# Administrator Guide

Administration and maintenance of Open Clinical History.

- Configuration
- SNOMED Re-index / Learning Queue
- Gemini / LLM Usage Dashboard

# Configuration

`admin_config.php` provides the central runtime configuration interface for Open Clinical History.

Application settings are stored in the `app_config` database table and accessed through `lib/app_config.php`. These settings are separate from deployment-level configuration such as database connection details.

The page also manages API bearer tokens through the `api_token` table.

# Configuration architecture

Runtime configuration follows this general flow:

```
Admin → Configuration
        |
        v
admin_config.php
        |
        v
AppConfig::set(...)
        |
        v
app_config database table
        |
        +------------------------+
        |                        |
        v                        v
AppConfig::get()          AppConfig::llmLimits()
        |                        |
        v                        v
Application code          Clinical processing pipeline
```

The `app_config` table is the source of truth for application runtime settings.

Values are stored with:

- configuration key

- configuration value
- value type
- user that last changed the value
- update timestamp

Changes made through the Configuration page take effect without modifying PHP configuration files.

---

# Dates

# Date format

**Configuration key**

```
date_format
```

**Default**

```
dd/mm/yyyy
```

**Allowed values**

| Value | Example |
| --- | --- |
| `dd/mm/yyyy` | `25/08/2026` |
| `mm/dd/yyyy` | `08/25/2026` |

Controls how dates are displayed throughout the user interface.

It also determines how non-ISO dates submitted through the API are interpreted.

ISO dates using:

```
YYYY-MM-DD
```

are always accepted regardless of this setting.

Machine integrations should therefore use ISO dates wherever possible.

# Used by

- UI date rendering
- API date parsing
- API date responses
- token expiry display
- clinical history date presentation

---

# SNOMED CT RF2

# RF2 release path

**Configuration key**

```
snomed_release_path
```

**Default**

Blank.

**Example**

```
/var/www/private/snomed/SnomedCT_Release_AU1000036_20260731
```

Specifies the server location of the SNOMED CT RF2 distribution used by the SNOMED import process.

The value may point to:

- the root of an RF2 release; or
- the release's `Snapshot` directory.

The path must be an **absolute server path** beginning with `/`.

Trailing `/` characters are removed when the setting is saved.

If the setting is blank, the SNOMED subsystem falls back to its built-in private-folder discovery mechanism.

# Temporary override

`snomed_import.php` can temporarily override the configured directory using:

```
?dir=/full/path/to/release
```

The configured value remains unchanged.

# Used by

- `snomed_import.php`
- `snomed_job.php`

---

# Gemini / LLM

# Enable LLM processing

**Configuration key**

```
llm_enabled
```

**Default**

Enabled.

Controls whether Open Clinical History is allowed to perform LLM processing.

When disabled, `LlmFactory` refuses to create an LLM provider and model-dependent extraction fails with a configuration error.

Disabling this setting therefore disables functionality including model-assisted clinical extraction, terminology processing and other AI-dependent pipeline stages.

---

# LLM provider

**Configuration key**

```
llm_provider
```

**Current value**

```
gemini
```

The current administration page does not provide a provider selector.

Whenever configuration is saved, `admin_config.php` explicitly sets:

```
llm_provider = gemini
```

The underlying LLM factory is designed to support provider selection, but the current implementation only recognises Gemini.

---

# Gemini API key

**Configuration key**

```
llm_gemini_api_key
```

**Default**

Blank.

Stores the Gemini API credential used by the LLM provider.

For security, the existing key is never rendered back into the configuration page.

If the API key field is left blank when saving the page, the existing key is retained.

The **Clear the stored Gemini API key** option explicitly replaces the stored value with an empty string.

# Security

Unlike Open Clinical History API bearer tokens, the Gemini API key must remain retrievable by the application and is therefore stored as a configuration value in `app_config`.

Database access to `app_config` should consequently be treated as secret-bearing access.

---

# Gemini model

**Configuration key**

```
llm_gemini_model
```

**Default**

```
gemini-3.5-flash-lite
```

Determines which Gemini model is used for LLM operations.

The configured value may contain:

```
A-Z
a-z
0-9
.
_
-
```

The configuration page validates the syntax of the model name but does not verify with Gemini that the model actually exists.

---

# Provider limits

# Gemini timeout

**Configuration key**

```
llm_gemini_timeout_seconds
```

**Default**

```
120 seconds
```

**Allowed range**

```
10 to 600 seconds
```

Sets the maximum HTTP request time allowed for a Gemini request.

A request exceeding this period is treated as a model/API failure.

---

# Daily request cap

**Configuration key**

```
llm_daily_request_cap
```

**Default**

```
4000
```

Controls the maximum number of Gemini requests permitted per day.

A value of:

```
0
```

means unlimited.

Before making a model request, the LLM budget manager checks current daily usage. Once the configured request count has been reached, further calls are rejected before Gemini is contacted.

This provides a hard cost and usage protection mechanism.

---

# Daily token cap

**Configuration key**

```
llm_daily_token_cap
```

**Default**

```
4,000,000
```

Controls the total number of recorded prompt and output tokens permitted per day.

A value of:

```
0
```

means unlimited.

The budget check uses:

```
prompt tokens + output tokens
```

from recorded daily usage.

Once the cap is reached, additional LLM requests are blocked.

---

# Maximum output tokens per request

**Configuration key**

```
llm_max_output_tokens
```

**Default**

```
4096
```

**Configuration range**

```
256 to 131072
```

Sets the requested maximum response size sent to the Gemini provider.

Larger values allow the model to produce larger structured responses but can increase:

- latency
- token consumption
- cost
- JSON processing requirements

The actual maximum may also be constrained by the capabilities of the selected Gemini model.

---

# Maximum prompt characters per request

**Configuration key**

```
llm_max_chars_per_request
```

**Default**

```
24000
```

**Allowed range**

```
8000 to 1,000,000 characters
```

Provides a hard application-level limit on the total size of the system and user prompt sent in a single LLM request.

The budget manager calculates:

```
strlen(system prompt) + strlen(user prompt)
```

before contacting Gemini.

Requests exceeding the configured limit are rejected locally.

This setting also influences the safe batch size used by SNOMED grounding and repair.

---

# Document extraction

## Maximum characters per legacy chunk

**Configuration key**

```
llm_max_chars_per_chunk
```

**Default**

```
4500
```

**Allowed range**

```
500 to 100,000
```

Controls the target maximum size of chunks created by the legacy document chunking pipeline.

Large source documents are divided into sections before further processing.

Increasing this value creates fewer, larger chunks.

Decreasing it creates more, smaller chunks.

---

# Maximum chunks per document

**Configuration key**

```
llm_max_chunks_per_document
```

**Default**

```
40
```

**Allowed range**

```
1 to 500
```

Limits the number of legacy chunks generated for a document.

If a document would generate more chunks than this limit, the excess chunks are truncated and are not processed.

This is therefore a **processing completeness limit**, not merely a performance setting.

> A very low value can result in part of a long clinical record not being processed.

---

# Maximum characters per source segment

**Configuration key**

```
llm_max_chars_per_source_segment
```

**Default**

```
2800
```

**Admin UI range**

```
500 to 100,000
```

**Effective runtime range**

```
1200 to 5000
```

Controls the size of source segments produced by the current segmentation pipeline.

Oversized source sections are divided into smaller pieces before extraction.

The segmentation service currently clamps the value internally:

```
minimum 1200
maximum 5000
```

Consequently, setting a value such as `10000` in the administration page does **not** produce 10,000-character source segments. The runtime uses 5,000.

---

# Maximum characters per extraction batch

**Configuration key**

```
llm_max_chars_per_extraction_batch
```

**Default**

```
6500
```

**Admin UI range**

```
1000 to 1,000,000
```

**Effective minimum**

```
1500
```

Controls how much source text can be grouped into one clinical extraction request.

Extraction batches are built until either:

- the character limit is reached; or
- the segment-count limit is reached.

Whichever limit is reached first closes the batch.

---

# Maximum segments per extraction batch

**Configuration key**

```
llm_max_segments_per_extraction_batch
```

**Default**

```
6
```

**Admin UI range**

```
1 to 100
```

**Effective runtime range**

```
1 to 10
```

Controls how many source segments may be sent to the extraction model in one request.

The extraction service currently imposes a hard maximum of 10 regardless of the higher value allowed by the administration page.

---

# Retry behaviour

# Transient retries

**Configuration key**

```
llm_transient_retries
```

**Default**

```
1
```

**Admin UI range**

```
0 to 10
```

**Effective runtime range**

```
0 to 3
```

Determines how many times a transient Gemini/API error can be retried.

Only errors classified as transient are retried.

A value of `0` disables application-level retries.

The current pipeline caps retries at three even if a higher value is configured.

