Project: A Comprehensive Technical Overview and Implementation Guide Document Version: 5.2 Date: November 4, 2025 Author: Nehemia

Revision History

- **Version 5.2 (November 4, 2025) - Nehemia**
  - **Integrated NLP Intent Classification:** The system now loads a pre-trained Multinomial Naive Bayes classifier (*classifier.model*) on startup. This allows *index.php* to predict user intent (e.g., "feedback_positive") for more nuanced interaction handling.
  - **Enhanced Text Preprocessing:** Replaced the basic *extractEntities* logic in *MemoryManager.php* with a robust NLP pipeline using the *nlp-tools* library. This incorporates tokenization, stemming, and configurable stop word removal for more accurate keyword extraction.
  - **Added Short-Term Memory:** Implemented a *FORCED_RECENT_INTERACTIONS* feature in *config.php*. This new mechanism in *MemoryManager.php* ensures the AI always recalls the last few conversational turns, improving conversational flow and context retention.
  - **New Training Script:** Introduced *train.php*, a dedicated script to build and serialize the classification model from a *training_data.csv* file, enabling customizable intent training.
  - **UI Enhancements:** The user interface in *index.php* has been updated to conditionally display the predicted intent and includes a new form for users to submit specific text feedback on AI responses.
  - **Refined Configuration:** Added *NLP_STOP_WORDS* and *FORCED_RECENT_INTERACTIONS* constants to *config.php* for centralized control over the new NLP and short-term memory features.
- **Version 5.1 (October 13, 2025) - Nehemia**
  - Enhanced Error Handling: Overhauled the *generateResponse* method in *GeminiClient.php* to provide more robust and user-friendly error handling.
  - Specific API Error Parsing: The client now attempts to parse the JSON error response from the Gemini API to log specific error details (e.g., INVALID_ARGUMENT, PERMISSION_DENIED).
  - User-Friendly Error Messages: The system now returns a generic, user-friendly error message to the UI to avoid exposing raw technical details, while logging the full error for debugging purposes.
  - Refined Exception Handling: Improved the logic for handling different HTTP status codes and cURL failures, ensuring a clear distinction between internal application errors and external API failures.
- **Version 5.0 (October 12, 2025) - Nehemia**

- Major Architectural Update: Transitioned from a purely keyword-based retrieval system to a Hybrid Search model, combining lexical (keyword) and semantic (vector) search for significantly improved context relevance.
- Decoupled Embedding Logic: Introduced a new, separate *EmbeddingClient.php* to handle all vector embedding generation. This modular design allows for future swapping of embedding providers (e.g., to a local model) without altering core logic.
- Overhauled Memory Retrieval: The *getRelevantContext()* method in *MemoryManager.php* was completely rewritten to perform the hybrid fusion of search results.
- Semantic Memory Encoding: The *updateMemory()* method now automatically generates and stores a vector embedding for every new interaction.
- Enhanced Configuration: Updated *config.php* with a master switch (*ENABLE_EMBEDDINGS*) to enable or disable the vector system, and tuning parameters (*HYBRID_SEARCH_ALPHA*, *VECTOR_SEARCH_TOP_K*) to control the search behavior.
- UI Update: Version badge in *index.php* updated to v5.0.

1. Executive Summary

This document provides a comprehensive and multi-layered overview of "Project NEMI," an advanced AI initiative focused on developing a dynamic, persistent, and self-organizing memory system. The project's architecture has evolved significantly from a simple data logger into a sophisticated, reinforcement-based knowledge graph that leverages a Retrieval-Augmented Generation (RAG) framework.

This guide details the project's conceptual evolution, from its initial goals to its current state, and provides a complete technical specification for version 5.2. This version introduces a resilient Hybrid Search capability, fuses traditional keyword retrieval with modern semantic vector search, and integrates a new Natural Language Processing (NLP) pipeline for intent classification. The system is designed to mimic key aspects of human memory, including a new short-term memory feature, the ability to prioritize relevant information, establish connections between concepts, and gradually "forget" outdated data, thereby providing a robust foundation for a memory-driven AGI.

2. Architectural Evolution

The architecture of Project NEMI has progressed through several key versions, with each stage addressing the limitations of the previous one.

2.1. V0.0: Foundational Concept - The Simple Log

The project began with a straightforward objective: to create a persistent memory by logging all user and AI interactions in a structured JSON format. This initial version suffered from critical limitations in scalability, data retrieval, and relational context.

2.2. V2.0: A Multi-Layered Memory Architecture

The architecture was redesigned to function more like a relational database, separating concerns into four distinct components: Sessions, Interactions, Entities (The Knowledge Graph), and Users. This transformed the memory from a linear log into a rich, interconnected graph of knowledge.

2.3. V3.0: Reinforcement-Based Memory and RAG

This version introduced the crucial concept of memory relevance, turning the static knowledge graph into a dynamic system that learns what information is important. It established core mechanisms for relevance scoring, reinforcement (reward), forgetting (decay), and Retrieval-Augmented Generation (RAG).

2.4. V4.0: Practical Implementation and the "Brain Analogy"

This version marked the transition to a tangible, functional PHP application. The "Brain Analogy" architectural pattern was introduced, promoting a clear separation of concerns into Orchestration (*index.php*), Memory System (*MemoryManager.php*), Communication (*GeminiClient.php*), and Rules & Personality (*config.php*).

2.5. V5.0: Hybrid Search and Decoupled Architecture

This version represented a major leap in retrieval intelligence, moving beyond purely lexical search to a hybrid model that combines the precision of keyword search with the conceptual breadth of semantic search using vector embeddings. The embedding logic was also decoupled into a dedicated *EmbeddingClient.php* for modularity.

3. Technical Specification and Implementation: Version 5.2

Version 5.2 is a mature PHP implementation of the Hybrid Search RAG architecture, now enhanced with an NLP-powered intent classification and a more robust memory model.

