Package: BFHllm (via r-universe)

June 4, 2026

Title LLM Integration Framework for BFH Packages

Version 0.2.0

Description AI-driven insights and text generation for healthcare quality improvement. Provides LLM chat interface, RAG (Retrieval Augmented Generation) integration with knowledge stores, and SPC-specific improvement suggestions. Designed for standalone use with BFHcharts or integration in Shiny applications.

License MIT + file LICENSE

URL https://github.com/johanreventlow/BFHllm

BugReports https://github.com/johanreventlow/BFHllm/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports ellmer (>= 0.4.0), ragnar (>= 0.1.0), digest, jsonlite, yaml

Suggests testthat (>= 3.0.0), shiny (>= 1.7.0), mockery, withr, BFHcharts (>= 0.4.0)

Remotes johanreventlow/BFHcharts

Config/testthat/edition 3

Config/pak/sysreqs cmake libicu-dev libpng-dev libxml2-dev libssl-dev python3 xz-utils

Repository https://johanreventlow.r-universe.dev

Date/Publication 2026-06-04 11:41:19 UTC

RemoteUrl https://github.com/johanreventlow/BFHllm

RemoteRef HEAD

RemoteSha f3bfd92a2ad423eb443efbe4a6c8b6a48fe5822b

Contents

`bfhllm_build_knowledge_store` *Build Knowledge Store*

Description

Builds Ragnar knowledge store from markdown documents. This is typically run once during package development, not at runtime.

Usage

```
bfhllm_build_knowledge_store(docs_path, output_path)
```

Arguments

docs_path	Character string, path to markdown documents directory
output_path	Character string, path to output store directory

Details

Requirements:

- ragnar package installed
- GOOGLE_API_KEY or GEMINI_API_KEY set (for embeddings)
- Markdown documents in docs_path

Process:

1. Initialize Ragnar store with Gemini embeddings
2. Read markdown files
3. Chunk documents (markdown-aware)
4. Generate embeddings via Gemini API
5. Build BM25 search index
6. Persist store to output_path

Value

invisible(NULL)

Examples

```
## Not run:
# Build store (typically run in data-raw/build_ragnar_store.R)
bfhllm_build_knowledge_store(
  docs_path = "inst/spc_knowledge",
  output_path = "inst/ragnar_store"
)

## End(Not run)
```

---

bfhllm_build_prompt *Build Prompt from Components*

Description

Concatenates multiple prompt components into a single prompt string. Useful for building complex prompts from modular pieces.

Usage

```
bfhllm_build_prompt(..., sep = "\n\n")
```

Arguments

...	Character strings or named list, prompt components
sep	Character string, separator between components (default: "\n\n")

Details

Named Arguments: If using named arguments, names are ignored (only values are used).

NULL Handling: NULL components are automatically filtered out.

Whitespace: Leading/trailing whitespace is trimmed from each component before concatenation.

Value

Character string with concatenated prompt

Examples

```
## Not run:
# Build SPC analysis prompt
system_msg <- "You are an SPC analysis expert."
context <- "Chart type: run chart. Data points: 24."
question <- "What does stable variation indicate?"

prompt <- bfhllm_build_prompt(
  system_msg,
  context,
  question,
  sep = "\n\n"
)

# With conditional components
prompt <- bfhllm_build_prompt(
  "Analyze this chart:",
  if (has_signals) "Signals detected: Run of 8",
  "Suggest actions."
)

## End(Not run)
```

`bfhllm_cache_create` *Create Generic In-Memory Cache*

Description

Creates a generic cache object for storing LLM responses. Cache uses hash-based keys and enforces TTL (time-to-live) on retrieval.

Usage

```
bfhllm_cache_create(ttl_seconds = 3600)
```

Arguments

`ttl_seconds`	Numeric, time-to-live in seconds (default: 3600 = 1 hour)

Details

Cache Structure: Each cache entry is stored as:

```
list(value = response_text, timestamp = Sys.time())
```

TTL Enforcement: Cache entries are checked on retrieval. If age exceeds TTL, NULL is returned (cache miss). Expired entries are not automatically removed - use `clear()` for manual cleanup.

Thread Safety: Cache is stored in an R environment. Not thread-safe - use separate cache instances per thread/process.

