Feature Engineering for Agents: An Adaptive Cognitive Architecture for Interpretable ML Monitoring

Gusseppe Bravo-Rocca
gusseppe.bravo@bsc.es
Barcelona Supercomputing Center
Barcelona, Spain

Peini Liu
peini.liu@bsc.es
Barcelona Supercomputing Center
Barcelona, Spain

Jordi Guitart
jordi.guitart@bsc.es
Barcelona Supercomputing Center
Universitat Politècnica de Catalunya
Barcelona, Spain

Rodrigo M Carrillo-Larco
rmcarri@emory.edu
Emory University
Atlanta, GA, USA

Ajay Dholakia
adholakia@lenovo.com
Lenovo Infrastructure Solutions Group
Morrisville, NC, USA

David Ellison
dellison@lenovo.com
Lenovo Infrastructure Solutions Group
Morrisville, NC, USA

ABSTRACT

Monitoring Machine Learning (ML) models in production environments is crucial, yet traditional approaches often yield verbose, low-interpretability outputs that hinder effective decision-making. We propose a cognitive architecture for ML monitoring that applies feature engineering principles to agents based on Large Language Models (LLMs), significantly enhancing the interpretability of monitoring outputs. Central to our approach is a Decision Procedure module that simulates feature engineering through three key steps: Refactor, Break Down, and Compile. The Refactor step improves data representation to better capture feature semantics, allowing the LLM to focus on salient aspects of the monitoring data while reducing noise and irrelevant information. Break Down decomposes complex information for detailed analysis, and Compile integrates sub-insights into clear, interpretable outputs. This process leads to a more deterministic planning approach, reducing dependence on LLM-generated planning, which can sometimes be inconsistent and overly general. The combination of feature engineering-driven planning and selective LLM utilization results in a robust decision support system, capable of providing highly interpretable and actionable insights. Experiments using multiple LLMs demonstrate the efficacy of our approach, achieving significantly higher accuracy compared to various baselines across several domains.

KEYWORDS

Agent-based Cognitive Architecture; Machine Learning Monitoring; Large Language Models

ACM Reference Format:
Gusseppe Bravo-Rocca, Peini Liu, Jordi Guitart, Rodrigo M Carrillo-Larco, Ajay Dholakia, and David Ellison. 2025. Feature Engineering for Agents: An Adaptive Cognitive Architecture for Interpretable ML Monitoring. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 9 pages.

Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Y. Vorobeychik, S. Das, A. Nowé (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

1 INTRODUCTION

Monitoring Machine Learning (ML) models in production environments is a critical task, as model performance can degrade over time due to various factors [3, 4]. Traditional monitoring approaches, such as distribution drift detection and feature attribution [7, 9], while important, often require substantial technical expertise to interpret and act upon. As the number of deployed models increases, the time required for thorough analysis of each model's performance becomes a significant bottleneck in maintaining up-to-date and reliable ML systems. A pressing question emerges: can we leverage Large Language Models (LLMs) to automate and enhance the monitoring of ML models? Specifically, can LLMs analyze the outputs of monitoring tools, interpret these results, and relate them to the dataset to provide meaningful, actionable insights? This capability would enable the generation of detailed, interpretable reports on model issues, facilitating crucial decisions such as model retraining, new data labeling, or model replacement. While established monitoring tools like Alibi Detect[1] provide valuable technical metrics (e.g., drift scores, SHAP values), interpreting these outputs often requires significant expertise. There is an opportunity to complement them by making their outputs more interpretable and actionable through natural language processing. To address this challenge, we propose a novel cognitive architecture for ML monitoring that applies feature engineering principles to LLM-based agents, significantly enhancing the interpretability of monitoring outputs. Our approach, termed CAMA (Cognitive Architecture for Monitoring Agent), combines structured memory components with a sophisticated decision procedure to generate highly interpretable and actionable insights (see Figure 1).

Central to our architecture is the Decision Procedure (DP) module, which implements a feature engineering-inspired approach through three key steps:

- Refactor: Improves data representation to better capture feature semantics, allowing the LLM to focus on salient aspects of the monitoring data while reducing noise.
- Break Down: Decomposes complex information for detailed analysis, enabling comprehensive understanding of individual features and their interactions.