3.1. The "Brain Analogy": System Components

The project is structured into modular components, each with a distinct role:

- ***index.php*** **(Conscious Thought):** The main orchestrator. It handles user requests, coordinates with memory and AI clients, builds the final prompt, and renders the UI. In v5.2, it also loads and utilizes the intent classification model.
- ***MemoryManager.php*** **(The Memory System):** The core brain. It manages all memory operations, including loading, saving, and the complex hybrid retrieval of relevant context. It now features a more advanced NLP pipeline for entity extraction and a dedicated short-term memory mechanism.
- ***GeminiClient.php*** **(Senses & Voice):** The dedicated communicator for generating content. It handles all conversational interactions with the Google Gemini API.

- ***EmbeddingClient.php* (The Conceptual Librarian):** A dedicated communicator for generating embeddings. It translates text into its semantic meaning (vectors) and is decoupled from the *GeminiClient*.
- ***train.php* (The Trainer):** A new component responsible for offline training. It reads a dataset of labeled examples and produces the serialized classifier models used by *index.php*.
- ***config.php* (Personality & Rules):** A centralized configuration file defining API keys, file paths, and the constants that govern memory, hybrid search, and the new NLP logic.

3.2. Key Features and Enhancements in V5.2

- **NLP-Powered Intent Classification:** The system can now predict the user's intent from their input (e.g., identifying positive or negative feedback) using a pre-trained model, allowing for smarter, more dynamic responses.
- **Short-Term Conversational Memory:** A new *FORCED_RECENT_INTERACTIONS* setting ensures the last few turns of a conversation are always included in the context, dramatically improving the AI's ability to follow immediate conversational flow.
- **Advanced NLP for Entity Extraction:** The old keyword extraction method has been replaced with a sophisticated NLP pipeline (tokenization, stop word removal, stemming) from the *nlp-tools* library, leading to more accurate context retrieval.
- **Modular Training Process:** The introduction of *train.php* separates the model training process from the main application runtime, adhering to best practices.
- **Enhanced User Feedback Loop:** The UI now not only allows for simple "good/bad" feedback but also includes a text field for more detailed user corrections and displays the AI's predicted intent for transparency.

