
| Backend            | Redis                                  | Sub-ms lookups, built-in TTL            |
| Default TTL        | 1 hour                                 | Configurable per model or globally      |
| Storage            | Full response JSON + token counts      | Enables instant response reconstruction |

## 4.2 Semantic Cache

Catches near-duplicate queries by comparing embeddings. Returns a cached response when the query is semantically equivalent but not lexically identical.

| Parameter            | Value                           | Rationale                                |
|----------------------|---------------------------------|--|
| Embedding model      | text-embedding-3-small (OpenAI) | 1536 dims, \$0.02/1M tokens, fast        |
| Similarity backend   | Qdrant (or Redis Vector Search) | Purpose-built for similarity search      |
| Similarity threshold | 0.95 (configurable)             | High threshold = low false-positive risk |
| Scope                | User query + system prompt hash | Same system context required for hit     |
| Default TTL          | 1 hour                          | Same as exact cache                      |
| Max entries          | 100K vectors                    | Configurable, auto-eviction by TTL       |

### Cache Safety

Semantic cache is opt-in per deployment. The similarity threshold of 0.95 is deliberately conservative. The proxy adds X-Cache-Similarity headers so users can audit cache hit quality. Temperature > 0 requests skip semantic cache by default.

## 5. Budget Enforcement

Two enforcement points: pre-request (reject if over budget) and post-request (update counters). Budget tracking is per API key with optional team/project grouping.

### 5.1 Budget Dimensions

| Dimension                     | Enforcement                     | Default            |
|-------------------------------|---------------------------------|--------------------|
| Daily spend per key           | Hard limit → 429 when exceeded  | \$10.00            |
| Monthly spend per key         | Hard limit → 429 when exceeded  | \$200.00           |
| Max output tokens per request | Overrides max_tokens in request | null (no override) |
| Warning threshold             | X-Budget-Warning header at 80%  | 80% of limit       |

### 5.2 Cost Calculation

The proxy maintains a pricing table for all supported models. Cost is calculated on every request (including cache hits, which are recorded at \$0) and persisted atomically in Redis.

```
cost = (input_tokens * model.input_price / 1,000,000)
```

```

+ (output_tokens * model.output_price / 1,000,000)

// Cache hits: cost = 0, but still logged for analytics
// Fallback: cost uses the model that actually served the request

```

## 5.3 Response Headers

Every response includes cost and budget information via custom headers:

| Header               | Example                   | Description                   |
|----------------------|---------------------------|-------------------------------|
| X-Request-Cost       | \$0.00234                 | Cost of this specific request |
| X-Budget-Daily-Used  | \$4.23                    | Spend so far today            |
| X-Budget-Daily-Limit | \$10.00                   | Daily budget limit            |
| X-Budget-Remaining   | \$5.77                    | Remaining daily budget        |
| X-Budget-Warning     | approaching_limit         | Present when > 80% used       |
| X-Tokens-Input       | 1523                      | Input tokens counted          |
| X-Tokens-Output      | 487                       | Output tokens counted         |
| X-Cache              | HIT / MISS / SEMANTIC_HIT | Cache status                  |
| X-Tokens-Saved       | 2010                      | Tokens saved (cache hits)     |

## 6. Built-in Benchmarking & A/B Testing

Benchmarking is a first-class feature, not an afterthought. The proxy must continuously prove its value by comparing optimized responses against direct API calls.

### 6.1 Three Modes

| Mode   | Description   | Cost Impact                   | Use Case                         |
|--------|---|-------------------------------|----------------------------------|
| Off    | No benchmarking, pure proxy                               | None                          | Production after validation      |
| Sample | 1-in-N requests get dual-pathed (direct + optimized)      | ~1% extra API cost at 1% rate | Continuous production monitoring |
| Shadow | 100% of traffic dual-pathed, client always gets optimized | 2x API cost                   | Initial validation, short-term   |

## 6.2 Benchmark Execution

```

if sampler.ShouldBenchmark(req) {
    // Path A: Direct to provider (bypass cache, budget, all optimization)
    resultA := provider.DirectCall(req)

    // Path B: Full pipeline (cache, budget, etc.)
    resultB := pipeline.Execute(req)

    // Compare (async — does not block response)
    go comparator.Compare(resultA, resultB)

    // Client always gets the optimized response
    return resultB
}
return pipeline.Execute(req) // normal path