[1] https://github.com/SeldonIO/alibi-detect

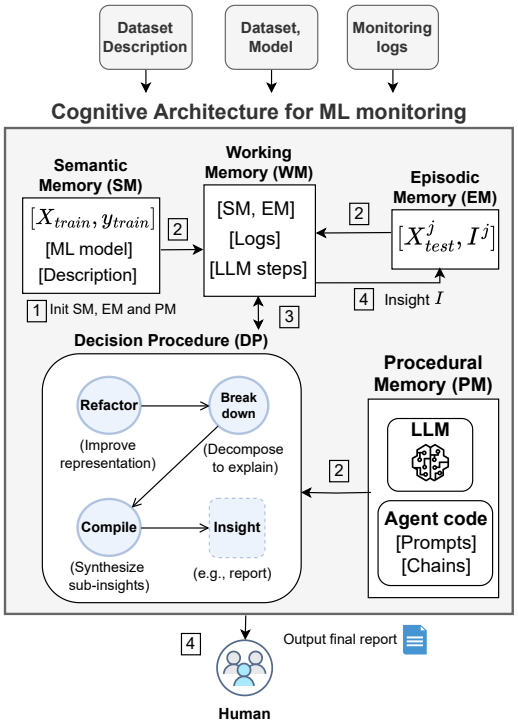

Figure 1: Cognitive architecture for ML monitoring. The system integrates Procedural (PM), Episodic (EM), Semantic (SM), and Working (WM) Memory. The central Decision Procedure (DP) implements a feature engineering-inspired approach: Refactor, Break Down, and Compile. SM stores reference data $[X_{\text{train}}, y_{\text{train}}]$, the ML model, and dataset descriptions. EM captures current test data $[X^j_{\text{test}}, I^j]$ and insights. WM facilitates information exchange. PM sets the LLM and agent code. This architecture enhances monitoring output interpretability, providing clear, actionable insights for human operators.

- Compile: Integrates sub-insights into clear, interpretable outputs, providing a holistic view of the model's performance.

This process leads to a more deterministic planning approach, reducing dependence on LLM-generated planning, which can sometimes be inconsistent and overly general. Our architecture incorporates principles from cognitive science, particularly the structured memory and thinking processes inspired by human cognition [11]. This approach ensures efficiency while maintaining the capability for deep, thoughtful analysis when required. The effectiveness of our approach is demonstrated through experimental evaluations using four different LLMs, ranging from smaller models like llama-3.2-1b to large-scale models such as llama3-70b [12] and gpt-4o-mini. Our method, CAMA, consistently outperforms various baselines across different metrics. In our experiments, we work with three distinct datasets with different level of complexity designed to simulate real-world distribution drift scenarios. These datasets represent diverse domains in financial services and customer's behavior prediction, allowing us to demonstrate the versatility and robustness of our approach in handling various types of distribution shifts commonly encountered in production environments.

Our contributions are the following:

- The development of a cognitive architecture for monitoring ML models that enhances the interpretability and actionability of monitoring reports.
- A novel decision procedure that applies feature engineering principles to LLM-based analysis, resulting in more deterministic and reliable planning.
- A comprehensive evaluation demonstrating the superiority of our approach across different LLM models and metrics, using datasets that mimic real-world distribution drift challenges.

2 RELATED WORK

Various prompting and planning techniques have been investigated to enhance the decision-making capabilities of LLMs. This section covers key methodologies relevant to our approach, highlighting their strengths and limitations, and contrasting them with our adaptive cognitive architecture.

2.1 Direct Prompting Techniques

Prompting (Standard): This method presents the model with a specific question or task (bare prompt) and requests a response without additional context. While effective for simple queries, it often falls short in complex reasoning tasks [1, 14]. For instance, while a model might easily answer "What is the capital of France?", solving equations typically requires careful regularization or model selection. Our solution leverages a multi-faceted cognitive architecture that scales with task complexity, providing more accurate and contextually sensitive responses.

Chain of Thought (CoT) prompting [6, 15] enables LLMs to formulate their own "thinking procedure" by eliciting intermediate reasoning steps. This technique significantly improves performance on complex reasoning tasks. However, CoT lacks a structured mechanism to handle diverse and evolving datasets systematically. Our approach incorporates similar reflective processes within a cognitive architecture, allowing the model to focus on minimal context stored in Working Memory (WM) and avoid noise.