---

# Retry delay

**Configuration key**

```
llm_transient_retry_delay_seconds
```

**Default**

```
4 seconds
```

**Admin UI range**

```
0 to 300 seconds
```

**Effective runtime range**

```
1 to 30 seconds
```

Sets the base delay between transient retries.

The retry code applies an increasing delay based on the retry attempt:

```
delay × attempt number
```

With the default value of four seconds:

```
Retry 1: 4 seconds
Retry 2: 8 seconds
Retry 3: 12 seconds
```

Although the UI accepts zero, the runtime enforces a minimum of one second.

---

# Advanced SNOMED grounding and repair

These settings control the terminology-resolution pipeline after clinical events have been extracted.

The normal flow is conceptually:

```
Extract clinical event
        |
        v
Generate local SNOMED candidates
        |
        v
Grounded LLM resolution
        |
```

```
        +---- confident result ----> accept
        |
        +---- unresolved/suspicious
                    |
                    v
                critic
                    |
                    v
             optional repair
                    |
                    v
             re-run grounding
```

---

# Enable AI-assisted SNOMED repair

**Configuration key**

```
llm_enable_snomed_ai_repair
```

**Default**

Disabled.

Allows the pipeline to use model-assisted repair when normal terminology grounding cannot confidently resolve an event.

When disabled, unresolved mappings do not enter the AI repair pathway.

This is separate from normal grounded terminology resolution.

---

# Grounded batch characters

**Configuration key**

```
llm_grounded_batch_chars
```

**Default**

```
18000
```

**Admin UI range**

```
1000 to 1,000,000
```

**Effective minimum**

```
6000
```

Controls the target maximum size of a grounded SNOMED resolution batch.

The actual batch budget is also constrained by `llm_max_chars_per_request`.

The pipeline reserves approximately 4,500 characters for prompt overhead:

```
effective budget =
min(
    grounded_batch_chars,
    max(6000, max_chars_per_request - 4500)
)
```

With the defaults:

```
max_chars_per_request = 24000
grounded_batch_chars  = 18000

effective budget = 18000
```

---

# Candidate limit

**Configuration key**

```
llm_grounded_candidate_limit
```

**Default**

```
12
```

**Admin UI range**

```
1 to 100
```

**Effective runtime range**

```
4 to 20
```

Controls the maximum number of locally generated SNOMED candidates supplied for each event during grounded terminology resolution.

More candidates can improve recall but also:

- enlarge prompts
- increase ambiguity
- consume more tokens
- make model selection more difficult

The runtime currently limits the value to 20.

---

# Search terms per event

**Configuration key**

```
llm_grounded_search_terms_per_event
```

**Default**

```
5
```

**Admin UI range**

```
1 to 20
```

**Effective runtime range**

```
1 to 8
```

Controls how many clinical search expressions may be used to locate local SNOMED candidates for an event.

The runtime currently caps this value at eight.

---

# Resolution minimum confidence

**Configuration key**

```
llm_grounded_resolution_min_confidence
```

**Default**

```
0.72
```

**Allowed range**

```
0.00 to 1.00
```

Specifies the minimum confidence required before a grounded model-selected SNOMED candidate is accepted.

Higher values make terminology resolution more conservative.

Lower values permit weaker matches to be accepted.

For clinical terminology, this setting should be changed cautiously because it directly influences whether a proposed terminology match is accepted.

---

# SNOMED repair

# Repair rounds

**Configuration key**

```
llm_grounded_repair_rounds
```

**Default**

```
2
```

**Admin UI range**

```
0 to 10
```

**Effective runtime range**

```
0 to 3
```

Sets the number of terminology repair cycles permitted for unresolved or suspicious matches.

A value of zero disables repair rounds even if the AI repair pathway is otherwise enabled.

The runtime currently permits a maximum of three rounds.

---

# Repair candidate limit

**Configuration key**

```
llm_grounded_repair_candidate_limit
```

**Default**

```
15
```

**Admin UI range**

```
1 to 100
```

**Effective runtime range**

```
4 to 20
```

Controls how many SNOMED candidates may be considered when an event is being re-evaluated during repair.

Repair deliberately uses a separate limit from first-pass terminology grounding.

---

# Repair search terms per event

**Configuration key**

```
llm_grounded_repair_search_terms_per_event
```

**Default**

```
6
```

**Admin UI range**

```
1 to 20
```

**Effective runtime range**

```
1 to 8
```

Controls how many revised search expressions may be used when attempting to resolve an event during a repair round.

---

# Repair maximum terms

**Configuration key**

```
llm_grounded_repair_max_terms
```

**Default**

```
6
```

**Admin UI range**

```
1 to 20
```

**Effective runtime range**

```
2 to 8
```

Limits the final collection of SNOMED search terms assembled for a repaired event.

The terms may originate from:

- repair-plan search terms
- existing SNOMED search terms
- condition text
- event title

Duplicates are normalised before the limit is applied.

---

# Repair batch characters

**Configuration key**

```
llm_grounded_repair_batch_chars
```

**Default**

```
15000
```

**Admin UI range**

```
1000 to 1,000,000
```

**Effective minimum**

```
6000
```

Controls the target size of LLM batches used to plan terminology repair.

The actual limit is constrained by the global prompt limit:

```
effective budget =
min(
    grounded_repair_batch_chars,
```

```
    max(6000, max_chars_per_request - 5000)
)
```

The reserved space accommodates system instructions and additional repair context.

---

# Grounded critic

# Enable grounded critic pass

**Configuration key**

```
llm_grounded_critic_enabled
```

**Default**

Enabled.

Enables an additional quality-control pass for grounded SNOMED resolutions.

The critic can flag a mapping for repair when:

- the selected match is below the configured confidence threshold
- the resolver's explanation contains uncertainty language
- terminology has very weak lexical similarity to the source event

This provides an additional safeguard against plausible but poorly grounded terminology selections.

---

# Critic minimum confidence

**Configuration key**

```
llm_grounded_critic_min_confidence
```

**Default**

```
0.85
```

**Allowed range**

```
0.00 to 1.00
```

Defines the confidence level below which a grounded AI terminology match is considered worthy of critic review.

This differs from the **resolution minimum confidence**.

Conceptually:

- `llm_grounded_resolution_min_confidence` determines whether a proposed match can be accepted.
- `llm_grounded_critic_min_confidence` determines whether an accepted grounded match should receive additional scrutiny.

---

# Anatomy and body-layer selection

## Body selection batch size

**Configuration key**

```
llm_body_selection_batch_size
```

**Default**

```
8
```

**Admin UI range**

```
1 to 100
```

**Effective runtime range**

```
1 to 16
```

Controls how many clinical events requiring AI-assisted anatomical layer selection are processed in a single model batch.

The body-layer model receives:

- the clinical event
- extracted anatomical information
- the selected SNOMED concept
- bounded installed image-layer candidates

The runtime currently caps the batch size at 16 events.

---

# Body candidate limit

**Configuration key**

```
llm_body_candidate_limit
```

**Default**

```
8
```

**Admin UI range**

```
1 to 100
```

**Effective runtime range**

```
3 to 15
```

Controls how many candidate anatomical image layers may be presented to the body-layer selection model.

Candidate layers are ranked before this limit is applied.

The model may select zero to four layers from those supplied candidates.

The runtime currently guarantees at least three candidates where available and caps the configured limit at 15.

# Import API

## Enable the import API

**Configuration key**

```
api_enabled
```

**Default**

Enabled.

Controls access to:

```
history_api_import.php
```

When disabled, the API returns:

```
HTTP 503
```

to callers before processing an import.

This can be used as an application-level emergency stop for external ingestion.

---

## Require HTTPS

**Configuration key**

```
api_require_https
```

**Default**

Enabled.

Rejects API requests that do not arrive over HTTPS.

The API recognises TLS using:

- the PHP `HTTPS` server variable
- `X-Forwarded-Proto: https`
- server port `443`

Requests received without HTTPS while this setting is enabled return an error.

This setting should remain enabled for any environment carrying real clinical records.

---

# Auto-commit by default

**Configuration key**

```
api_default_auto_commit
```

**Default**

Enabled.

Controls what happens when an API caller does not explicitly supply an `auto_commit` value.

When enabled, an imported document automatically proceeds to the audit and auto-commit stage after extraction.

A caller may explicitly override the default.

If auto-commit is requested, the caller's API token must have the:

```
commit
```

scope.

---

# Maximum document size

**Configuration key**

```
api_max_upload_mb
```

**Default**

```
2 MB
```

**Allowed range**

```
1 to 64 MB
```

Sets the maximum size of a document accepted by the import API.

Uploads larger than the configured value are rejected with:

```
HTTP 413
```

This value is also exposed to the LLM limits structure as the application upload limit.

---

# Ingest queue

Queue settings are shown only when the `ingest_queue` table exists.

When the queue is enabled, API imports are inserted into the queue instead of immediately spawning an independent worker for every request.