```

## 6.3 Metrics Collected Per Sample

| Metric              | Path A (Direct) | Path B (Optimized) | Comparison                       |
|---------------------|-----------------|--------------------|----------------------------------|
| Model used          | ✓               | ✓                  | Same or different (fallback)     |
| Input tokens        | ✓               | ✓                  | Difference = tokens saved        |
| Output tokens       | ✓               | ✓                  | Difference = output savings      |
| Latency (ms)        | ✓               | ✓                  | Overhead = B - A                 |
| TTFB (ms)           | ✓               | ✓                  | Streaming overhead               |
| Cost (\$)           | ✓               | ✓                  | Savings = A - B                  |
| Response hash       | ✓               | ✓                  | Exact match detection            |
| Cache hit           | —               | ✓                  | Was this a cache hit?            |
| Semantic similarity | —               | —                  | Computed post-hoc via embeddings |

## 6.4 Benchmark API

```

GET /v1/benchmark/summary
?period=24h|7d|30d
Response: {
    total_samples, avg_cost_savings_pct, avg_token_savings_pct,
    avg_latency_overhead_ms, avg_semantic_similarity,
    cache_hit_rate, fallback_trigger_rate
}

GET /v1/benchmark/results
?limit=100&offset=0
Response: [ { request_id, timestamp, path_a: {...}, path_b: {...},

```

```
savings: {...}, quality: {...} } ]

GET /v1/benchmark/compare
?technique=cache|budget|fallback
Response: per-technique effectiveness breakdown
```

## 7. Configuration

Single YAML file + environment variable overrides. No database required for config — Redis is the only external dependency.

```
server:
  port: 8080
  read_timeout: 60s
  write_timeout: 120s

providers:
  openai:
    api_key: ${OPENAI_API_KEY}
    models: ["gpt-4o", "gpt-4o-mini", "gpt-4.1-nano"]
  anthropic:
    api_key: ${ANTHROPIC_API_KEY}
    models: ["claude-sonnet-4-5", "claude-haiku-4-5"]
  google:
    api_key: ${GOOGLE_API_KEY}
    models: ["gemini-2.5-flash", "gemini-2.5-pro"]
  deepseek:
    api_key: ${DEEPSEEK_API_KEY}
    base_url: "https://api.deepseek.com/v1"
  groq:
    api_key: ${GROQ_API_KEY}
    base_url: "https://api.groq.com/openai/v1"
  together:
    api_key: ${TOGETHER_API_KEY}
    base_url: "https://api.together.xyz/v1"

cache:
  exact:
    enabled: true
    redis_url: "redis://localhost:6379/0"
    ttl: 1h
  semantic:
    enabled: true
    similarity_threshold: 0.95
    embedding_provider: openai # uses OpenAI embedding API
    qdrant_url: "http://localhost:6333"
    ttl: 1h

budget:
  enabled: true
  default_daily: 10.00
  default_monthly: 200.00
  warning_threshold: 0.80
  policies:
```

```

- key_pattern: "sk-team-*"
  daily: 50.00
- key_pattern: "sk-dev-*"
  daily: 5.00
  max_output_tokens: 500

fallback:
  retry_count: 2
  retry_backoff: "1s"
  chains:
    - primary: "gpt-4o"
      fallback: ["gpt-4o-mini", "gemini-2.5-flash"]
    - primary: "claude-sonnet-4-5"
      fallback: ["gpt-4o", "gemini-2.5-pro"]

benchmark:
  enabled: true
  mode: "sample"           # off | sample | shadow
  sample_rate: 0.01        # 1% of requests
  compare_quality: true
  retention: "30d"

```

## 8. Deployment

Single docker-compose up. Three containers, no external dependencies beyond the providers themselves.

```

# docker-compose.yaml
services:
  proxy:
    build: .
    ports: ["8080:8080"]
    environment:
      - OPENAI_API_KEY
      - ANTHROPIC_API_KEY
      - GOOGLE_API_KEY
      - DEEPSEEK_API_KEY
      - GROQ_API_KEY
      - TOGETHER_API_KEY
    depends_on: [redis, qdrant]

  redis:
    image: redis:7-alpine
    ports: ["6379:6379"]

  qdrant:
    image: qdrant/qdrant:latest
    ports: ["6333:6333"]