Value

Cache object (environment) with methods:

get(key) Retrieve cached value, or NULL if not found/expired

set(key, value) Store value with current timestamp

clear() Remove all cached entries

stats() Return cache statistics

Examples

```
## Not run:
# Create cache with 1 hour TTL
cache <- bfhllm_cache_create(ttl_seconds = 3600)

# Store value
cache$set("key1", "response text")

# Retrieve value
value <- cache$get("key1")
```

```
# Check stats
stats <- cache$stats()
print(stats$entries) # 1

# Clear cache
cache$clear()

## End(Not run)
```

`bfhllm_cache_shiny` *Create Shiny Session Cache*

Description

Creates a cache object that stores data in Shiny session's userData. Cache is automatically cleaned up when session ends.

Usage

```
bfhllm_cache_shiny(session, ttl_seconds = 3600)
```

Arguments

`session`	Shiny session object
`ttl_seconds`	Numeric, time-to-live in seconds (default: 3600)

Details

Storage Location: Cache is stored in `session$userData$bfhllm_cache` as a reactiveVal. This ensures cache is session-scoped and automatically cleaned on disconnect.

Automatic Cleanup: Registers `session$onSessionEnded()` callback to clear cache when user disconnects, preventing memory leaks.

Idempotent: Safe to call multiple times - will reuse existing cache if already initialized.

Use Cases:

- Shiny applications using BFHllm for AI suggestions
- Reduce API calls and costs within a user session
- Consistent responses for repeated queries

Value

Cache object with same interface as `bfhllm_cache_create()`

Examples

```
## Not run:
# In Shiny server function
server <- function(input, output, session) {
  # Create session cache (call once per session)
  cache <- bfhllm_cache_shiny(session, ttl_seconds = 3600)

  # Use cache with LLM calls
  observeEvent(input$generate_btn, {
    key <- bfhllm_generate_cache_key(
      prompt = input$prompt_text,
      model = "gemini-2.5-flash-lite"
    )

    # Check cache first
    cached <- cache$get(key)
    if (!is.null(cached)) {
      output$result <- renderText(cached)
      return()
    }

    # Make API call
    response <- bfhllm_chat(input$prompt_text)

    # Cache response
    cache$set(key, response)

    output$result <- renderText(response)
  })
}

## End(Not run)
```

---

bfhllm_chat *LLM Chat Interface*

Description

Generic chat function with provider abstraction, caching, circuit breaker protection, and response validation.

Usage

```
bfhllm_chat(
  prompt,
  model = NULL,
  provider = "gemini",
  timeout = NULL,
```

```
  max_chars = NULL,
  cache = NULL,
  validate = TRUE
)
```

Arguments

prompt	Character string, prompt to send to LLM
model	Character string, model identifier (default: from config)
provider	Character string, provider name (default: "gemini")
timeout	Numeric, timeout in seconds (default: from config)
max_chars	Integer, maximum response length (default: from config)
cache	Cache object from bfhllm_cache_create() or bfhllm_cache_shiny() (optional). If provided, responses are cached to reduce API calls.
validate	Logical, whether to validate and sanitize response (default: TRUE)

Details

Features:

- Provider abstraction (currently Gemini, extensible to others)
- Circuit breaker protection (opens after threshold failures)
- Optional caching (reduces API calls and costs)
- Response validation (HTML removal, markdown balancing)
- Timeout handling

Configuration: Default values are read from configuration (set via bfhllm_configure() or environment variables). Function arguments override configuration.

Caching: If cache is provided, responses are cached based on prompt + model hash. Cache hits avoid API calls entirely. Use bfhllm_cache_create() for standalone or bfhllm_cache_shiny() for Shiny apps.

Error Handling: Returns NULL on errors (timeout, API failure, validation failure). Check circuit breaker status with bfhllm_circuit_breaker_status().

Value

Character string with LLM response, or NULL on error

Examples

```
## Not run:
# Basic usage
Sys.setenv(GOOGLE_API_KEY = "your_api_key")
response <- bfhllm_chat("Explain SPC in 2 sentences")
print(response)

# With caching
```

```
cache <- bfhllm_cache_create()
response1 <- bfhllm_chat("What is variation?", cache = cache)
response2 <- bfhllm_chat("What is variation?", cache = cache) # Cache hit

# Custom configuration
response <- bfhllm_chat(
  "Analyze this data pattern",
  model = "gemini-2.0-flash-exp",
  timeout = 15,
  max_chars = 500
)

# In Shiny app
server <- function(input, output, session) {
  cache <- bfhllm_cache_shiny(session)

  observeEvent(input$generate_btn, {
    response <- bfhllm_chat(
      input$prompt,
      cache = cache
    )
    output$result <- renderText(response)
  })
}

## End(Not run)
```

`bfhllm_chat_available` *Check if LLM Chat is Available*

Description

Quick check if chat functionality is available (API key configured, packages installed, etc.)