4. Complete Source Code (Version 5.2)

```php
<?php

// --- API Configuration ---
// The secret key required to authenticate with the Google Gemini API.
// CRITICAL: Replace 'YOUR_GEMINI_API_KEY_HERE' with your actual key.
define('GEMINI_API_KEY', 'AIzaSyBEkzJRNr-CvwqVCJQtcYs3bb2M-Ikq0pA');

// Specifies the exact generative model to use for creating responses.
// 'gemini-1.5-flash-latest' is a fast and capable model suitable for chat.
define('MODEL_ID', 'gemini-2.5-flash-lite');

// The specific Gemini API function to call. 'generateContent' is the standar
d
// endpoint for getting a complete response from the model.
define('API_ENDPOINT', 'generateContent');


// --- Embedding Configuration ---
// A master switch to turn the entire vector/semantic search system on or off
.
```

```php
// Set to 'false' to revert to a purely keyword-based search.
define('ENABLE_EMBEDDINGS', true);

// The specialized model used for converting text into numerical vectors (emb
eddings).
// 'text-embedding-004' is optimized for this task and is highly efficient.
define('EMBEDDING_MODEL_ID', 'text-embedding-004');


// --- NLP Configuration ---
define('NLP_STOP_WORDS', [
    'a', 'an', 'the', 'is', 'in', 'it', 'of', 'for', 'on', 'what', 'were',
    'my', 'that', 'we', 'to', 'user', 'note', 'system', 'please', 'and'
    // Add other words as needed
]);


// --- File Paths ---
// Defines the directory where all persistent data (memory files) will be sto
red.
define('DATA_DIR', __DIR__ . '/data');

// The filename for the JSON file that stores all conversation history.
define('INTERACTIONS_FILE', DATA_DIR . '/interactions.json');

// The filename for the JSON file that acts as the knowledge graph, storing
// information about concepts and keywords.
define('ENTITIES_FILE', DATA_DIR . '/entities.json');

// The filename for the log that records every prompt sent to the Gemini API.
// This is used for debugging and performance analysis.
define('PROMPTS_LOG_FILE', DATA_DIR . '/prompts.json');


// --- Memory Logic Configuration ---
// The amount to increase a memory's 'relevance_score' when it is successfull
y
// used as context for a good response. Higher values make the AI "learn" fas
ter.
define('REWARD_SCORE', 0.5);

// The base amount to decrease a memory's 'relevance_score' during each new
// interaction. This simulates "forgetting" and prevents old, unused memories
from cluttering the system.
define('DECAY_SCORE', 0.05);

// The starting 'relevance_score' for any new memory created.
define('INITIAL_SCORE', 1.0);

// The maximum number of interactions to keep in memory. When this number is
```

```
// exceeded, the system will delete the interactions with the lowest relevanc
e scores.
define('PRUNING_THRESHOLD', 500);

// The maximum number of tokens to include in the context sent to the AI.
// This prevents the prompt from becoming too large and expensive.
define('CONTEXT_TOKEN_BUDGET', 4000);

// --- NEW: Short-Term Memory Configuration ---
// The number of most recent interactions to ALWAYS include in the context.
// Set to 1 to guarantee the AI remembers the very last thing said.
// Set to 2 or 3 to give it a slightly better conversational short-term memor
y.
// Set to 0 to disable and rely purely on the hybrid search.
define('FORCED_RECENT_INTERACTIONS', 2);


// --- Hybrid Search Tuning ---
// The core dial that balances keyword search against vector search.
// 0.0 = 100% keyword-based. The AI will only find exact matches.
// 1.0 = 100% semantic-based. The AI will only find conceptually similar idea
s.
// 0.5 = A balanced mix, providing both precision and conceptual relevance.
define('HYBRID_SEARCH_ALPHA', 0.5);

// The number of top results to fetch from the semantic (vector) search stage
.
// A higher number allows the fusion algorithm to consider more conceptually
related
// memories, but may slightly slow down the retrieval process.
define('VECTOR_SEARCH_TOP_K', 15);


// --- Advanced Scoring & Relationships ---
// An extra 'relevance_score' bonus given to a new memory if it contains a
// keyword (entity) the AI has never seen before. Encourages learning new top
ics.
define('NOVELTY_BONUS', 0.3);

// The amount to increase the strength of the connection between two keywords
// every time they appear in the same user prompt.
define('RELATIONSHIP_STRENGTH_INCREMENT', 0.1);

// A multiplier that reduces the 'DECAY_SCORE' for memories related to the
// current conversation topic. This helps the AI "stay on topic" by forgettin
g
// relevant memories more slowly.
define('RECENT_TOPIC_DECAY_MODIFIER', 0.1);

// The amount to increase the 'relevance_score' of an interaction and its
```

```
// context when the user clicks the "Good" feedback button.
define('USER_FEEDBACK_REWARD', 0.5);

// The amount to decrease the 'relevance_score' of an interaction and its
// context when the user clicks the "Bad" feedback button.
define('USER_FEEDBACK_PENALTY', -0.5);

<?php

/**
 * Handles the generation of vector embeddings via the Gemini API.
 * This class is decoupled from the main content generation client
 * to allow for easy swapping with other embedding providers (e.g., local Oll
ama).
 */
class EmbeddingClient
{
    private string $apiKey;
    private string $modelId;
    private string $apiUrl;

    public function __construct()
    {
        $this->apiKey = GEMINI_API_KEY;
        $this->modelId = EMBEDDING_MODEL_ID;
        $this->apiUrl = "https://generativelanguage.googleapis.com/v1beta/mod
els/{$this->modelId}:embedContent?key={$this->apiKey}";
    }

    /**
     * Converts a string of text into a vector embedding.
     *
     * @param string $text The text to embed.
     * @return array|null The vector as an array of floats, or null on error.
     */
    public function getEmbedding(string $text): ?array
    {
        if (!ENABLE_EMBEDDINGS) {
            return null; // Return null if embeddings are disabled in config
        }

        try {
            $requestBody = [
                'model'   => "models/{$this->modelId}",
                'content' => ['parts' => [['text' => $text]]]
            ];

            $ch = curl_init();
            curl_setopt($ch, CURLOPT_URL, $this->apiUrl);
            curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
```

```
            curl_setopt($ch, CURLOPT_POST, true);
            curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($requestBody));
            curl_setopt($ch, CURLOPT_HTTPHEADER, ['Content-Type: application/
json']);

            $rawResponse = curl_exec($ch);
            if (curl_errno($ch)) {
                throw new Exception("cURL Error getting embedding: " . curl_e
rror($ch));
            }
            curl_close($ch);

            $decodedResponse = json_decode($rawResponse, true);
            $embedding = $decodedResponse['embedding']['values'] ?? null;

            if ($embedding === null) {
                // Log the actual response for debugging
                error_log("Failed to find embedding in API response: " . $raw
Response);
                throw new Exception("Could not find embedding in API response
.");
            }

            return $embedding;
        } catch (Exception $e) {
            error_log($e->getMessage());
            return null;
        }
    }
}
```

```
<?php

class GeminiClient
{
    private string $apiKey;
    private string $modelId;
    private string $apiUrl;

    public function __construct()
    {
        $this->apiKey = GEMINI_API_KEY;
        $this->modelId = MODEL_ID;
        $this->apiUrl = "https://generativelanguage.googleapis.com/v1beta/mod
els/{$this->modelId}:" . API_ENDPOINT . "?key={$this->apiKey}";
    }

    private function logPrompt(string $prompt, array $requestBody, string $ra
wResponse, ?string $error = null): void
    {
```

```php
        if (!file_exists(PROMPTS_LOG_FILE)) {
            file_put_contents(PROMPTS_LOG_FILE, '[]');
        }
        $logData = json_decode(file_get_contents(PROMPTS_LOG_FILE), true);