The queue provides:

- bounded worker processing
- retries
- job claiming
- failure tracking
- leases
- recovery of abandoned jobs

`queue_worker.php` must be running or scheduled for queued documents to be processed.

---

# Route submissions through the queue

**Configuration key**

```
queue_enabled
```

**Default**

Enabled.

When enabled, new API imports are submitted to `ingest_queue`.

When disabled, API submission falls back to spawning an individual worker for each request.

Queue workers also check this setting before claiming work. If disabled, a daemon worker remains alive but does not claim queued documents.

---

# Attempts before parking as failed

**Configuration key**

```
queue_max_attempts
```

**Default**

```
3
```

**Allowed range**

```
1 to 10
```

Controls how many processing attempts a newly queued item receives before being permanently marked:

```
failed
```

Each claim increments the item's attempt counter.

Failures that still have attempts remaining are automatically returned to the queue.

Retry delay uses quadratic backoff:

```
attempt 1:  60 seconds
attempt 2: 240 seconds
attempt 3: 540 seconds
...
```

with a maximum delay of one hour.

The configured value is copied into each queue item's `max_attempts` field when it is created.

Changing the configuration therefore primarily affects subsequently queued work.

---

# Extraction timeout

**Configuration key**

```
queue_job_timeout
```

**Default**

```
2700 seconds
```

or:

```
45 minutes
```

**Allowed range**

```
60 to 21600 seconds
```

Defines the maximum processing time allowed for an extraction job managed by the queue worker.

A job running beyond this period is considered timed out and enters the queue failure/retry process.

# Claim lease

**Configuration key**

```
queue_lease_seconds
```

**Default**

```
3600 seconds
```

or:

```
60 minutes
```

**Allowed range**

```
60 to 21600 seconds
```

Controls how long a queue item remains claimed by a worker.

When a worker claims an item, a lease expiry timestamp is stored.

If a worker crashes or disappears without completing the item, an expired lease allows another worker to return the item to the queue.

The lease should normally be **comfortably longer than the extraction timeout**.

With the defaults:

```
Job timeout: 2700 seconds / 45 minutes
Lease:       3600 seconds / 60 minutes
```

This gives a 15-minute safety margin.

---

# API token management

`admin_config.php` also provides API bearer-token management.

These values are stored in `api_token`, not `app_config`.

# Token format

New tokens have the form:

```
och_<48 hexadecimal characters>
```

For example:

```
och_0123456789abcdef...
```

The complete plaintext token is displayed only once immediately after creation.

Only its SHA-256 hash is stored in the database.

The first 12 characters are stored separately as a token prefix so administrators can identify tokens without exposing the credential.

---

# Token label

**Maximum length**

```
120 characters
```

A human-readable name identifying the system using the token.

Recommended naming convention:

```
PMS nightly feed
Hospital integration
Test import client
```

Tokens should be named for the calling system rather than an individual user.

---

# Token expiry

The administration screen accepts an expiry period in days.

```
0 = does not expire
```

The UI permits values between:

```
0 and 3650 days
```

An expired token can no longer authenticate API requests.

---

# Token scopes

Three scopes are currently supported.

## `import`

Allows the caller to submit documents for clinical extraction.

## `status`

Allows the caller to retrieve job, document and processing status.

## `commit`

Allows the caller to initiate the audit/auto-commit process.

A token should ideally receive only the scopes required by its integration.

---

# Revoking a token

Revoking a token:

- immediately prevents further authentication
- retains the database record
- retains the audit history
- records who revoked it and when

Revocation should normally be preferred when retiring a credential.

---

# Deleting a token

Deleting permanently removes the token row from the database.

This also removes its direct token audit record.

Use deletion primarily for incorrectly created or unnecessary credentials. Use revocation when the history of the credential should be retained.

---

# Security considerations

# Gemini credential

The Gemini API key is stored in `app_config` because the application must recover the plaintext credential to call Gemini.

Access to the database and configuration administration interface should therefore be appropriately restricted.

# Open Clinical History API tokens

OCH API tokens use a different security model.

The plaintext token is:

1. generated using cryptographically secure random bytes
2. displayed once

3. hashed using SHA-256
4. stored only as a hash

The original token cannot subsequently be recovered from the application database.

# HTTPS

`api_require_https` should remain enabled outside explicitly isolated development environments.

Clinical documents, authentication credentials and patient metadata must not be transmitted over an unencrypted connection.

---

# Effective runtime limits

Several advanced settings currently have different administration-page ranges and runtime ranges.

| Setting | Admin range | Effective runtime |
|---|---|---|
| Source segment characters | 500–100,000 | **1,200–5,000** |
| Extraction segments per batch | 1–100 | **1–10** |
| Transient retries | 0–10 | **0–3** |
| Retry delay | 0–300 sec | **1–30 sec** |
| Grounded candidate limit | 1–100 | **4–20** |
| Grounded search terms/event | 1–20 | **1–8** |
| Repair rounds | 0–10 | **0–3** |
| Repair candidate limit | 1–100 | **4–20** |
| Repair search terms/event | 1–20 | **1–8** |
| Repair maximum terms | 1–20 | **2–8** |
| Body selection batch size | 1–100 | **1–16** |
| Body candidate limit | 1–100 | **3–15** |

`Grounded batch characters` and `Repair batch characters` are additionally constrained dynamically by `llm_max_chars_per_request`.

---

# Current implementation notes

## Provider is currently fixed to Gemini

Although `llm_provider` exists as a configuration setting, `admin_config.php` currently saves:

```
gemini
```

every time settings are saved.

The UI does not currently support selecting another provider.

## Empty token scope selection

There is a current implementation inconsistency in token creation.

The token store correctly rejects a token with no scopes, but `admin_config.php` currently converts an empty scope selection into:

```
ApiTokenStore::ALL_SCOPES
```

before calling the token store.

As a result, manually unticking **all three scopes currently creates a token with all scopes**, rather than rejecting the request.

This should be corrected because it violates least-privilege expectations.

# Some advanced UI ranges do not match runtime behaviour

The administration form permits values that are later clamped by the processing pipeline.

The UI should eventually be changed to reflect the actual effective limits documented above.

# Legacy configuration message

The legacy document chunker still contains an error message instructing the administrator to:

```
Raise the limit in config.php
```

The setting has moved to `app_config` and should instead direct the administrator to:

```
Admin → Configuration
```

---

# Related components

| Component | Responsibility |
|---|---|
| `admin_config.php` | Configuration administration UI |
| `lib/app_config.php` | Configuration persistence, defaults and access |
| `lib/llm.php` | Gemini provider and LLM budget enforcement |
| `lib/history.php` | SNOMED grounding, repair and body-layer processing |
| `lib/history/ingest/HistorySegmentationService.php` | Source segmentation |
| `lib/history/extract/HistoryExtractionService.php` | Extraction batching |
| `lib/history/support/HistoryPipelineAiService.php` | LLM retry handling |
| `lib/api_token.php` | API token creation, validation and revocation |
| `lib/ingest_queue.php` | Ingest work queue |
| `queue_worker.php` | Queue execution |

| | |
|---|---|
| `history_api_import.php` | External import API |
| `snomed_import.php` | SNOMED import interface |
| `snomed_job.php` | SNOMED import worker |

# Configuration defaults summary

| Configuration key | Default |
|---|---|
| `date_format` | `dd/mm/yyyy` |
| `snomed_release_path` | blank / automatic |
| `llm_enabled` | enabled |
| `llm_provider` | `gemini` |
| `llm_gemini_api_key` | blank |
| `llm_gemini_model` | `gemini-3.5-flash-lite` |
| `llm_gemini_timeout_seconds` | `120` |
| `llm_daily_request_cap` | `4000` |
| `llm_daily_token_cap` | `4000000` |
| `llm_max_output_tokens` | `4096` |
| `llm_max_chars_per_request` | `24000` |
| `llm_max_chars_per_chunk` | `4500` |
| `llm_max_chunks_per_document` | `40` |
| `llm_max_chars_per_source_segment` | `2800` |
| `llm_max_chars_per_extraction_batch` | `6500` |
| `llm_max_segments_per_extraction_batch` | `6` |
| `llm_transient_retries` | `1` |
| `llm_transient_retry_delay_seconds` | `4` |
| `llm_enable_snomed_ai_repair` | disabled |
| `llm_grounded_batch_chars` | `18000` |
| `llm_grounded_candidate_limit` | `12` |
| `llm_grounded_search_terms_per_event` | `5` |
| `llm_grounded_repair_rounds` | `2` |

| | |
|---|---|
| `llm_grounded_repair_candidate_limit` | `15` |
| `llm_grounded_repair_search_terms_per_event` | `6` |
| `llm_grounded_critic_enabled` | enabled |
| `llm_grounded_critic_min_confidence` | `0.85` |
| `llm_grounded_repair_max_terms` | `6` |
| `llm_grounded_repair_batch_chars` | `15000` |
| `llm_grounded_resolution_min_confidence` | `0.72` |
| `llm_body_selection_batch_size` | `8` |
| `llm_body_candidate_limit` | `8` |
| `api_enabled` | enabled |
| `api_require_https` | enabled |
| `api_default_auto_commit` | enabled |
| `api_max_upload_mb` | `2` |
| `queue_enabled` | enabled |
| `queue_max_attempts` | `3` |
| `queue_job_timeout` | `2700` seconds |
| `queue_lease_seconds` | `3600` seconds |

# SNOMED Re-index / Learning Queue

The **SNOMED Re-index / Learning Queue** manages clinical terms that Open Clinical History could not safely resolve during patient-history import.