Usage

```
bfhllm_chat_available()
```

Value

Logical, TRUE if chat is available

Examples

```
## Not run:
if (bfhllm_chat_available()) {
  response <- bfhllm_chat("Hello")
} else {
  message("Chat not available - check API key")
```

```
}

## End(Not run)
```

`bfhllm_chat_with_rag` *RAG-Enhanced Chat*

Description

Performs RAG-enhanced LLM chat by retrieving relevant knowledge, injecting it as context, and calling the LLM. Combines `bfhllm_query_knowledge()` and `bfhllm_chat()`.

Usage

```
bfhllm_chat_with_rag(
  question,
  context = NULL,
  store = NULL,
  top_k = 5,
  method = "hybrid",
  ...
)
```

Arguments

`question`	Character string, the question to answer
`context`	Character string, additional context (optional). Will be combined with RAG-retrieved context.
`store`	ragnar_store object (optional). If NULL, loads from cache/default.
`top_k`	Integer, number of knowledge chunks to retrieve (default: 5)
`method`	Character string, RAG search method: "hybrid", "semantic", "bm25"
`...`	Additional arguments passed to `bfhllm_chat()` (model, timeout, cache, etc.)

Details

Workflow:

1. Query knowledge store for relevant chunks
2. Format retrieved context
3. Build structured prompt with RAG context + user context + question
4. Call LLM via `bfhllm_chat()`

Graceful Degradation: If RAG query fails, proceeds with non-RAG chat (uses only user-provided context). Logs warning but does not fail.

Prompt Structure: Uses `bfhllm_create_structured_prompt()` to organize:

- System: (optional, via ... or config)
- Context: RAG results + user context
- Question: User question
- Format: (optional, via ... or config)

Value

Character string with LLM response, or NULL on error

Examples

```
## Not run:
# Basic RAG-enhanced question
answer <- bfhllm_chat_with_rag("What is a run chart?")

# With additional context
answer <- bfhllm_chat_with_rag(
  question = "Should I use a run chart or SPC chart?",
  context = "Data: 24 observations, stable process"
)

# With chat options
answer <- bfhllm_chat_with_rag(
  question = "Interpret this control chart",
  context = "8 points above centerline",
  model = "gemini-2.5-flash-lite",
  max_chars = 500,
  cache = my_cache
)

## End(Not run)
```

---

bfhllm_circuit_breaker_reset

Reset Circuit Breaker

Description

Manually resets circuit breaker state. Useful for testing or administrative intervention after resolving API issues.

Usage

```
bfhllm_circuit_breaker_reset()
```

Examples

```
## Not run:
# After fixing API connectivity issues
bfhllm_circuit_breaker_reset()

## End(Not run)
```

`bfhllm_circuit_breaker_status` *Get Circuit Breaker Status*

Description

Returns current circuit breaker state for monitoring and debugging.

Usage

```
bfhllm_circuit_breaker_status()
```

Value

List with circuit breaker status:

is_open Logical, TRUE if circuit breaker is open

failures Integer, current failure count

last_failure_time POSIXct, timestamp of last failure (or NULL)

Examples

```
## Not run:
status <- bfhllm_circuit_breaker_status()
if (status$is_open) {
  message("Circuit breaker is open - API calls blocked")
}

## End(Not run)
```

`bfhllm_configure` *Configure BFHllm Settings*

Description

Set runtime configuration for LLM provider, model, and behavior. Configuration is stored in package environment and persists for the session.