        $decodedResponse = json_decode($rawResponse, true);

        $logEntry = [
            'timestamp' => date('c'),
            'request' => [
                'model' => $this->modelId,
                'tools' => array_keys(array_column($requestBody['tools'] ?? [
], null, 0)),
                'prompt_text' => $prompt
            ],
            'response' => [
                'token_usage' => $decodedResponse['usageMetadata'] ?? null,
                'finish_reason' => $decodedResponse['candidates'][0]['finishR
eason'] ?? null,
                'response_text' => $decodedResponse['candidates'][0]['content
']['parts'][0]['text'] ?? null,
            ],
            'error' => $error
        ];

        $logData[] = $logEntry;
        file_put_contents(PROMPTS_LOG_FILE, json_encode($logData, JSON_PRETTY
_PRINT | JSON_UNESCAPED_SLASHES));
    }

    public function generateResponse(string $prompt, array $enabledTools = []
): string
    {
        if (empty($this->apiKey) || $this->apiKey === 'YOUR_GEMINI_API_KEY_HE
RE') {
            return "ERROR: Gemini API Key is not configured in config.php. Pl
ease set a valid key to proceed.";
        }

        $rawResponse = '';
        $requestBody = [];

        try {
            $requestBody = [
                'contents' => [['role' => 'user', 'parts' => [['text' => $pro
mpt]]]],
                'generationConfig' => ['maxOutputTokens' => 8192],
            ];

            $requestTools = [];
            if (in_array('googleSearch', $enabledTools)) {
```

```
                $requestTools[] = ['googleSearch' => new stdClass()];
            }

            if (!empty($requestTools)) {
                $requestBody['tools'] = $requestTools;
            }

            $ch = curl_init();
            curl_setopt($ch, CURLOPT_URL, $this->apiUrl);
            curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
            curl_setopt($ch, CURLOPT_POST, true);
            curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($requestBody));
            curl_setopt($ch, CURLOPT_HTTPHEADER, ['Content-Type: application/
json']);

            $rawResponse = curl_exec($ch);

            if (curl_errno($ch)) {
                throw new Exception("cURL Error: " . curl_error($ch));
            }

            $httpCode = curl_getinfo($ch, CURLINFO_HTTP_CODE);
            if ($httpCode !== 200) {
                throw new Exception("API Error: HTTP {$httpCode}. Response: "
. $rawResponse);
            }
            curl_close($ch);

            $decodedResponse = json_decode($rawResponse, true);
            if (json_last_error() !== JSON_ERROR_NONE) {
                throw new Exception("JSON Decode Error for API response.");
            }

            $responseText = $decodedResponse['candidates'][0]['content']['par
ts'][0]['text'] ?? null;
            if ($responseText === null) {
                throw new Exception("Could not find text part in the API resp
onse. The response may contain a tool call or be empty.");
            }

            $this->logPrompt($prompt, $requestBody, $rawResponse);
            return $responseText;
        } catch (Exception $e) {
            $detailedError = $e->getMessage();
            $userFriendlyError = "An unexpected error occurred while communic
ating with the AI. Please check the system logs for more details.";

            // Attempt to parse a more specific error message from the API's
raw response
            if (!empty($rawResponse)) {
```

```
                $decodedError = json_decode($rawResponse, true);
                if (isset($decodedError['error']['message'])) {
                    $apiErrorMsg = $decodedError['error']['message'];
                    $detailedError = "API Error: " . $apiErrorMsg; // Overwri
te for cleaner logs

                    // Tailor the user-facing message for common, understanda
ble errors
                    if (isset($decodedError['error']['status'])) {
                        switch ($decodedError['error']['status']) {
                            case 'INVALID_ARGUMENT':
                                $userFriendlyError = "There was an issue with
the request sent to the AI, possibly due to a configuration problem or invali
d input.";
                                break;
                            case 'PERMISSION_DENIED':
                                $userFriendlyError = "Authentication with the
AI service failed. Please verify that the API key is correct and has the nece
ssary permissions.";
                                break;
                        }
                    }
                }
            }

            // Log the detailed, technical error and return the safe, user-fr
iendly one.
            $this->logPrompt($prompt, $requestBody, $rawResponse, $detailedEr
ror);
            return "ERROR: " . $userFriendlyError;
        }
    }
}
```

```
<?php
ini_set('display_errors', 1);
error_reporting(E_ALL);

require_once 'config.php';
require_once 'train.php'; // Needed for preprocessText function
require_once 'MemoryManager.php';
require_once 'GeminiClient.php';
require_once __DIR__ . '/../vendor/autoload.php'; // If not already loaded

use NlpTools\Documents\TokensDocument;
use NlpTools\Tokenizers\WhitespaceTokenizer;
use NlpTools\Stemmers\PorterStemmer;

$featureFactory = null;
$classifier = null;
```

```php
try {
    $tfIdfModelPath = DATA_DIR . '/tf_idf.model'; // Adjust path if models ar
e elsewhere
    $classifierModelPath = DATA_DIR . '/classifier.model'; // Adjust path

    if (!file_exists($tfIdfModelPath) || !file_exists($classifierModelPath))
{
        // Handle error: models not found. You might want to log this or thro
w an exception.
        error_log("Classification models not found. Run train.php first.");
    } else {
        $featureFactory = unserialize(file_get_contents($tfIdfModelPath));
        $classifier = unserialize(file_get_contents($classifierModelPath));

        if ($featureFactory === false || $classifier === false) {
            error_log("Failed to unserialize classification models. Models mi
ght be corrupted.");
        }
    }
} catch (Exception $e) {
    error_log("Error loading classification models: " . $e->getMessage());
}

// Handle User Feedback
if ($_SERVER["REQUEST_METHOD"] == "GET" && isset($_GET['feedback'], $_GET['id
'])) {
    $memory = new MemoryManager();
    $isGood = $_GET['feedback'] === 'good';
    $memory->applyFeedback($_GET['id'], $isGood);
    $memory->saveMemory();
    header("Location: index.php");
    exit();
}

// Handle Optional Text Feedback
if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST['text_feedback'], $_
POST['interaction_id_for_feedback'])) {
    $memory = new MemoryManager();
    $interactionIdForFeedback = $_POST['interaction_id_for_feedback'];
    $textFeedback = trim($_POST['text_feedback']);

    if (!empty($textFeedback)) {
        $memory->addTextFeedback($interactionIdForFeedback, $textFeedback);
        $memory->saveMemory();
    }
    // Redirect to prevent form resubmission on refresh
    header("Location: index.php");
    exit();
}
```

```php
function runCoreLogic(string $userInput, $featureFactory, $classifier): array
{
    $memory = new MemoryManager();
    $gemini = new GeminiClient();

    $predictedIntent = "unknown";
    if ($classifier && $featureFactory) {
        $processedTokens = preprocessText($userInput); // Use the same prepro
cessing as during training
        $document = new TokensDocument($processedTokens);

        $predictedIntent = $classifier->classify(['feedback_positive', 'feedb
ack_negative'], $document);
    }

    // Dynamic Tool Selection Logic
    $toolsToUse = ['googleSearch']; // Enable search by default
    $urlPattern = '/\b(https?|ftp|file):\/\/[-A-Z0-9+&@#\/%?=~_|!:,.;]*[-A-Z0
-9+&@#\/%=~_|]/i';
    if (preg_match($urlPattern, $userInput)) {
        // The prompt guides the AI to use its search tool to analyze the URL
        $userInput .= "\n\n[SYSTEM NOTE: The user has provided a URL. Priorit