The page is:

```
/process_queue.php
```

During patient import, Open Clinical History deliberately avoids guessing when it cannot establish a sufficiently reliable terminology or anatomical mapping.

Instead, unresolved items are placed into:

```
history_unmatched_queue
```

The Learning Queue allows those items to be reconsidered later using the latest:

- local SNOMED CT dictionaries
- SNOMED terminology indexes
- curated condition catalogue
- curated aliases
- terminology matching logic
- configured grounded LLM resolution

This allows terminology coverage to improve over time without re-extracting the original patient document.

---

# Why the Learning Queue Exists

Clinical language is highly variable.

The same clinical concept may appear in source records using:

- formal terminology
- abbreviations
- local terminology
- spelling variations
- historical terminology
- clinician shorthand
- descriptive phrases
- overly specific or ambiguous wording

For example:

```
myocardial infarction

heart attack

acute MI

NSTEMI

non-ST elevation myocardial infarction
```

may refer to related clinical concepts, but Open Clinical History should not simply assume that every unfamiliar expression means the same thing.

The import pipeline therefore follows a conservative rule:

> If a clinical event cannot be resolved safely, retain it for later learning rather than inventing a mapping.

Conceptually:

```
Patient document
      |
      v
Clinical event extracted
      |
      v
SNOMED resolution
      |
```

```
      +---- reliable match ------> continue import
      |
      +---- unresolved/unsafe
                    |
                    v
         history_unmatched_queue
                    |
                    v
              Learning Queue
```

---

# This Is Not LLM Training

The term **Learning Queue** does not mean that Open Clinical History trains Gemini or modifies an underlying AI model.

No model weights are changed.

Instead, learning occurs through the application's own controlled terminology layer.

For example:

```
Previously unknown clinical term
          |
          v
Add/refine local terminology or alias
          |
          v
Re-run terminology resolution
          |
          v
Valid local SNOMED concept found
          |
          v
Update original event proposal
```

The Open Clinical History terminology database becomes more capable of recognising the language encountered in real clinical records.

---

# Two-Stage Learning

The current architecture deliberately separates terminology learning from changing the patient record.

```
STAGE 1
Terminology learning / re-indexing

process_queue.php
        |
        v
Can this previously unresolved term
now be mapped to SNOMED?
        |
        v
Update event_proposal
        |
        v
Mark learning item resolved



STAGE 2
Apply learned mapping

queue_worker.php
        |
        v
Re-evaluate anatomical mapping
        |
        v
Reapply clinical safety gates
        |
        +---- safe ------> clinical history
        |
        +---- unsafe ----> held for review
```

`process_queue.php` performs **Stage 1**.

Resolving an item on this page does **not by itself add the event to the patient's clinical history**.

That distinction is important.

---

# How Items Enter the Learning Queue

During Audit & Import, an extracted event is automatically committed only when it passes the required safety conditions.

Broadly:

```
Audit passed
AND
SNOMED concept resolved
AND
not negated
AND
not family history
AND
not merely historical-summary text
AND
no anatomical mapping warnings
```

If an event cannot safely be committed, its proposal is rejected from automatic import and an entry is written to:

```
history_unmatched_queue
```

The original event proposal is preserved.

---

# Learning Queue Item Types

Each queue item has an `unmatched_type`.

The current values are:

```
snomed_concept
body_layer
specificity
```

---

# SNOMED Concept

```
snomed_concept
```

means that no suitable SNOMED concept was available when automatic import was attempted.

Conceptually:

```
Clinical event
      |
      v
No sufficiently reliable SNOMED match
      |
      v
SNOMED_CONCEPT learning item
```

This is the most direct terminology-learning case.

---

# Body Layer

```
body_layer
```

means a SNOMED concept was available, but the anatomical-image mapping contained warnings.

For example:

```
Clinical concept resolved
        |
```

```
        v
Anatomical mapping ambiguous
        |
        v
BODY_LAYER learning item
```

A later learning/application cycle can re-evaluate the event using the current image/SNOMED database.

---

# Specificity

```
specificity
```

is used where a SNOMED concept exists and there are no body-layer warnings, but another automatic-import safety condition prevented commitment.

It represents a broader class of events requiring further resolution or review.

---

# Opening the Learning Queue

Open:

**Process Learning Queue**

from the administration menu.

The page first displays the current state of terminology learning.

---

# Queue Statistics

The statistics at the top of the page provide several different views of the learning backlog.

---

# Need Re-indexing

Example:

```
73
NEED RE-INDEXING
8 document(s)
```

This is the number of rows in:

```
history_unmatched_queue
```

whose:

```
review_status = pending
```

These items have not yet been successfully resolved.

The document count shows how many different source documents contain those pending items.

---

# Successfully Re-indexed

Example:

```
568
SUCCESSFULLY RE-INDEXED
20 document(s)
```

These are learning rows where:

```
review_status = resolved
```

A later terminology-resolution attempt successfully found a SNOMED concept.

When this occurs, the linked:

```
event_proposal
```

is updated with the new terminology match.

This does **not necessarily mean the event has already been applied to the patient timeline**.

Terminology resolution and patient-history application are separate stages.

---

# Still Unmatched

Example:

```
7
STILL UNMATCHED
Already attempted, still pending
```

These are pending learning items that have already been through at least one re-indexing attempt but still could not be resolved.

Internally these are identified by a resolution note beginning with:

```
Still unmatched
```

The current note written by the processor is:

```
Still unmatched. Add to condition_alias table.
```

These items remain in the queue so they can be tried again after terminology resources have improved.

---

# Awaiting First Attempt

Example:

```
66
AWAITING FIRST ATTEMPT
Pending with no processing result yet
```

These are queue items where:

```
review_status = pending
```

and there is no existing processing result in `resolution_notes`.

They have not yet been processed by the Learning Queue.

---

# Ignored

Example:

```
0
IGNORED
Blank / deliberately skipped queue items
```

These are queue rows where:

```
review_status = ignored
```

The current processor automatically ignores an item when there is no usable search term.

For example:

```
condition_text = blank

and

event_title = blank
```

produces:

```
Skipped: Blank search term
```

and the queue item is marked ignored.

---

# Unresolved Proposals

The **Unresolved Proposals** statistic examines the underlying:

```
event_proposal
```

table.

It counts proposals in the relevant active states where:

```
matched_concept_id IS NULL
```

This is a different measurement from the Learning Queue itself.

The Learning Queue measures:

```
history_unmatched_queue
```

while this metric looks directly at:

```
event_proposal
```

It is therefore possible for these figures to differ.

---

# Unresolved Not in Pending Queue

This statistic highlights a possible difference between unresolved event proposals and pending learning rows.

The current calculation is:

```
unresolved proposals - pending learning items
```

with the result prevented from falling below zero.

It is intended as a diagnostic indicator rather than an exact row-by-row reconciliation.

A non-zero value can indicate unresolved proposal work that has not yet appeared as pending terminology-learning work.

---

# Queue Items Ever

Example:

```
641
QUEUE ITEMS EVER
568 processed
```

This is the total historical number of rows in:

```
history_unmatched_queue
```

including:

- pending
- resolved
- ignored

The processed value is:

```
resolved + ignored
```

---

# Learning Queue Completion

The completion bar shows how much of the historical Learning Queue has reached a terminal learning state.

The calculation is:

```
resolved + ignored
------------------ × 100
 total queue rows
```

For example:

```
568 resolved
0 ignored
73 pending
```

```
641 total
```

produces:

```
88.6% processed
```

---

# Resolution Rate

The page also displays a **resolution rate**.

This is intended to answer:

> Of the items that have actually received a meaningful resolution attempt, how many have been successfully matched?

The current calculation uses:

```
resolved
---------------------------------- × 100
resolved + still unmatched + ignored
```

New items awaiting their first attempt are excluded.

For example:

```
568 resolved
7 still unmatched
0 ignored
```

produces:

```
98.8%
```

---

# Manual Queue Processing

The main action on the page is:

**Process Next 50 Items**

The Learning Queue is deliberately processed in bounded batches.

Each request processes up to:

```
50
```

pending items.

This prevents one browser request from attempting to resolve the entire learning database in a single operation.