Usage

```
bfhllm_configure(
  provider = NULL,
  model = NULL,
  timeout_seconds = NULL,
  max_response_chars = NULL,
  circuit_breaker = NULL,
  cache = NULL,
  rate_limit = NULL
)
```

Arguments

`provider`	Character string, LLM provider name (default: "gemini")
`model`	Character string, model identifier (default: "gemini-2.5-flash-lite")
`timeout_seconds`	Numeric, API timeout in seconds (default: 10)
`max_response_chars`	Integer, maximum response length (default: 350)
`circuit_breaker`	List with circuit breaker settings (optional)
`cache`	List with cache settings (optional)
`rate_limit`	List with rate limit settings (optional): **enabled** Logical, enable/disable rate limiting (default: TRUE) **rpm** Integer, max requests per minute (default: 15, Free Tier) **rpd** Integer, max requests per day (default: 1500, Free Tier) **behavior** Character, "wait" (sleep until capacity) or "degrade" (return fallback message). Default: "wait"

Value

Invisibly returns the updated configuration list

Examples

```
## Not run:
# Basic configuration
bfhllm_configure(
  model = "gemini-2.0-flash-exp",
  timeout_seconds = 15
)

# Advanced configuration
bfhllm_configure(
  circuit_breaker = list(
    enabled = TRUE,
    failure_threshold = 3,
    reset_timeout_seconds = 180
  ),
  cache = list(
    enabled = TRUE,
    ttl_seconds = 7200
  )
)

## End(Not run)
```

`bfhllm_create_structured_prompt` *Create Structured Prompt*

Description

Creates a prompt with structured sections (system, context, question). Useful for RAG-enhanced prompts or complex multi-part queries.

Usage

```
bfhllm_create_structured_prompt(
  question,
  context = NULL,
  system = NULL,
  format = NULL
)
```

Arguments

`question`	Character string, main question or instruction
`context`	Character string, additional context (optional)
`system`	Character string, system message or role (optional)
`format`	Character string, desired output format instructions (optional)

Value

Character string with structured prompt

Examples

```
## Not run:
prompt <- bfhllm_create_structured_prompt(
  question = "What causes special cause variation?",
  context = "Run chart shows 8 consecutive points above centerline",
  system = "You are an SPC methodology expert",
  format = "Respond in Danish, max 2 sentences"
)

## End(Not run)
```

---

bfhllm_extract_spc_metadata

Extract SPC Metadata for AI Prompts

Description

Extracts structured SPC metadata from BFHcharts/qicharts2 result objects for use in AI prompt generation.

Usage

```
bfhllm_extract_spc_metadata(spc_result)
```

Arguments

`spc_result`	List from BFHcharts or qicharts2 with components: • metadata: list with chart_type, n_points, signals_detected, anhoej_rules • qic_data: data.frame with x, y, cl, ucl, lcl columns

Value

Named list with extracted metadata:

- chart_type: Chart type code
- chart_type_dansk: Danish name
- n_points: Number of observations
- signals_detected: Anhøj rule violations
- longest_run: Longest run above/below centerline
- n_crossings: Centerline crossings
- n_crossings_min: Expected minimum crossings

- centerline: Mean centerline value
- start_date: First x value
- end_date: Last x value
- process_variation: "naturligt" or "ikke naturligt"

Returns NULL if spc_result is invalid or missing required components.

Examples

```
## Not run:
# With BFHcharts result
spc_result <- BFHcharts::create_spc_chart(data, ...)
metadata <- bfhllm_extract_spc_metadata(spc_result)

# With qicharts2 result
qic_result <- qicharts2::qic(x, y, chart = "run", return.data = TRUE)
metadata <- bfhllm_extract_spc_metadata(qic_result)

## End(Not run)
```

---

bfhllm_file_cache_create

Opret fil-baseret cache

Description

Opretter en cache der gemmer data i en .rds-fil paa disk. Cachen overlever R-sessions og kan deles mellem koersler.

Usage

```
bfhllm_file_cache_create(cache_dir)
```

Arguments

cache_dir	Character, sti til cache-mappe

Details

Cachen bruger en enkelt RDS index-fil til at gemme alle entries. Filen placeres i en `.bfhllm_cache` undermappe af `cache_dir`.

Fordi data gemmes paa disk, overlever cachen R-sessions. Designet til single-process brug – ikke egnet til concurrent adgang.