ize analyzing its content.]";
    }

    $recalled = $memory->getRelevantContext($userInput);
    $context = $recalled['context'];

    $systemPrompt = $memory->getTimeAwareSystemPrompt();
    $currentTime = "CURRENT_TIME: " . date('Y-m-d H:i:s T');
    $finalPrompt = "{$systemPrompt}\n\n---RECALLED CONTEXT---\n{$context}---E
ND CONTEXT---\n\n{$currentTime}\n\nUser query: \"{$userInput}\"";

    $aiResponse = $gemini->generateResponse($finalPrompt, $toolsToUse);

    $newInteractionId = $memory->updateMemory($userInput, $aiResponse, $recal
led['used_interaction_ids']);
    $memory->saveMemory();

    return [
        'userInput' => $userInput,
        'aiResponse' => $aiResponse,
        'context' => $context,
        'interactionId' => $newInteractionId,
        'predictedIntent' => $predictedIntent
    ];
}

// --- Main Web Mode Execution ---
```

```php
$userInput = '';
$aiResponse = '';
$context = '';
$interactionId = '';

if ($_SERVER["REQUEST_METHOD"] == "POST" && !empty($_POST['prompt'])) {
    $userInput = trim($_POST['prompt']);
    $result = runCoreLogic($userInput, $featureFactory, $classifier);
    $aiResponse = $result['aiResponse'];
    $context = $result['context'];
    $interactionId = $result['interactionId'];
    $predictedIntent = $result['predictedIntent'];
}
?>
<!DOCTYPE html>
<html lang="en" data-bs-theme="dark">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Project NEMI v5.2 (Hybrid Search + NLP)</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstr
ap.min.css" rel="stylesheet">
    <style>
        body {
            background-color: #212529;
            color: #e9ecef;
        }

        .container {
            max-width: 800px;
        }

        .card {
            border-color: #495057;
        }

        .card-header {
            background-color: #343a40;
        }

        .feedback-buttons a {
            margin-right: 10px;
        }
    </style>
</head>

<body>
    <div class="container mt-5">
        <div class="text-center mb-4">
```

```
            <h1 class="display-4">Project NEMI <span class="badge bg-success"
>v5.2</span></h1>
            <p class="lead text-muted">AI with Hybrid Search & Intent Classif
ication</p>
        </div>
        <div class="card bg-dark mb-3">
            <div class="card-body">
                <form action="index.php" method="post">
                    <textarea class="form-control bg-dark text-white mb-3" na
me="prompt" rows="3" placeholder="Ask anything or provide a URL to analyze...
"><?= htmlspecialchars($userInput) ?></textarea>
                    <button type="submit" class="btn btn-primary w-100">Send<
/button>
                </form>
            </div>
        </div>
        <?php if ($_SERVER["REQUEST_METHOD"] == "POST"): ?>
            <div class="card bg-dark mb-3">
                <div class="card-header">Your Prompt</div>
                <div class="card-body">
                    <p class="card-text"><?= nl2br(htmlspecialchars($userInpu
t)) ?></p>
                </div>
            </div>
            <?php if (isset($predictedIntent) && $predictedIntent !== 'unknow
n' && $predictedIntent !== 'question' && $predictedIntent !== 'command'): ?>
                <div class="card bg-dark mb-3">
                    <div class="card-header">Predicted Intent</div>
                    <div class="card-body">
                        <p class="card-text">
                            <span class="badge bg-info"><?= htmlspecialchars(
$predictedIntent) ?></span>
                        </p>
                    </div>
                </div>
            <?php endif; ?>
            <div class="card bg-dark mb-3">
                <div class="card-header d-flex justify-content-between align-
items-center">
                    AI Response
                    <div class="feedback-buttons">
                        <a href="?feedback=good&id=<?= urlencode($interaction
Id) ?>" class="btn btn-sm btn-outline-success">👍 Good</a>
                        <a href="?feedback=bad&id=<?= urlencode($interactionI
d) ?>" class="btn btn-sm btn-outline-danger">👎 Bad</a>
                    </div>
                </div>
                <div class="card-body">
                    <p class="card-text"><?= nl2br(htmlspecialchars($aiRespon
se)) ?></p>
```

```
                </div>
            </div>
            <?php if (!empty($interactionId)): ?>
            <div class="card bg-dark mb-3">
                <div class="card-header">Optional Text Feedback</div>
                <div class="card-body">
                    <form action="index.php" method="post">
                        <input type="hidden" name="interaction_id_for_feedbac
k" value="<?= htmlspecialchars($interactionId) ?>">
                        <textarea class="form-control bg-dark text-white mb-3
" name="text_feedback" rows="2" placeholder="Provide additional feedback...">
</textarea>
                        <button type="submit" class="btn btn-sm btn-outline-s
econdary">Submit Text Feedback</button>
                    </form>
                </div>
            </div>
            <?php endif; ?>
            <div class="card bg-dark">
                <div class="card-header"><a class="text-decoration-none" data
-bs-toggle="collapse" href="#debugCollapse">Debug: Recalled Context</a></div>
                <div class="collapse" id="debugCollapse">
                    <div class="card-body">
                        <pre class="text-white-50 small"><code><?= htmlspecia
lchars($context) ?></code></pre>
                    </div>
                </div>
            </div>
        <?php endif; ?>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstr
ap.bundle.min.js"></script>
</body>

</html>

<?php
// Now requires the new EmbeddingClient
require_once 'EmbeddingClient.php';
require_once __DIR__ . '/../vendor/autoload.php';

use NlpTools\Tokenizers\WhitespaceAndPunctuationTokenizer;
use NlpTools\Stemmers\PorterStemmer;
use NlpTools\Utils\StopWords; // Corrected use statement

class MemoryManager
{
    private array $interactions = [];
    private array $entities = [];
    private ?EmbeddingClient $embeddingClient;
```

```php
    public function __construct()
    {
        if (!is_dir(DATA_DIR)) {
            mkdir(DATA_DIR, 0775, true);
        }
        $this->loadMemory();

        // Instantiate the client only if embeddings are enabled.
        // This is where you would swap in a different client (e.g., new Olla
maEmbeddingClient()).
        $this->embeddingClient = ENABLE_EMBEDDINGS ? new EmbeddingClient() :
null;

        // Stop words are now loaded from config.php via NLP_STOP_WORDS const
ant
    }

    private function loadMemory(): void
    {
        $this->interactions = file_exists(INTERACTIONS_FILE) ? json_decode(fi
le_get_contents(INTERACTIONS_FILE), true) : [];
        $this->entities = file_exists(ENTITIES_FILE) ? json_decode(file_get_c
ontents(ENTITIES_FILE), true) : [];
    }

    public function saveMemory(): void
    {
        ksort($this->interactions);
        ksort($this->entities);
        $options = JSON_PRETTY_PRINT | JSON_UNESCAPED_SLASHES;
        file_put_contents(INTERACTIONS_FILE, json_encode($this->interactions,
$options));
        file_put_contents(ENTITIES_FILE, json_encode($this->entities, $option
s));
    }

    // --- Core Memory Retrieval (HYBRID SEARCH) ---

    /**
     * Calculates the cosine similarity between two vectors.
     * @return float A value between -1 and 1. Higher is more similar.
     */
    private function cosineSimilarity(array $vecA, array $vecB): float
    {
        $dotProduct = 0.0;
        $magA = 0.0;
        $magB = 0.0;
        $count = count($vecA);
```

```php
        for ($i = 0; $i < $count; $i++) {
            $dotProduct += $vecA[$i] * $vecB[$i];
            $magA += $vecA[$i] * $vecA[$i];
            $magB += $vecB[$i] * $vecB[$i];
        }

        $magA = sqrt($magA);
        $magB = sqrt($magB);

        if ($magA == 0 || $magB == 0) {