---

# Which Items Are Processed First?

Pending records are selected using:

```
updated_at ASC
id ASC
```

In other words, the oldest pending work is selected first.

Up to 50 rows are loaded.

---

# Queue Rotation After an Unsuccessful Attempt

When an item remains unmatched, its:

```
updated_at
```

timestamp is updated.

Because pending work is ordered by the oldest `updated_at`, the unsuccessful item effectively moves towards the back of the queue.

For example:

```
73 pending


Process first 50
       |
       +---- 45 resolved
       |
       +---- 5 still unmatched
                     |
                     v
              updated_at = now
                     |
                     v
             move behind older
             unattempted items
```

This allows new/unattempted work to receive a first attempt before repeatedly retrying the same difficult terms.

---

# Preventing Concurrent Processing

The selected rows are loaded inside a database transaction using:

```
FOR UPDATE
```

This locks the batch while it is being processed.

The purpose is to prevent two browser requests from processing the same Learning Queue rows simultaneously.

Conceptually:

```
Administrator A
      |
      v
locks next 50 rows
      |
      +-------------------------+
                                |
Administrator B                 |
      |                         |
      v                         |
cannot independently modify     |
those same locked rows          |
                                |
                        transaction completes
```

---

# CSRF Protection

The processing action also requires a session-specific CSRF token.

This prevents another website from causing an authenticated administrator's browser to submit the Learning Queue processing form without their intention.

---

# Choosing the Search Term

For each queue item, Open Clinical History uses:

```
condition_text
```

if available.

Otherwise it falls back to:

```
event_title
```

Conceptually:

```
condition_text present?
      |
   yes|       no
      |        |
      v        v
condition   event_title
   text
      \        /
       \      /
        v    v
       search term
```

If both are empty, the item is ignored.

---

# How Re-indexing Works

The queue does more than perform a simple text lookup.

For each term it calls the same terminology-resolution service used by the clinical ingestion pipeline:

```
HistoryIngest::matchCondition()
```

This allows the item to benefit from improvements made since the original document was processed.

---

# 1. Normalise the Clinical Term

The search text is first normalised.

Open Clinical History also creates useful clinical variations where appropriate.

Examples include UK/US spelling variants such as:

```
ischaemia   ↔ ischemia
oedema      ↔ edema
haemorrhage ↔ hemorrhage
anaemia     ↔ anemia
tumour      ↔ tumor
apnoea      ↔ apnea
```

and terminology variants such as:

```
appendicectomy ↔ appendectomy
```

---

# 2. Simplify Clinical Phrases

The matcher can generate less contextual variants of a term.

For example:

```
history of myocardial infarction
```

may also be searched as:

```
myocardial infarction
```

Likewise modifiers such as:

```
previous
prior
diagnosis of
diagnosed with
residual
```

can be removed for candidate discovery.

Some severity or temporal prefixes may also be stripped when searching for the underlying clinical concept.

These transformations help retrieve candidates.

They do not by themselves automatically determine the final SNOMED concept.

---

# 3. Check the Curated Condition Catalogue and Aliases

If installed, Open Clinical History checks:

```
condition_catalogue
condition_alias
```

first.

An alias provides a controlled mapping between terminology encountered in source records and a known catalogue entry.

Conceptually:

```
Local clinical expression
        |
        v
condition_alias
        |
        v
condition_catalogue
        |
        v
SNOMED concept
```

This is one of the principal mechanisms by which the system can learn terminology encountered in real-world clinical data.

---

# Why Aliases Matter

Suppose repeated source records use:

```
local clinical expression X
```

and Open Clinical History cannot reliably resolve it.

After an administrator establishes that it should map to a known catalogue concept, an appropriate alias can be added.

The next Learning Queue pass can then resolve:

```
local clinical expression X
        |
        v
alias recognised
        |
        v
known catalogue concept
        |
        v
SNOMED CT
```

The source patient record does not need to be extracted again.

---

# 4. Search the Local SNOMED Dictionary

If no curated match exists, the matcher searches the compact SNOMED indexes:

```
snomed_health_history_lookup
snomed_health_history_term_lookup
```

It first looks for exact matches.

A single semantically compatible exact result can be accepted deterministically.

Possible exact methods include:

```
exact_fsn
exact_synonym
```

---

# 5. Search for SNOMED Candidates

When a unique exact result is unavailable, Open Clinical History searches its local SNOMED terminology index for candidate concepts.

Potential candidates are ranked locally before any AI involvement.

The number of candidates is bounded by the configured terminology-resolution limits.

---

# 6. Grounded AI Resolution

If an LLM is configured, ambiguous candidate sets can be passed through the grounded SNOMED-resolution process.

Importantly, the LLM does **not** supply an arbitrary SNOMED ID.

Instead:

```
Clinical term
      |
      v
Local SNOMED search
      |
      v
Candidate A
Candidate B
Candidate C
      |
      v
LLM selects from supplied candidates
```

```
      |
      v
Local SNOMED concept
```

The LLM receives opaque candidate keys and must select one of the candidates Open Clinical History supplied.

It is explicitly instructed not to invent a SNOMED identifier.

---

# Grounded Repair

Where enabled, the normal SNOMED repair process may also be used.

This can generate revised terminology search expressions and retry local candidate resolution.

The number of repair rounds and candidate limits are controlled under:

**Admin → Configuration**

using the grounded SNOMED settings.

---

# Fallback if the LLM Fails

If AI-assisted resolution raises an error, `process_queue.php` does not automatically abandon the queue batch.

`matchCondition()` falls back to deterministic local candidate matching.

This allows local terminology improvements and exact aliases to remain useful even when the configured LLM is temporarily unavailable.

---

# When a Match Is Found

If a valid SNOMED concept is found, the linked:

```
event_proposal
```

is updated.

The following values are written:

```
matched_concept_id
matched_term
match_method
match_confidence
```

For example, conceptually:

```
event_proposal #482

Before:
matched_concept_id = NULL

After:
matched_concept_id = <SNOMED ID>
matched_term       = <SNOMED term>
match_method       = <resolution method>
match_confidence   = <confidence>
```

The Learning Queue row is then changed to:

```
review_status = resolved
```

with a resolution note similar to:

```
Matched to SNOMED ID: <concept-id> (<term>)
```

---

# Successfully Re-indexed Does Not Mean Committed

This is one of the most important behaviours of the page.

When the queue says:

```
Successfully re-indexed
```

it means:

> A previously unresolved event proposal now has a SNOMED terminology match.

It does **not** mean:

> The event has automatically been inserted into the patient's clinical history.

The original clinical safety controls are deliberately retained.

---

# Stage 2: Applying Learned Mappings

The separate learned-mapping application process can subsequently reconsider the event.

It uses the newly resolved SNOMED concept to:

1. recalculate anatomical layers
2. update the event proposal mapping information
3. reapply the original audit/safety conditions
4. either commit the event or hold it for review

Conceptually:

```
process_queue.php
        |
        v
SNOMED learned
        |
        v
event_proposal updated
```

```
        |
        v
Apply Learned Mapping
        |
        v
Rebuild anatomy
        |
        v
Safety gate
        |
        +---- safe ----------> Commit
        |
        +---- not safe ------> Hold
```

The original source-document extraction is not repeated.

---

# Existing Committed Events

The two-stage learning architecture also supports updating an event that has already been committed.

When learned terminology is later applied to such an event, Open Clinical History can refresh:

- SNOMED coding
- anatomical mapping
- associated visual sites

without creating another clinical event.

The operation records that the coding was updated through the learning/re-index process.

---

# Learned Mapping Safety Gates

A newly learned terminology match is still not sufficient by itself for automatic commitment.

The application stage checks conditions including:

- the original clinical audit passed
- the source was not negated
- the source was not family history
- the source was not merely a historical summary
- the terminology match is sufficiently strong
- the matching method is sufficiently reliable
- required anatomical mappings are available

The current learned-application code requires a strong terminology result for unattended commitment.

Weak results remain held.

---

# Still-Unmatched Items

If no suitable SNOMED concept is found, the queue item remains:

```
review_status = pending
```

and receives:

```
Still unmatched. Add to condition_alias table.
```

Its `updated_at` timestamp is refreshed so it moves towards the back of the pending queue.

The item can then be reconsidered after changes such as:

- adding a curated alias
- correcting a catalogue entry
- rebuilding the SNOMED indexes
- installing a newer SNOMED release
- improving terminology matching
- improving grounded resolution

---

# Processing a New Batch

Click:

**Process Next 50 Items**

again whenever terminology resources have changed or more queue work remains.

The button automatically shows the number of items in the next batch.

For example:

```
Process Next 50 Items (50)
```

or, where fewer remain:

```
Process Next 50 Items (23)
```

When there are no pending items, the button is disabled.

---

# Processing Result

After a batch completes, the page reports:

```
Queue processing complete.
Evaluated 50 term(s):
42 re-indexed,
8 still unmatched,
0 ignored.
```

Note that blank items marked ignored are not included in the `Evaluated` count because no terminology lookup was attempted for them.