Value

Liste med get/set/has/clear/stats funktioner:

get(key) Hent vaerdi fra cache, eller NULL hvis ikke fundet

set(key, value) Gem vaerdi i cache

has(key) Returnerer TRUE/FALSE om key eksisterer

clear() Fjern alle cache-entries

stats() Returnerer cache-statistik

Examples

```
## Not run:
# Opret cache i midlertidig mappe
cache <- bfhllm_file_cache_create(cache_dir = tempdir())

# Gem og hent vaerdi
cache$set("key1", "analyse tekst")
cache$get("key1")  # "analyse tekst"

# Tjek om key eksisterer
cache$has("key1")  # TRUE

# Statistik
cache$stats()  # list(entries = 1L, cache_dir = "...")

# Ryd cache
cache$clear()

## End(Not run)
```

`bfhllm_format_rag_context`

Format RAG Context for Prompt

Description

Formats RAG search results into a structured context string suitable for inclusion in LLM prompts. Combines retrieved chunks with metadata.

Usage

```
bfhllm_format_rag_context(results, max_chunks = 5, include_scores = FALSE)
```

Arguments

`results`	Data frame from `bfhllm_query_knowledge()`, or NULL
`max_chunks`	Integer, maximum chunks to include (default: 5)
`include_scores`	Logical, include similarity scores in output (default: FALSE)

Details

Output Format:

```
Context from knowledge base:

[1] <chunk text>
[2] <chunk text>
...
```

NULL Handling: Returns NULL if results is NULL or empty. Calling code should handle this gracefully (e.g., proceed without RAG context).

Value

Character string with formatted context, or NULL if no results

Examples

```
## Not run:
# Query and format
results <- bfhllm_query_knowledge("run chart")
context <- bfhllm_format_rag_context(results)

# Include scores for debugging
context <- bfhllm_format_rag_context(results, include_scores = TRUE)

# Limit chunks
context <- bfhllm_format_rag_context(results, max_chunks = 3)

## End(Not run)
```

---

bfhllm_generate_cache_key

Generate Cache Key

Description

Creates deterministic hash from input data using xxhash64 algorithm. Ensures same inputs always produce same cache key.

Usage

```
bfhllm_generate_cache_key(...)
```

Arguments

`...`	Named arguments to include in cache key

Details

Usage: Pass all relevant parameters that should distinguish cached responses. Exclude volatile data (timestamps, random values, reactive triggers).

Hash Algorithm: Uses xxhash64 via digest package - fast and collision-resistant.

Value

Character string (hex hash)

Examples

```
## Not run:
# Basic key generation
key <- bfhllm_generate_cache_key(
  prompt = "What is SPC?",
  model = "gemini-2.5-flash-lite"
)

# With metadata
key <- bfhllm_generate_cache_key(
  prompt = prompt_text,
  metadata = list(chart_type = "run", n_points = 24),
  context = list(title = "Chart Title")
)

## End(Not run)
```

---

bfhllm_get_config *Get BFHllm Configuration*

Description

Retrieve current configuration with environment variable fallbacks.

Usage

```
bfhllm_get_config()
```

Value

List with current configuration settings

Examples

```
## Not run:
config <- bfhllm_get_config()
print(config$model)

## End(Not run)
```

`bfhllm_interpolate` *Interpolate Prompt Template*

Description

Replaces placeholders in prompt template with values from data list. Placeholders use `{{variable}}` syntax (double curly braces).

Usage

```
bfhllm_interpolate(template, data)
```

Arguments

`template`	Character string, prompt template with `{{placeholders}}`
`data`	Named list with values to interpolate

Details

Placeholder Syntax: Use `{{variable_name}}` in template. Spaces inside braces are allowed (`{{ variable }}` works).

Missing Variables: If a placeholder variable is not found in data, it is left unchanged in the output (no error thrown).

Type Coercion: All values are coerced to character strings via `as.character()`.

Value

Character string with interpolated prompt

Examples

```
## Not run:
# Basic interpolation
template <- "Hello {{name}}, you are {{age}} years old"
data <- list(name = "Alice", age = 30)
prompt <- bfhllm_interpolate(template, data)
# "Hello Alice, you are 30 years old"

# SPC prompt template
template <- paste(
```

```
  "Analyze this {{chart_type}} chart with {{n_points}} data points.",
  "The process shows {{variation_type}} variation.",
  "Suggest improvements."
)
data <- list(
  chart_type = "run chart",
  n_points = 24,
  variation_type = "stable"
)
prompt <- bfhllm_interpolate(template, data)

## End(Not run)
```

`bfhllm_load_knowledge_store` *Load Ragnar Knowledge Store*

Description

Loads the pre-built Ragnar knowledge store. Store is loaded once per session and cached for performance. Automatically detects development vs production mode and adjusts paths accordingly.