            return 0;
        }

        return $dotProduct / ($magA * $magB);
    }

    public function getRelevantContext(string $userInput): array
    {
        // --- Vector Search (Semantic) ---
        $semanticResults = [];
        if ($this->embeddingClient !== null) {
            $inputVector = $this->embeddingClient->getEmbedding($userInput);
            if ($inputVector !== null) {
                $similarities = [];
                foreach ($this->interactions as $id => $interaction) {
                    if (isset($interaction['embedding']) && is_array($interac
tion['embedding'])) {
                        $similarity = $this->cosineSimilarity($inputVector, $
interaction['embedding']);
                        $similarities[$id] = $similarity;
                    }
                }
                // Sort by similarity score, descending
                arsort($similarities);
                $semanticResults = array_slice($similarities, 0, VECTOR_SEARC
H_TOP_K, true);
            }
        }

        // --- Keyword Search (Lexical) ---
        $inputEntities = $this->extractEntities($userInput);
        $searchEntities = $this->normalizeAndExpandEntities($inputEntities);
        $keywordResults = [];
        foreach ($searchEntities as $entityKey) {
            if (isset($this->entities[$entityKey])) {
                foreach ($this->entities[$entityKey]['mentioned_in'] as $id)
{
                    // Store the relevance score for ranking
                    if (isset($this->interactions[$id])) {
                        $keywordResults[$id] = $this->interactions[$id]['rele
```

```
vance_score'];
                    }
                }
            }
        }
        arsort($keywordResults);

        // --- Hybrid Fusion ---
        $fusedScores = [];
        $allIds = array_unique(array_merge(array_keys($semanticResults), arra
y_keys($keywordResults)));

        foreach ($allIds as $id) {
            $semanticScore = $semanticResults[$id] ?? 0.0;
            // Normalize relevance score to be roughly between 0 and 1 for be
tter blending
            $relevanceScore = isset($keywordResults[$id]) ? tanh($keywordResu
lts[$id] / 10) : 0.0;

            // Weighted average of both scores
            $fusedScores[$id] = (HYBRID_SEARCH_ALPHA * $semanticScore) + ((1
- HYBRID_SEARCH_ALPHA) * $relevanceScore);
        }
        arsort($fusedScores);

        // --- Build Context from Fused Results ---
        $context = '';
        $tokenCount = 0;
        $usedInteractionIds = [];

        // --- NEW: Force-Inject a variable number of Recent Interactions ---
        // Check if the feature is enabled and if there are any interactions.
        if (defined('FORCED_RECENT_INTERACTIONS') && FORCED_RECENT_INTERACTIO
NS > 0 && !empty($this->interactions)) {
            // Get the last N interactions, where N is our config value.
            // `true` preserves the original keys (the interaction IDs).
            $recentInteractions = array_slice($this->interactions, -FORCED_RE
CENT_INTERACTIONS, null, true);

            // Loop through them and add them to the context string.
            foreach ($recentInteractions as $id => $interaction) {
                $timestamp = date('Y-m-d H:i:s', strtotime($interaction['time
stamp']));
                $memoryText = "[On {$timestamp}] User: '{$interaction['user_i
nput_raw']}'. You: '{$interaction['ai_output']}'.\n";
                $memoryTokenCount = str_word_count($memoryText);

                // Ensure we don't exceed the budget right away.
                if ($tokenCount + $memoryTokenCount <= CONTEXT_TOKEN_BUDGET)
{
```

```
                    $context .= $memoryText;
                    $tokenCount += $memoryTokenCount;
                    // Keep track of the IDs we've already added.
                    $usedInteractionIds[] = $id;
                }
            }
        }
        // --- End of New Code ---

        // Loop through the main hybrid search results.
        foreach ($fusedScores as $id => $score) {
            // --- UPDATED: Check if this ID was already force-included. ---
            if (in_array($id, $usedInteractionIds)) {
                continue; // Skip this memory to avoid duplication.
            }

            if (!isset($this->interactions[$id])) continue;
            $memory = $this->interactions[$id];
            $timestamp = date('Y-m-d H:i:s', strtotime($memory['timestamp']))
;
            $memoryText = "[On {$timestamp}] User: '{$memory['user_input_raw'
]}'. You: '{$memory['ai_output']}'.\n";
            $memoryTokenCount = str_word_count($memoryText);

            if ($tokenCount + $memoryTokenCount <= CONTEXT_TOKEN_BUDGET) {
                $context .= $memoryText;
                $tokenCount += $memoryTokenCount;
                $usedInteractionIds[] = $id;
            } else {
                break;
            }
        }

        return [
            'context' => empty($context) ? "No relevant memories found.\n" :
$context,
            'used_interaction_ids' => $usedInteractionIds
        ];
    }

    public function updateMemory(string $userInput, string $aiOutput, array $
usedInteractionIds): string
    {
        $recentEntities = [];
        foreach ($usedInteractionIds as $id) {
            if (isset($this->interactions[$id])) {
                $this->interactions[$id]['relevance_score'] += REWARD_SCORE;
                $this->interactions[$id]['last_accessed'] = date('c');
                if (isset($this->interactions[$id]['processed_input']['keywor
ds'])) {
```

```
                    $recentEntities = array_merge($recentEntities, $this->int
eractions[$id]['processed_input']['keywords']);
                }
            }
        }
        $recentEntities = array_unique($recentEntities);

        foreach ($this->interactions as &$interaction) {
            $keywords = $interaction['processed_input']['keywords'] ?? [];
            $isRelatedToRecentTopic = !empty(array_intersect($keywords, $rece
ntEntities));
            $decay = $isRelatedToRecentTopic ? DECAY_SCORE * RECENT_TOPIC_DEC
AY_MODIFIER : DECAY_SCORE;
            $interaction['relevance_score'] -= $decay;
        }

        $newId = uniqid('int_', true);
        $keywords = $this->extractEntities($userInput);

        // --- NEW: Generate and store embedding for the new interaction ---
        $embedding = null;
        if ($this->embeddingClient !== null) {
            $fullText = "User: {$userInput} | AI: {$aiOutput}";
            $embedding = $this->embeddingClient->getEmbedding($fullText);
        }

        $this->interactions[$newId] = [
            'timestamp' => date('c'),
            'user_input_raw' => $userInput,
            'processed_input' => ['keywords' => $keywords],
            'ai_output' => $aiOutput,
            'relevance_score' => INITIAL_SCORE,
            'last_accessed' => date('c'),
            'context_used_ids' => $usedInteractionIds,
            'embedding' => $embedding // Add the new embedding to the record
        ];

        $this->updateEntitiesFromInteraction($keywords, $newId);
        $this->pruneMemory();

        return $newId;
    }

    private function updateEntitiesFromInteraction(array $keywords, string $i
nteractionId): void
    {
        $isNovel = false;
        foreach ($keywords as $keyword) {
            $entityKey = strtolower($keyword);
            if (!isset($this->entities[$entityKey])) {
```

```
                $isNovel = true;
                $this->entities[$entityKey] = [
                    'name' => $keyword,
                    'type' => 'Concept',
                    'access_count' => 0,
                    'relevance_score' => INITIAL_SCORE,
                    'mentioned_in' => [],
                    'relationships' => []
                ];
            }
            $this->entities[$entityKey]['access_count']++;
            $this->entities[$entityKey]['relevance_score'] += REWARD_SCORE;