---

# Outstanding: Next in Queue

The **Outstanding** table displays up to the next 50 pending records.

Columns include:

| Column | Meaning |
|---|---|
| ID | Learning Queue row identifier |

| Column | Meaning |
| --- | --- |
| Type | `SNOMED_CONCEPT`, `BODY_LAYER` or `SPECIFICITY` |
| Original Term | Clinical term being reconsidered |
| Attempt State | Whether this is new or has previously failed |
| Last Attempt Notes | Result of the previous attempt |

---

# Attempt State: NEW

```
NEW
```

means no previous Learning Queue result has been recorded.

The notes display:

```
Awaiting first processing attempt...
```

---

# Attempt State: RETRY NEEDED

```
RETRY NEEDED
```

means the term has already been processed but could not yet be resolved.

The notes display:

```
Still unmatched. Add to condition_alias table.
```

Because failed attempts are moved towards the back of the queue, newly encountered terms are given an opportunity before difficult terms are repeatedly retried.

---

# Processed Log

The bottom of the page displays the most recent:

```
50
```

non-pending Learning Queue items.

These include:

```
resolved
ignored
```

The table shows:

- queue ID
- original clinical term
- final learning status
- resolution notes

For a resolved item, the notes contain the matched SNOMED concept.

For example:

```
Matched to SNOMED ID: <concept-id> (<SNOMED term>)
```

---

# When Should the Learning Queue Be Processed?

The queue is particularly useful after:

- patient histories have produced unresolved terminology
- curated aliases have been added
- terminology dictionaries have been changed
- SNOMED CT has been updated
- SNOMED indexes have been rebuilt
- terminology-resolution logic has been improved

- anatomical mappings have been rebuilt

A typical learning cycle is:

```
Patient imports
      |
      v
Unresolved terminology accumulates
      |
      v
Review recurring terminology gaps
      |
      v
Improve aliases/catalogue/SNOMED data
      |
      v
Process Learning Queue
      |
      v
Previously unresolved terms resolve
      |
      v
Apply learned mappings
      |
      v
Patient histories improve
```

---

# Relationship to SNOMED Updates

Installing a newer SNOMED CT release may introduce:

- new concepts
- new descriptions
- changed preferred terminology
- additional synonyms

After importing the new release and rebuilding the required lookup databases, the Learning Queue can be processed again.

A term that previously had no valid local match may now resolve.

This avoids having to re-import the patient's original source document merely because the terminology database improved.

---

# Relationship to the Image/SNOMED Database

A terminology resolution can also change the anatomical mapping available to an event.

After a SNOMED concept has been learned, the second-stage application process uses the current:

```
body_layer_lookup
body_layer_term
```

tables to determine the appropriate visual layers.

This means improvements made through:

**Image/SNOMED Summary Build**

can also benefit previously unresolved patient events.

---

# Important Distinction Between the Queues

Open Clinical History contains more than one queue with different responsibilities.

## Ingest Queue

```
ingest_queue
```

controls document-processing jobs.

It manages:

- patient-document extraction
- worker concurrency
- retries
- worker leases
- learned-mapping application jobs

# Learning Queue

```
history_unmatched_queue
```

contains unresolved **clinical event mappings**.

It manages terminology and mapping gaps discovered during patient processing.

Conceptually:

```
INGEST QUEUE

document
   |
   v
worker
   |
   v
clinical extraction



LEARNING QUEUE

unresolved clinical event
   |
   v
terminology re-index
   |
```

```
   v
learned mapping
```

They should not be confused.

---

# Recommended Operational Workflow

When unresolved terminology is present:

```
1. Open Process Learning Queue
        |
        v
2. Review outstanding terms
        |
        v
3. Identify recurring terminology gaps
        |
        v
4. Improve curated aliases/catalogue
   or SNOMED/index data where appropriate
        |
        v
5. Process Next 50 Items
        |
        v
6. Review successful and unmatched counts
        |
        +---- unresolved ----> investigate/curate
        |
        +---- resolved ------> learned mapping ready
        |
        v
7. Apply learned mappings
        |
```

```
        v
8. Review held items
```

---

# Do Not Add Aliases Blindly

A recurring unfamiliar term should not automatically be assigned to the first plausible SNOMED concept.

Before adding a curated alias, confirm that:

- the source expression has a stable clinical meaning
- the chosen SNOMED concept represents that meaning
- the semantic type is appropriate
- the mapping is not more specific than the source
- laterality is not being incorrectly encoded
- an abbreviation is not ambiguous between specialties

For example, an alias should not convert an ambiguous abbreviation into a specific diagnosis unless that meaning is genuinely reliable within the intended context.

The learning mechanism is deliberately governed because the resulting mappings can eventually become part of a patient's clinical record.

---

# Summary

`process_queue.php` is the **terminology-learning and SNOMED re-indexing stage** of Open Clinical History.

It takes events that were previously unresolved:

```
Unresolved clinical term
```

and asks:

```
Can the current Open Clinical History
terminology system resolve this now?
```

using:

```
Curated aliases
      +
Local SNOMED CT
      +
Clinical spelling/term normalisation
      +
Local candidate search
      +
Bounded grounded AI resolution
```

When a reliable concept is found:

```
history_unmatched_queue
        |
        v
resolved

event_proposal
        |
        v
SNOMED mapping updated
```

A separate governed application stage then determines whether that learned mapping can safely update the patient's clinical history.

This separation allows Open Clinical History to **learn from unresolved terminology without bypassing the clinical safety controls that govern the patient record**.

# Gemini / LLM Usage Dashboard

The **Gemini Usage** dashboard provides operational visibility into the Large Language Model usage generated by Open Clinical History.

The page is:

```
/gemini_usage.php
```

It displays:

- current LLM configuration state
- today's request budget
- today's token budget
- historical request and token usage
- usage by provider and model
- current configured provider limits
- recent patient-document LLM usage

The page is **read-only**.

Configuration changes are made under:

**Admin → Configuration**

---

# Purpose

Open Clinical History uses an LLM during several processing operations, including:

- patient-history segmentation
- clinical event extraction
- document reconciliation
- grounded SNOMED resolution
- SNOMED repair
- anatomical body-layer selection

- clinical audit
- image/SNOMED catalogue construction

Some patient histories can require dozens of LLM calls.

The Usage dashboard provides a single location for monitoring that activity.

Conceptually:

```
Open Clinical History
        |
        +--> Source segmentation
        |
        +--> Event extraction
        |
        +--> Reconciliation
        |
        +--> SNOMED resolution
        |
        +--> Anatomy selection
        |
        +--> Clinical audit
        |
        v
Configured LLM provider
        |
        v
Usage recorded
        |
        v
llm_usage
        |
        v
Gemini Usage Dashboard
```

---

# Usage Tracking

LLM usage is recorded in:

```
llm_usage
```

Usage is grouped by:

```
date
+
provider
+
model
```

For example:

```
2026-08-25
gemini
gemini-3.5-flash-lite
```

has its own request and token counters.

This allows Open Clinical History to retain usage history even if the application or worker processes restart.

---

# Dashboard Refresh

The Usage dashboard automatically refreshes every:

```
30 seconds
```

This makes it useful while:

- importing patient histories
- running large extraction batches
- rebuilding image/SNOMED mappings
- processing terminology workloads

There is no need to manually reload the page continuously.

---

# Database Day

At the top of the dashboard, the page displays:

```
database day YYYY-MM-DD
```

The current date is obtained from the database using:

```
CURDATE()
```

where possible.

This is significant because daily usage limits are also recorded according to the database day.

The database server's timezone should therefore be configured appropriately for the installation.

---

# Time Period

Historical usage can be viewed over:

```
7 days
30 days
90 days
1 year
```

The default is:

```
30 days
```

The period can also be selected through the URL:

```
gemini_usage.php?days=7

gemini_usage.php?days=30

gemini_usage.php?days=90
```

```
gemini_usage.php?days=365
```

Any unsupported value falls back to 30 days.

---

# LLM Status

The status indicators near the top of the page show the current runtime configuration.

For example:

```
LLM enabled

Gemini key configured

provider gemini

active model gemini-3.5-flash-lite
```

---

# LLM Enabled

This reflects:

```
llm_enabled
```

from **Admin → Configuration**.

If disabled, Open Clinical History will not create an LLM provider for processing requests.

The usage dashboard remains available so historical usage can still be inspected.

---

# Gemini Key Configured

This indicates whether:

```
llm_gemini_api_key
```

contains a value.

The API key itself is **never displayed** on the Usage dashboard.

The indicator shows only:

```
configured
```

or:

```
missing
```

---

# Provider

The dashboard displays the currently configured:

```
llm_provider
```

The current implementation uses:

```
gemini
```

The underlying LLM architecture is provider-aware, so usage records retain the provider name rather than assuming all historical requests necessarily belong to Gemini.