Usage

```
bfhllm_load_knowledge_store(store_path = NULL)
```

Arguments

`store_path`	Character string, path to store (optional). If NULL, auto-detects based on package installation status.

Details

Path Detection:

- Production: `system.file("ragnar_store", package = "BFHllm")`
- Development: `inst/ragnar_store` (relative to package root)

API Key Setup: Ragnar requires `GEMINI_API_KEY`. If not set, automatically falls back to `GOOGLE_API_KEY` if available.

Caching: Store is cached in package environment after first successful load. Subsequent calls return cached store (fast).

Error Handling: Returns NULL on errors (store not found, ragnar not installed, load failure). Check return value before querying.

Value

ragnar_store object, or NULL if store not found/cannot be loaded

Examples

```
## Not run:
# Load store (auto-detect path)
store <- bfhllm_load_knowledge_store()

if (!is.null(store)) {
  # Query store
  results <- bfhllm_query_knowledge("run chart", store)
}

# Force reload (clears cache)
bfhllm_reset_knowledge_store_cache()
store <- bfhllm_load_knowledge_store()

## End(Not run)
```

`bfhllm_map_chart_type_danish` *Map Chart Type to Danish Name*

Description

Translates English SPC chart type codes to Danish display names.

Usage

```
bfhllm_map_chart_type_danish(chart_type)
```

Arguments

`chart_type`	Character string, chart type code (e.g., "run", "p", "c")

Details

Supported Chart Types:

- run: Serieplot (run chart)
- i: I-chart (individuelle værdier)
- mr: MR-chart (moving range)
- xbar: X-bar chart (gennemsnit)
- s: S-chart (standardafvigelse)
- t: T-chart (tid mellem events)
- p: P-chart (andel)
- pp: PP-chart (andel per periode)
- c: C-chart (antal events)

- u: U-chart (rate per enhed)
- g: G-chart (events mellem)
- prime: Prime chart

Unknown Types: Returns original English name if chart type not recognized.

Value

Character string with Danish chart type name

Examples

```
bfhllm_map_chart_type_danish("run")
# "serieplot (run chart)"

bfhllm_map_chart_type_danish("p")
# "P-chart (andel)"

bfhllm_map_chart_type_danish("unknown")
# "unknown"
```

---

bfhllm_query_knowledge

Query Knowledge Store

Description

Queries the Ragnar knowledge store for relevant information. Supports both semantic search (embeddings) and keyword search (BM25), or hybrid mode combining both.

Usage

```
bfhllm_query_knowledge(query, store = NULL, top_k = 5, method = "hybrid")
```

Arguments

query	Character string, the query to search for
store	ragnar_store object (optional). If NULL, loads from cache/default.
top_k	Integer, number of top results to return (default: 5)
method	Character string, search method: "hybrid" (default), "semantic", or "bm25"

Details

Search Methods:

- "hybrid": Combines semantic + BM25 (best for most queries)
- "semantic": Vector similarity search via embeddings
- "bm25": Keyword-based search (fast, good for exact terms)

Store Loading: If store is NULL, automatically loads via `bfhllm_load_knowledge_store()`. Store is cached after first load for performance.

Error Handling: Returns NULL if store unavailable or query fails. Check return value before using results.

Value

Data frame with columns: chunk_id, text, score, metadata, or NULL on error

Examples

```
## Not run:
# Query with default hybrid search
results <- bfhllm_query_knowledge("run chart interpretation")

# Query with semantic search only
results <- bfhllm_query_knowledge(
  "special cause variation",
  method = "semantic",
  top_k = 3
)

# Use existing store
store <- bfhllm_load_knowledge_store()
results <- bfhllm_query_knowledge("control limits", store = store)

## End(Not run)
```

---

bfhllm_rate_limit_reset

Reset Rate Limiter

Description

Manually resets all rate limiter state. Useful for testing or after resolving rate limit issues.

Usage

```
bfhllm_rate_limit_reset()
```

Examples

```
## Not run:
bfhllm_rate_limit_reset()

## End(Not run)
```

`bfhllm_rate_limit_status` *Get Rate Limit Status*

Description

Returns current rate limit usage, remaining capacity, and configuration.