```

**Binary size:** ~15MB (Go static build). **Memory:** ~30MB baseline. **CPU:** Minimal — mostly waiting on provider I/O.

## 9. Explicitly Out of Scope (V1 / V2)

These are designed for but not built in MVP. The pipeline architecture supports adding them as new Stage implementations.

| Feature                                | Phase | Why Deferred  |
|--|-------|---|
| Intelligent model routing              | V1    | Requires complexity classifier validation via A/B testing first           |
| Prompt compression (Go heuristics)     | V1    | Add after proving cache + budget value                                    |
| LLMLingua-2 sidecar (deep compression) | V2    | Python dependency, needs quality validation                               |
| Conversation history summarization     | V2    | Medium quality risk, needs careful A/B testing                            |
| Web dashboard                          | V2    | Benchmark API provides data; UI is nice-to-have                           |
| Multi-tenant auth (teams, projects)    | V2    | API key-level budget is sufficient for MVP                                |
| Provider-side prompt caching           | V1    | Easy to add but provider-specific (Anthropic <code>cache_control</code> ) |

## 10. Build Plan

Ordered by dependency. Each milestone is independently testable and demoable.

### Milestone 1: Transparent Proxy (Days 1–2)

Goal: Proxy that forwards requests to all 6 providers with < 10ms overhead.

- HTTP server with `/v1/chat/completions` endpoint
- Provider abstraction + 3 adapters (OpenAI-compatible, Anthropic, Google)
- Streaming SSE passthrough
- Token counting + cost calculation
- Benchmark harness: direct vs proxied latency comparison

#### Gate

Must pass G1 (< 10ms overhead) and G2 (streaming parity) before proceeding.

**Milestone 2: Caching (Days 3–4)**

Goal: Exact and semantic caching with measurable cost savings.

- Redis exact-match cache (SHA-256 hash)
- Semantic cache via embeddings + Qdrant
- Cache response headers (X-Cache, X-Tokens-Saved)
- Cache bypass for temperature > 0 (configurable)
- Benchmark: measure cache hit rate and cost savings on repeated workloads

**Milestone 3: Budget Enforcement (Day 5)**

Goal: Per-key spend tracking with hard limits and output token caps.

- Redis-backed atomic spend counters
- Pre-request budget check (429 on exceed)
- Output token budget (override max\_tokens)
- Budget response headers (X-Budget-Remaining, etc.)
- Benchmark: verify budget accuracy over 1K requests

**Milestone 4: Fallback (Day 6)**

Goal: Zero request failures when providers have issues.

- Retry with exponential backoff (respects Retry-After)
- Configurable fallback chains per model
- Fallback headers (X-Fallback-Model, X-Fallback-Reason)
- Benchmark: inject synthetic 429/500 errors, verify 100% success rate

**Milestone 5: A/B Benchmarking System (Days 7–8)**

Goal: Continuous proof-of-value with production traffic.

- Sampler (configurable rate) + shadow mode
- Dual-path execution (direct + optimized)
- Semantic similarity comparison via embeddings
- Benchmark result storage + /v1/benchmark/\* API
- Summary report generation (savings, quality, overhead)

**Gate**

Full A/B benchmark suite must demonstrate  $\geq 20\%$  cost savings (G3) and 0% failure rate under provider errors (G4) before declaring MVP complete.

## 11. MVP Success Criteria

The MVP is complete when all four gates pass on a standardized test suite of 50+ diverse prompts across all 6 providers:

| #   | Criterion                                     | Target | Status |
|-----|---|--------|--------|
| G1  | P99 latency overhead (non-streaming)          | < 10ms |        |
| G2  | P99 TTFB overhead (streaming)                 | < 10ms |        |
| G3a | Cost savings on test suite                    | ≥ 20%  |        |
| G3b | Semantic similarity on cache hits             | ≥ 0.95 |        |
| G4  | Success rate under injected provider failures | 100%   |        |
| B1  | Benchmark API returns accurate summary stats  | Pass   |        |
| B2  | A/B comparison shows savings with evidence    | Pass   |        |

*End of Specification*