---

# Active Model

The currently configured model comes from:

```
llm_gemini_model
```

For example:

```
gemini-3.5-flash-lite
```

Changing the configured model does not remove usage recorded against an earlier model.

Historical usage remains visible separately.

---

# Today's Request Budget

The first budget panel displays:

```
Today's request budget
```

For example:

```
2,145 / 50,000
```

The configured maximum comes from:

```
llm_daily_request_cap
```

The counter represents requests recorded today for the **currently active provider and model**.

The dashboard also calculates:

- percentage used
- requests remaining

For example:

```
4.29% used · 47,855 remaining
```

---

# Request Cap Enforcement

Before each LLM request, Open Clinical History checks the current usage for:

```
today
+
active provider
+
active model
```

If the number of requests has already reached the configured cap, the request is blocked before Gemini is contacted.

For example:

```
Recorded today: 50,000
Configured cap: 50,000
        |
        v
New LLM request
        |
        v
BLOCKED
```

The caller receives an LLM budget error explaining that the daily request cap has been reached.

---

# Unlimited Request Budget

A configured request cap of:

```
0
```

means:

```
Unlimited
```

The dashboard displays the infinity symbol for the budget rather than calculating a percentage.

---

# Today's Token Budget

The second budget panel displays the token allowance.

For example:

```
2,400,000 / 8,000,000
```

The configured maximum comes from:

```
llm_daily_token_cap
```

The token total is:

```
prompt tokens
+
output/thinking tokens
```

---

# Prompt Tokens

Prompt tokens are reported by Gemini as:

```
promptTokenCount
```

These represent the input supplied to the model, including the effective system and user prompt.

---

# Output / Thinking Tokens

For Gemini 3.x models, Open Clinical History includes both:

```
candidate/output tokens
+
thinking tokens
```

in the recorded output figure.

Conceptually:

```
output_tokens =
candidatesTokenCount
+
thoughtsTokenCount
```

This is deliberate.

Thinking tokens are still part of model consumption, even though the model's internal thinking content is not included in the clinical JSON returned to the application.

Therefore:

> **Output / thinking** is a usage figure, not simply the number of visible response tokens.

---

# Token Cap Enforcement

Before calling the LLM, Open Clinical History calculates:

```
today's prompt tokens
+
today's output/thinking tokens
```

If that value has already reached the configured daily token cap, the request is blocked.

For example:

```
Current recorded use: 8,000,000
Configured limit:     8,000,000
```

```
Next request → blocked
```

---

# Token Caps Are Pre-request Guards

The exact number of tokens a request will consume cannot be known until the provider has processed it.

The token budget is therefore checked **before** a request using already recorded consumption.

For example:

```
Current usage: 7,990,000
Daily cap:     8,000,000
        |
        v
Request allowed
        |
        v
Request consumes 20,000 tokens
        |
        v
New recorded total: 8,010,000
```

The cap can therefore be exceeded slightly by the request that crosses the boundary.

With multiple workers making requests concurrently, several requests can also pass their pre-request checks before another worker's usage has been recorded.

The daily caps should therefore be treated as **protective application limits**, not exact billing ceilings.

---

# Daily Budgets Are Per Provider and Model

This is an important implementation detail.

The budget gate reads usage using:

```
usage_day
+
provider
+
model
```

Therefore:

```
gemini / model A
```

and:

```
gemini / model B
```

have separate daily usage buckets.

For example:

```
Model A today
48,000 / 50,000 requests

Administrator changes to Model B

Model B today
0 / 50,000 requests
```

The dashboard will now show Model B's budget because Model B is the active model.

The previous Model A usage remains recorded and visible in historical/provider statistics.

This means the configured daily cap is currently effectively a:

> **daily cap per provider/model combination**

rather than one global LLM allowance across the entire installation.

---

# Today: Active Model

The first summary card displays today's usage for the active provider/model.

It includes:

```
requests
prompt tokens
output/thinking tokens
```

For example:

```
TODAY · ACTIVE MODEL


315 requests
482,741 prompt
211,309 output/thinking
```

If the configured model was changed today, this card shows only usage for the **new active model**.

It does not combine today's usage from earlier models.

---

# Period Token Usage

The next card shows total token usage over the selected reporting period.

For example:

```
30 DAY TOKENS
```

```
17,128,086


11,660,943 prompt
5,467,143 output/thinking
```

Unlike the daily budget panels, the historical period totals include:

> **all recorded providers and models**

within the selected period.

The total is:

```
prompt tokens
+
output/thinking tokens
```

---

# Period Request Usage

The request card displays total requests over the selected reporting period.

For example:

```
30 DAY REQUESTS


8,139
```

It also calculates:

```
average tokens/request
average prompt tokens/request
average output tokens/request
```

The average total is:

```
period prompt tokens + period output tokens
-------------------------------------------
```

```
            period requests
```

This can be useful for spotting changes in workload characteristics.

For example, a significant increase in average tokens per request may indicate:

- larger extraction batches
- larger prompts
- more complex patient records
- increased SNOMED candidate context
- a change in model behaviour
- a configuration change

---

# Lifetime Recorded

The **Lifetime Recorded** card displays all usage currently present in:

```
llm_usage
```

It includes:

- total tokens
- total requests
- first recorded usage day
- last recorded usage day

For example:

```
17,128,086 tokens


8,139 requests


2026-07-30 → 2026-08-24
```

The term **lifetime** means:

> The lifetime of the usage records currently retained in the Open Clinical History database.

It does not necessarily mean all Gemini usage since the installation was originally created.

If the `llm_usage` table has been cleared, recreated or introduced after the system began operating, earlier provider usage will not appear.

---

# Usage Trend

The **Usage trend** section contains two charts.

# Tokens per day

Displays:

```
prompt
+
output/thinking
```

tokens recorded on each day.

The chart automatically scales relative to the highest-use day in the selected period.

Hovering over a bar displays the date and token count.

---

# Requests per day

Displays the total number of LLM requests recorded each day.

This is useful for identifying:

- large import runs
- SNOMED/image rebuild activity
- bursts of patient processing
- changes in processing volume

---

# Zero-use Days

The chart deliberately includes days where no LLM usage was recorded.

For example:

```
1 Aug   0
2 Aug   0
3 Aug   4,182
4 Aug   0
```

This ensures that the horizontal timeline represents the complete selected period rather than showing only days on which activity occurred.

---

# Provider and Model Usage

The **Provider and model usage** table breaks the selected period down by:

```
provider
+
model
```

Columns include:

| Column | Meaning |
|---|---|
| Provider | LLM provider |
| Model | Model used |
| Requests | Number of recorded requests |
| Prompt | Prompt/input tokens |
| Output / thinking | Output plus Gemini thinking tokens |
| Total tokens | Prompt + output/thinking |
| Avg / request | Average total tokens per request |

The currently configured provider/model is marked:

```
active
```

---

# Historical Model Changes

If the model has changed, the table may contain multiple rows.

For example:

```
gemini   gemini-2.x-model          1,240 requests
gemini   gemini-3.5-flash-lite     6,899 requests   active
```

This makes it possible to see how much processing was performed by each model.

The statistics are not rewritten when the active model changes.

---

# Current Provider Limits

The **Current provider limits** section displays the settings currently defined under:

**Admin → Configuration**

These include:

```
Daily requests
Daily tokens
Max output / request
Max prompt chars
Timeout
```

These values are read directly from `app_config`.

The dashboard does not provide controls for modifying them.

---

# Daily Requests

Configuration key:

```
llm_daily_request_cap
```

Controls the daily request gate.

A value of:

```
0
```

means unlimited.

---

# Daily Tokens

Configuration key:

```
llm_daily_token_cap
```

Controls the daily recorded token gate.

The total includes:

```
prompt
+
output
+
Gemini thinking tokens
```

A value of:

```
0
```

means unlimited.

---

# Maximum Output per Request

Configuration key:

```
llm_max_output_tokens
```

Controls the `maxOutputTokens` value sent with Gemini requests.

This is **not a daily limit**.

It limits an individual model response.

If Gemini reaches this maximum before completing a response, Open Clinical History detects:

```
MAX_TOKENS
```

and treats the response as failed rather than attempting to consume truncated clinical JSON.

The resulting error advises reducing the input size or increasing the output-token allowance.

---

# Maximum Prompt Characters

Configuration key:

```
llm_max_chars_per_request
```

This is measured in:

```
characters
```

not tokens.

Before an LLM request is made, Open Clinical History calculates:

```
strlen(system prompt)
+
strlen(user prompt)
```

If the result exceeds the configured value, the request is blocked locally.

This provides protection against accidentally sending an unexpectedly large prompt.

---

# Timeout

Configuration key:

```
llm_gemini_timeout_seconds
```

This controls the HTTP timeout for an individual Gemini request.