Usage

```
bfhllm_rate_limit_status()
```

Value

List with rate limit status:

requests_this_minute Integer, requests in current sliding window

requests_today Integer, requests since midnight

rpm_limit Integer, configured RPM limit

rpd_limit Integer, configured RPD limit

rpm_remaining Integer, remaining RPM capacity

rpd_remaining Integer, remaining RPD capacity

daily_reset_date Date, when daily counter was last reset

enabled Logical, whether rate limiting is active

behavior Character, "wait" or "degrade"

Examples

```
## Not run:
status <- bfhllm_rate_limit_status()
message(sprintf("RPM: %d/%d, RPD: %d/%d",
  status$requests_this_minute, status$rpm_limit,
  status$requests_today, status$rpd_limit))

## End(Not run)
```

bfhllm_reset_knowledge_store_cache *Reset Knowledge Store Cache*

Description

Clears cached store and load attempt flag. Use for testing or forcing reload after store rebuild.

Usage

```
bfhllm_reset_knowledge_store_cache()
```

Value

invisible(NULL)

Examples

```
## Not run:
# Force reload after rebuilding store
bfhllm_reset_knowledge_store_cache()
store <- bfhllm_load_knowledge_store()

## End(Not run)
```

bfhllm_spc_suggestion *Generate SPC Improvement Suggestion*

Description

Main function for generating AI-powered improvement suggestions for SPC charts. Combines metadata extraction, RAG context retrieval, prompt building, and LLM generation into a single high-level interface.

Usage

```
bfhllm_spc_suggestion(
  spc_result,
  context,
  min_chars = 300,
  max_chars = 375,
  use_rag = TRUE,
  cache = NULL,
  ...
)
```

Arguments

spc_result	List from BFHcharts/qicharts2 with metadata and qic_data
context	Named list with user context: • data_definition: Indicator description (character) • chart_title: Chart title (character) • y_axis_unit: Unit of measurement (e.g., "dage", "antal", "procent") • target_value: Target value (numeric, optional)
min_chars	Minimum characters in response (default: 300)
max_chars	Maximum characters in response (default: 375)
use_rag	Logical, use RAG for SPC methodology context (default: TRUE)
cache	Cache object from bfhllm_cache_create() or bfhllm_cache_shiny() (optional)
...	Additional arguments passed to bfhllm_chat() (model, timeout, etc.)

Details

Workflow:

1. Extract SPC metadata from result object
2. Query RAG knowledge store (if use_rag = TRUE)
3. Build structured prompt with metadata + context + RAG
4. Call LLM via bfhllm_chat()
5. Validate and return response

Caching: If cache provided, checks cache before API call and stores result after. Cache key includes metadata, context, and RAG content for uniqueness.

RAG Integration: When use_rag = TRUE, queries knowledge store for relevant SPC methodology based on chart type, signals detected, and target comparison.

Value

Character string with AI-generated improvement suggestion in Danish, or NULL on error

Examples

```
## Not run:
# Basic usage
spc_result <- BFHcharts::create_spc_chart(data, ...)
context <- list(
  data_definition = "Ventetid til operation",
  chart_title = "Ventetid ortopædkirurgi 2024",
  y_axis_unit = "dage",
  target_value = 30
)

suggestion <- bfhllm_spc_suggestion(spc_result, context)
```

```
# With caching (Shiny)
cache <- bfhllm_cache_shiny(session)
suggestion <- bfhllm_spc_suggestion(
  spc_result, context,
  cache = cache
)

# Without RAG
suggestion <- bfhllm_spc_suggestion(
  spc_result, context,
  use_rag = FALSE
)

## End(Not run)
```

`bfhllm_spc_suggestions_batch` *Batch SPC Analyse for Flere Diagrammer*

Description

Genererer AI-drevne analyser for flere SPC-diagrammer i batch. Bruger fil-baseret cache til at undgaa gentagne API-kald og respekterer rate limits.

Usage

```
bfhllm_spc_suggestions_batch(
  contexts,
  batch_size = 25L,
  use_cache = TRUE,
  cache_dir = NULL,
  force_refresh = FALSE,
  min_chars = 300,
  max_chars = 375
)
```

Arguments

`contexts`	Named list: diagram_key -> list(spc_result, llm_context). Hver kontekst indeholder: **spc_result** Liste med metadata (chart_type, n_points, signals_detected, anhoej_rules) **llm_context** Liste med data_definition, chart_title, y_axis_unit, target_value, signal_examples
`batch_size`	Integer, antal diagrammer per API-kald (default: 25)
`use_cache`	Logical, brug fil-baseret cache (default: TRUE)
`cache_dir`	Character, sti til cache-mappe (default: tempdir())

force_refresh	Logical, ignorer cache og kald API igen (default: FALSE)
min_chars	Integer, minimum tegn per analyse (default: 300)
max_chars	Integer, maksimum tegn per analyse (default: 375)