Reflection methods leverage the model's ability to reconsider and refine its previous responses, improving accuracy through iterative self-correction [10]. While effective, Reflection is limited by its reliance on continuous feedback loops, which can be computationally expensive. Our approach mitigates these limitations by utilizing a structured Decision Procedure to balance quick assessments with thorough analyses, thereby optimizing resource utilization.

2.2 Agentic Behavior Techniques

ReAct [16] combines reasoning and action at every step, excelling in dynamic, context-specific responses. However, it can rush to solutions without sufficient analysis when deeper understanding is required. In contrast, our approach embeds action-oriented strategies within a structured cognitive framework. This allows for dynamic responses while also ensuring deeper, more systematic analysis when needed. By applying feature engineering principles, our

approach refines and compiles monitoring data, delivering more contextually relevant and actionable insights, balancing immediate action with thorough analysis.

Self-Discover methodology [17] enables LLMs to autonomously generate reasoning structures, reducing dependency on pre-constructed prompts. While effective for novel problems, it may struggle with consistency across different contexts without a structured memory system. Our approach enhances self-discovery by using structured Semantic Memory (SM), allowing the LLM to leverage current data like dataset descriptions to improve reasoning and maintain contextually relevant insights.

The **Plan-and-Solve** technique [13] prompts models to plan a solution pathway before execution, improving zero-shot chain-of-thought reasoning. However, its linear approach can be restrictive for complex problem-solving. In contrast, our approach embeds planning in the architecture. This allows for a more flexible and dynamic approach, ensuring more efficient and accurate execution of complex reasoning tasks.

More recently, **PH-LLM** [2], a fine-tuned variant of the Gemini model for interpreting personal health data, highlights the effectiveness of using domain-specific data to generate personalized insights. However, PH-LLM relies on model fine-tuning, whereas our approach is fully LLM-agnostic. Rather than customizing specific models, we employ structured memory and reasoning techniques that work across any LLM, regardless of size or architecture.

In contrast to existing methods, our approach integrates structured memory, feature engineering, and adaptive decision-making. This combination enables more efficient, accurate, and contextually relevant analysis of ML model performance, overcoming the limitations of current approaches in handling complex, evolving datasets in production environments.

3 APPROACH

While our architecture incorporates established memory components, our key innovation lies in the decision-making process (Refactor, Break Down, Compile) that applies feature engineering principles to enhance monitoring interpretability. CAMA operates primarily on the outputs of monitoring tools (e.g., drift metrics, SHAP values) rather than directly on raw datasets. The architecture interprets these monitoring results to provide actionable recommendations, such as model retraining timing or data pipeline adjustments. This design choice makes CAMA format-agnostic and applicable across different monitoring scenarios and metrics, regardless of the underlying data domains.

3.1 Memory Modules

Our architecture utilizes four distinct memory modules, each serving a specific role in the monitoring process:

Procedural Memory ($\mathcal{M}_\mathcal{P}$): Stores the agent code, including prompts and chains, enabling effective utilization of the LLM within the cognitive architecture. It encompasses both explicit procedural knowledge encoded in the agent's code and implicit knowledge embedded in the LLM weights.

Episodic Memory ($\mathcal{M}_\mathcal{E}$): Retains specific past instances of model monitoring, including test data X^j_{test} and generated insights I^j. Formally represented as:

$$\mathcal{M}_\mathcal{E} = (X^j_{\text{test}}, I^j)_{j=1}^n \tag{1}$$

This memory is crucial for learning from historical data and enhancing decision-making based on past experiences.

Semantic Memory ($\mathcal{M}_\mathcal{S}$): Contains generalized knowledge extracted from training data, the ML model, and monitoring tools. Defined as:

$$\mathcal{M}_\mathcal{S} = ([X_{\text{train}}, y_{\text{train}}], \mathcal{H}, \mathcal{T}) \tag{2}$$

where $\mathcal{H}$ represents the ML model and $\mathcal{T}$ represents the tools. This memory supports contextual understanding and retrieval of relevant information.

Working Memory ($\mathcal{M}_\mathcal{W}$): Holds the current context, including test data, intermediate insights, and ongoing reasoning processes (LLM steps). Formulated as:

$$\mathcal{M}_\mathcal{W} = (\mathcal{M}_\mathcal{E}, \mathcal{M}_\mathcal{S}, \text{LLM steps}) \tag{3}$$

$\mathcal{M}_\mathcal{W}$ facilitates dynamic adaptation of responses based on real-time analysis.