            if (!in_array($interactionId, $this->entities[$entityKey]['mentio
ned_in'])) {
                $this->entities[$entityKey]['mentioned_in'][] = $interactionI
d;
            }
        }

        if ($isNovel) {
            $this->interactions[$interactionId]['relevance_score'] += NOVELTY
_BONUS;
        }

        if (count($keywords) > 1) {
            foreach ($keywords as $k1) {
                foreach ($keywords as $k2) {
                    if ($k1 !== $k2) {
                        $this->entities[$k1]['relationships'][$k2] = ($this->
entities[$k1]['relationships'][$k2] ?? 0) + RELATIONSHIP_STRENGTH_INCREMENT;
                    }
                }
            }
        }
    }

    public function applyFeedback(string $interactionId, bool $isGood): void
    {
        if (!isset($this->interactions[$interactionId])) return;

        $adjustment = $isGood ? USER_FEEDBACK_REWARD : USER_FEEDBACK_PENALTY;

        $this->interactions[$interactionId]['relevance_score'] += $adjustment
;

        $contextIds = $this->interactions[$interactionId]['context_used_ids']
?? [];
        foreach ($contextIds as $id) {
            if (isset($this->interactions[$id])) {
                $this->interactions[$id]['relevance_score'] += $adjustment /
```

```
2;
            }
        }
    }

    public function addTextFeedback(string $interactionId, string $feedbackTe
xt): void
    {
        if (!isset($this->interactions[$interactionId])) return;

        // You might want to process this feedback further (e.g., sentiment a
nalysis, entity extraction)
        // For now, we'll just store it.
        $this->interactions[$interactionId]['user_feedback_text'] = $feedback
Text;
        // Optionally, you could adjust relevance scores based on text feedba
ck,
        // e.g., if you later classify the text feedback as positive or negat
ive.
    }

    private function pruneMemory(): void
    {
        if (count($this->interactions) > PRUNING_THRESHOLD) {
            uasort($this->interactions, fn($a, $b) => ($a['relevance_score']
?? 1.0) <=> ($b['relevance_score'] ?? 1.0));
            $this->interactions = array_slice($this->interactions, count($thi
s->interactions) - PRUNING_THRESHOLD, null, true);
        }
    }

    public function getTimeAwareSystemPrompt(): string
    {
        return "**PRIMARY DIRECTIVE: YOU ARE A HELPFUL, TIME-AWARE ASSISTANT.
**\n\n" .
            "**RULES OF OPERATION:**\n" .
            "1.  **ANALYZE TIMESTAMPS:** You will be given a `CURRENT_TIME` a
nd `RECALLED_CONTEXT`. Use this to understand the history of events.\n" .
            "2.  **CALCULATE RELATIVE TIME:** Interpret expressions like 'yes
terday' against the provided `CURRENT_TIME`.\n\n" .
            "**TOOL EXECUTION MANDATE:**\n" .
            "3.  **DIRECTLY USE TOOLS:** You have a `googleSearch` tool. Your
primary goal is to use this tool to directly answer the user's question. **DO
NOT describe that you are going to use a tool.** Execute it and provide the f
inal answer based on its output.\n" .
            "4.  **FULFILL THE REQUEST:** If the user provides a URL, use you
r search ability to access its content and provide a summary. If they ask a g
eneral question, use search to find the answer.";
    }
```

```
    private function extractEntities(string $text): array
    {
        // 1. Sanitize and Normalize Text
        $text = strtolower($text);
        $text = preg_replace('/https?:\/\/[^\s]+/', ' ', $text); // Keep URL
removal

        // 2. Tokenize the text using the library's robust tokenizer
        $tokenizer = new WhitespaceAndPunctuationTokenizer();
        $tokens = $tokenizer->tokenize($text);

        // 3. Filter out stop words using the list from config.php
        // Use StopWords class from NlpTools\Utils and its transform method
        $stopWords = new StopWords(NLP_STOP_WORDS);
        $filteredTokens = [];
        foreach ($tokens as $token) {
            $transformedToken = $stopWords->transform($token);
            if ($transformedToken !== null) {
                $filteredTokens[] = $transformedToken;
            }
        }

        // 4. Reduce words to their root form (stemming)
        $stemmer = new PorterStemmer();
        $stemmedTokens = array_map([$stemmer, 'stem'], $filteredTokens);

        // 5. Final cleanup and return unique entities
        // Replace the arbitrary strlen > 3 with a slightly more lenient filt
er
        return array_filter(array_unique($stemmedTokens), fn($word) => strlen
($word) > 2);
    }

    private function normalizeAndExpandEntities(array $baseEntities): array
    {
        $expanded = $baseEntities;
        foreach ($baseEntities as $entityKey) {
            if (isset($this->entities[$entityKey]['relationships'])) {
                $expanded = array_merge($expanded, array_keys($this->entities
[$entityKey]['relationships']));
            }
        }
        return array_unique($expanded);
    }
}

<?php
ini_set('display_errors', 1);
error_reporting(E_ALL & ~E_DEPRECATED);
```

```php
require_once __DIR__ . '/../vendor/autoload.php';
require_once __DIR__ . '/config.php';

use NlpTools\Tokenizers\WhitespaceAndPunctuationTokenizer;
use NlpTools\Stemmers\PorterStemmer;
use NlpTools\Utils\StopWords;
use NlpTools\FeatureFactories\DataAsFeatures;
use NlpTools\Classifiers\MultinomialNBClassifier;
use NlpTools\Models\FeatureBasedNB;
use NlpTools\Documents\TrainingSet;
use NlpTools\Documents\TokensDocument;

// --- 1. Preprocessing Function (mirrors MemoryManager::extractEntities) ---
function preprocessText(string $text): array
{
    // 1. Sanitize and Normalize Text
    $text = strtolower($text);
    $text = preg_replace('/https?:\/\/[^\s]+/', ' ', $text); // Keep URL remo
val

    // 2. Tokenize the text using the library's robust tokenizer
    $tokenizer = new WhitespaceAndPunctuationTokenizer();
    $tokens = $tokenizer->tokenize($text);

    // 3. Filter out stop words using the list from config.php
    $stopWords = new StopWords(NLP_STOP_WORDS);
    $filteredTokens = [];
    foreach ($tokens as $token) {
        $transformedToken = $stopWords->transform($token);
        if ($transformedToken !== null) {
            $filteredTokens[] = $transformedToken;
        }
    }

    // 4. Reduce words to their root form (stemming)
    $stemmer = new PorterStemmer();
    $stemmedTokens = array_map([$stemmer, 'stem'], $filteredTokens);

    // 5. Final cleanup and return unique entities
    return array_filter(array_unique($stemmedTokens), fn($word) => strlen($wo
rd) > 2);
}

// --- 2. Load Training Data ---
$trainingDataFile = __DIR__ . '/test/training_data.csv';
$trainingData = [];

if (($handle = fopen($trainingDataFile, 'r')) !== FALSE) {
    $header = fgetcsv($handle); // Skip header row
    while (($row = fgetcsv($handle)) !== FALSE) {
```

```php
        if (count($row) == 2) {
            $trainingData[] = ['text' => $row[0], 'label' => $row[1]];
        }
    }
    fclose($handle);
} else {
    die("Error: Could not open training_data.csv\n");
}

if (empty($trainingData)) {
    die("Error: No training data found in training_data.csv\n");
}



// --- 3. Prepare Training Set for NLP-Tools ---
$trainingSet = new TrainingSet();
$labels = [];

foreach ($trainingData as $item) {
    $processedTokens = preprocessText($item['text']);
    $trainingSet->addDocument($item['label'], new TokensDocument($processedTo
kens));
    $labels[] = $item['label'];
}

// --- 4. Feature Generation (TF-IDF) ---
$featureFactory = new DataAsFeatures();

// --- 5. Classifier Training (Multinomial Naive Bayes) ---
$model = new FeatureBasedNB();

$model->train($featureFactory, $trainingSet);


$classifier = new MultinomialNBClassifier($featureFactory, $model);


// --- 6. Serialize and Save Models ---
$tfIdfModelFile = DATA_DIR . '/tf_idf.model';
$classifierModelFile = DATA_DIR . '/classifier.model';

file_put_contents($tfIdfModelFile, serialize($featureFactory));
file_put_contents($classifierModelFile, serialize($classifier));



?>
```