Value

Liste med:

analyses Named list: diagram_key -> analyse tekst

from_cache Integer, antal analyser hentet fra cache

from_api Integer, antal analyser genereret via API

failed Character vector, diagram_keys der fejlede

rpd_exhausted Logical, TRUE hvis daglig rate limit var opbrugt

Examples

```
## Not run:
# Opret kontekster for 3 diagrammer
contexts <- list(
  diagram_1 = list(
   spc_result = list(metadata = list(chart_type = "run", n_points = 24, signals_detected = 0)),
    llm_context = list(data_definition = "Ventetid", chart_title = "Ventetid 2024",
                       y_axis_unit = "dage", target_value = 30)
  ),
  diagram_2 = list(
   spc_result = list(metadata = list(chart_type = "p", n_points = 12, signals_detected = 2)),
   llm_context = list(data_definition = "Infektionsrate", chart_title = "Infektioner Q1",
                       y_axis_unit = "procent", target_value = 5)
  )
)

result <- bfhllm_spc_suggestions_batch(contexts)
print(result$analyses)

## End(Not run)
```

---

bfhllm_validate_setup *Validate BFHllm Setup*

Description

Checks if all prerequisites for using BFHllm are met:

- Required packages installed (ellmer)
- API key configured
- Configuration valid

Usage

```
bfhllm_validate_setup()
```

Value

Logical, TRUE if setup is valid, FALSE otherwise (with warnings)

Examples

```
## Not run:
if (bfhllm_validate_setup()) {
  # Proceed with LLM operations
} else {
  # Handle setup issues
}

## End(Not run)
```

---

list_providers *List Available Providers*

Description

Returns names of all registered LLM providers.

Usage

```
list_providers()
```

Value

Character vector of provider names

Examples

```
## Not run:
providers <- list_providers()
print(providers) # "gemini"

## End(Not run)
```

print.bfhllm_cache *Print Cache Object*

Description

Print Cache Object

Usage

```
## S3 method for class 'bfhllm_cache'
print(x, ...)
```

Arguments

x	Cache object
...	Unused

print.bfhllm_cache_shiny *Print Shiny Cache Object*

Description

Print Shiny Cache Object

Usage

```
## S3 method for class 'bfhllm_cache_shiny'
print(x, ...)
```

Arguments

x	Shiny cache object
...	Unused

`validate_response` *Validate and Sanitize LLM Response*

Description

Validates API response text and sanitizes it by:

- Checking for NULL or empty responses
- Removing HTML tags
- Normalizing whitespace
- Trimming to maximum character limit
- Balancing markdown formatting

Usage

```
validate_response(text, max_chars = 350)
```

Arguments

`text`	Character string, raw response from LLM
`max_chars`	Integer, maximum allowed characters (default: 350)

Details

Sanitization Steps:

1. Check for NULL/empty input
2. Remove HTML tags
3. Normalize whitespace (collapse multiple spaces)
4. Trim to max_chars (preserving word boundaries)
5. Balance markdown asterisks (ensure even count)

Character Limit: When text exceeds max_chars, it is truncated at the last complete word before the limit, and "..." is appended. Markdown formatting is balanced to avoid broken bold/italic markers.

Value

Character string with sanitized text, or NULL if invalid

Examples

```
## Not run:
# Valid response
text <- validate_response(
  "This is a **valid** response",
  max_chars = 100
)

# Long response (will be truncated)
long_text <- paste(rep("word", 100), collapse = " ")
text <- validate_response(long_text, max_chars = 50)

## End(Not run)
```

`validate_response_length` *Validate Response Length*

Description

Simple check if response is within character limit.

Usage

```
validate_response_length(text, max_chars = 350)
```

Arguments

`text`	Character string
`max_chars`	Integer, maximum allowed characters

Value

Logical, TRUE if within limit

Examples

```
## Not run:
validate_response_length("Short text", max_chars = 100) # TRUE
validate_response_length(paste(rep("word", 100), collapse = " "), max_chars = 50) # FALSE

## End(Not run)
```

Index