For example:

```
120 sec
```

means an individual request may wait for up to approximately two minutes for the provider response before the request is treated as failed.

Transient pipeline retry behaviour is controlled separately by the retry settings under **Admin → Configuration**.

---

# Recent Document Usage

The **Recent document usage** section displays the latest:

```
20
```

patient-history documents with recorded LLM activity.

A document is included when any of the following is greater than zero:

```
requests
prompt_tokens
output_tokens
```

The records are ordered by the document's most recent update time.

---

# Patient / Document

Where patient information is available, the dashboard attempts to display the patient name from patient attributes.

It checks, in order:

```
display_name
full_name
name
```

If no name is available, the patient record number is used.

If neither is available:

```
Unlinked patient
```

is displayed.

Below the patient identifier the dashboard displays:

```
document ID
source filename
```

For example:

```
Example Patient


doc 142 · specialist-letter.txt
```

---

# Document Status

The status column shows the current `history_document` state.

Depending on the workflow this may include states such as:

```
uploaded
processing
proposed
committed
failed
```

This provides useful context when reviewing LLM consumption.

A document with unusually high usage that is still processing or failed may warrant investigation.

---

# Document Requests

The request counter represents LLM activity accumulated against that:

```
history_document
```

during clinical processing.

It can include model calls made by multiple pipeline stages rather than only the initial clinical extraction.

For example:

```
Document
   |
   +--> segmentation requests
   |
   +--> extraction requests
   |
   +--> SNOMED resolution requests
   |
```

```
    +--> body-layer requests
    |
    +--> reconciliation/audit requests
    |
    v
document request total
```

This is why a single patient document can result in many LLM requests.

---

# Document Prompt and Output Usage

For each recent document, the dashboard displays:

```
Prompt
Output
Total
```

where:

```
Total = Prompt + Output
```

These counters are cumulative for that document's recorded processing activity.

They are useful for comparing how computationally expensive different patient records have been.

---

# Recent Documents Are Not Filtered by the Period

# Selector

The:

```
7 days
30 days
90 days
1 year
```

selector controls the historical usage statistics and charts.

It does **not** change the **Recent document usage** section.

That table always displays the latest 20 documents with recorded LLM usage, regardless of which reporting period is selected.

---

# Why One Document May Use Many Requests

Open Clinical History deliberately separates the clinical processing problem into multiple bounded model operations.

For example:

```
Large patient history
        |
        v
Source segmentation
        |     several requests
        v
Event extraction
        |     several requests
        v
Document reconciliation
```

```
        |     one or more requests
        v
SNOMED grounding
        |     one or more requests
        v
Body mapping
        |     one or more requests
        v
Clinical audit
              one or more requests
```

A document showing:

```
60 requests
```

does not imply the document was sent to the LLM 60 times in its entirety.

It reflects the series of bounded processing operations required to construct and audit the final clinical history.

---

# Usage and Application Budgets

The usage dashboard is not merely informational.

The values recorded in:

```
llm_usage
```

are also used by the LLM budget gate.

Conceptually:

```
New LLM request
      |
      v
Read today's llm_usage
```

```
      |
      +--> Request cap reached? ----> BLOCK
      |
      +--> Token cap reached? ------> BLOCK
      |
      +--> Prompt too large? -------> BLOCK
      |
      v
Call Gemini
      |
      v
Receive usage metadata
      |
      v
Update llm_usage
```

The budget state therefore survives:

- browser reloads
- PHP request completion
- application-worker restarts
- separate worker processes

---

# Usage Is Recorded After a Successful Model Response

The shared `llm_usage` counter is updated after the configured LLM provider successfully returns a usable response.

This has an important operational implication.

Requests that fail before Open Clinical History receives a usable result, for example:

- HTTP errors
- transport failures
- timeouts
- safety refusal

- truncated `MAX_TOKENS` responses
- empty responses

are not necessarily represented in `llm_usage`.

Therefore:

> The Open Clinical History usage dashboard should be treated as an operational application-usage record, not as an authoritative Gemini billing statement.

For definitive provider billing information, use the billing/usage facilities supplied by the LLM provider.

---

# No Cost Estimate

The dashboard deliberately does **not** attempt to calculate monetary cost.

Gemini pricing can depend on:

- model
- API tier
- input tokens
- output tokens
- thinking tokens
- provider pricing changes
- account arrangements

Open Clinical History does not store a pricing table.

Displaying a calculated dollar figure would therefore risk becoming misleading as provider pricing changes.

The dashboard reports the underlying usage metrics instead.

---

# Missing Usage Table

If:

```
llm_usage
```

does not exist, the dashboard displays:

```
LLM usage table missing.
```

Historical usage cannot be displayed until the required database schema has been installed.

This does not itself establish whether Gemini is configured correctly.

---

# Dashboard Query Failure

If the table exists but usage cannot be queried, the page reports:

```
Usage dashboard error.
```

with the underlying database error.

The remainder of the page attempts to remain available where possible.

---

# Missing Document Tables

The usage dashboard can operate without the patient-document detail section.

If:

```
history_document
```

is unavailable, aggregate LLM statistics can still be displayed.

If:

```
patient
```

is unavailable, the document query can still operate without patient demographics.

This makes the high-level LLM usage monitoring relatively independent of the patient-history reporting tables.

---

# Privacy Consideration

The **Recent document usage** section can display:

- patient name
- patient record number
- source filename
- document status

The page should therefore be considered an administrative view containing potentially identifiable clinical information.

Access should be restricted to authorised Open Clinical History administrators and operators.

The dashboard itself does not display the contents of the patient document or the Gemini API key.

---

# Interpreting the Dashboard

## High token usage with relatively few requests

This can indicate:

- large prompts
- large extraction batches
- extensive SNOMED candidate context
- large model responses

Check:

```
Max prompt chars
Max output / request
```

and the document-level usage.

---

# High request count with relatively low token usage

This may indicate:

- many small source segments
- many small clinical records
- conservative batch sizes
- terminology resolution occurring in numerous small batches

Review the LLM pipeline batching settings under **Admin → Configuration**.

---

# One document uses significantly more tokens than others

Check:

- source-document size
- number of extracted events
- extraction/reconciliation complexity
- SNOMED repair activity
- anatomical mapping activity
- audit behaviour

The document's processing status can also indicate whether repeated or incomplete processing should be investigated.

---

# Today's budget says zero but historical usage is high

This is normal when:

- no requests have been made today; or
- the active model has changed.

The daily budget panels display only the currently active:

```
provider + model
```

while the historical period totals aggregate all recorded providers and models.

---

# Recommended Operational Use

The Usage dashboard is useful during:

# Patient import testing

Watch request and token consumption while tuning the extraction pipeline.

# Large import batches

Confirm that processing is not approaching configured daily limits.

# Model changes

Compare historical usage between old and new models.

# Configuration tuning

Observe whether changes to:

- chunk sizes
- extraction batch sizes
- grounded candidate limits
- repair rounds
- body mapping batch sizes

materially alter LLM consumption.

# Troubleshooting

Identify documents associated with unusually high model usage.

---

# Related Configuration

The primary settings shown or enforced by this dashboard are:

```
llm_enabled
llm_provider
llm_gemini_api_key
llm_gemini_model
llm_gemini_timeout_seconds
llm_daily_request_cap
llm_daily_token_cap
llm_max_output_tokens
llm_max_chars_per_request
```

Additional pipeline settings can also materially influence overall usage, including:

```
llm_max_chars_per_source_segment
llm_max_chars_per_extraction_batch
llm_max_segments_per_extraction_batch
llm_transient_retries
llm_grounded_candidate_limit
llm_grounded_search_terms_per_event
llm_grounded_repair_rounds
llm_body_selection_batch_size
llm_body_candidate_limit
```

These are managed under:

**Admin → Configuration**

---

# Related Components

| Component | Purpose |
|---|---|
| `gemini_usage.php` | Read-only usage dashboard |
| `lib/llm.php` | Provider interface, Gemini client, budget enforcement and usage recording |
| `lib/app_config.php` | Runtime LLM configuration and limits |
| `llm_usage` | Daily provider/model usage counters |
| `history_document` | Per-document accumulated LLM usage |
| `admin_config.php` | LLM provider and budget configuration |

---

# Summary

The Gemini Usage dashboard answers four operational questions:

```
Is the LLM configured?
        |
        v
```

```
How much have we used today?
        |
        v
How much have we used historically?
        |
        v
Which patient-processing workloads are consuming it?
```

The global usage record is stored by:

```
day
+
provider
+
model
```

while patient histories also retain their own document-level usage totals.

Together these provide both:

```
SYSTEM VIEW
How much LLM capacity is Open Clinical History using?

and

DOCUMENT VIEW
Which patient-processing jobs are using it?
```

The dashboard provides visibility only.

**Admin → Configuration remains the source of truth for provider settings and budget limits.**