Nota Bene: Rules for Future Document Updates

AI and/or agents must adhere to the following rules to maintain the consistency and integrity of this document.

1. **Strict Versioning:** All modifications, regardless of size, mandate a version increment.
   - Patch (e.g., 4.2 -> 4.2.1): Typographical corrections, code comments, non-functional changes.
   - Minor (e.g., 4.2 -> 4.3): New features, changes in logic, enhancements that are backward-compatible.
   - Major (e.g., 4.0 -> 5.0): Significant architectural changes or rewrites that are not backward-compatible.
2. **Mandatory Changelog:** Every new version must be documented in a "Revision History" section added to the top of the document. The entry must include the new version number, date, author, and a concise, bulleted list of all changes made.
3. **Code Synchronization:** The source code presented within this document must be an exact and complete representation of the functional source code files. Any change to a .php file must be immediately and accurately reflected in its corresponding code block herein.
4. **Preserve Architecture:** New functionality must be integrated into the existing "Brain Analogy" component structure (index.php, MemoryManager.php, GeminiClient.php, config.php). If a new component is necessary, its purpose and interaction with the existing components must be explicitly defined.
5. **Centralize Configuration:** All new constants, thresholds, or configurable parameters must be added exclusively to config.php. No hardcoded "magic values" are permitted in the logic of other files.
6. **Maintain Structural Integrity:** The existing numbered heading structure of this document must be preserved. New content should be added as subsections within the relevant existing section. Do not alter the primary document flow.
7. **Direct and Specific Language:** All descriptions of changes must be technical, direct, and unambiguous. Clearly state what was changed, why it was changed, and reference the specific function or component affected.