3.2 Decision Procedure

Our decision procedure is the core of the agent, it implements a feature engineering-inspired approach through three key steps: Refactor, Break Down, and Compile. This process enhances the interpretability and actionability of monitoring insights. Algorithm 1 outlines the overall decision procedure.

Algorithm 1: Decision Procedure

Input: Procedural Memory $\mathcal{M}_\mathcal{P}$, Semantic Memory $\mathcal{M}_\mathcal{S}$, Episodic Memory $\mathcal{M}_\mathcal{E}$, Test data X_{test};

Output: Monitoring Report or Deep Insight I_{deep};

Initialize Working Memory:
$\mathcal{M}_\mathcal{W} \leftarrow \mathcal{M}_\mathcal{E}, \mathcal{M}_\mathcal{S}, \text{LLM steps}$;

Refactor Step: **begin**
  $I_{\text{ref}} \leftarrow \text{Refactor}(X_{\text{test}}, \mathcal{M}_\mathcal{S}, \mathcal{M}_\mathcal{E})$;
end

Break Down Step: **begin**
  forall *features* $f_i \in I_{ref}$ ***in parallel*** **do**
    Generate prompt p_i using $\mathcal{M}_\mathcal{P}$ and $\mathcal{M}_\mathcal{W}$;
    $r_{f_i} \leftarrow \text{AnalyzeFeature}(f_i, p_i)$;
  end
  $I_{\text{div}} \leftarrow r_{f_i} \mid f_i \in I_{\text{ref}}$;
end

Compile Step: **begin**
  $I_{\text{deep}} \leftarrow \text{CompileReport}(I_{\text{div}}, \mathcal{M}_\mathcal{P})$;
end

Update Episodic Memory: $\mathcal{M}_\mathcal{E} \leftarrow \mathcal{M}_\mathcal{E} \cup (X_{\text{test}}, I_{\text{deep}})$;

return I_{deep};

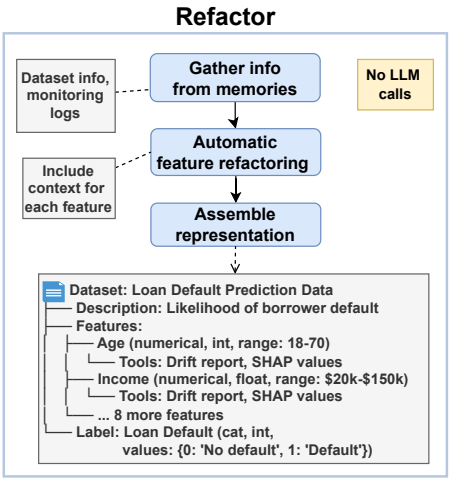

Figure 2: Refactor step gathers and structures information from memories without LLM calls.

3.2.1 Decision Steps. The decision procedure consists of three key steps that work together to provide comprehensive monitoring analysis:

Refactor improves data representation by gathering information from memories, performing automatic feature refactoring without LLM calls, and assembling a structured representation with context for each feature. The output is a well-organized dataset representation, as shown in Figure 2.

Break Down decomposes complex information for detailed analysis. As illustrated in Figure 3, this step uses a dynamic system prompt to guide parallel feature analysis, collecting detailed insights and integrating tool results like drift reports and SHAP [8] values.

Compile integrates all insights into clear, interpretable outputs. As shown in Figure 4, it generates an overview, gathers feature reports, assembles a comprehensive report using LLM calls, and saves it to Episodic Memory.

The resulting report provides a holistic view through summary, key points, and detailed feature insights. By leveraging structured memory, feature engineering principles, and adaptive learning, we enhance the LLM's ability to provide accurate, context-aware insights. The LLM serves as a reasoning engine enabling multi-step analysis, and our LLM-agnostic approach allows flexibility in model selection, with even small models performing well at certain tasks.

3.3 Operational Characteristics

CAMA is designed to be flexible in its deployment and operation. While primarily user-triggered, it can be integrated into automated workflows through cron jobs or DevOps pipelines. The architecture supports composability, allowing multiple CAMA agents to be chained together in a multi-agent pipeline. For example, when one agent detects significant drift, it can trigger another agent to analyze the drift's impact and suggest model improvements. This multi-agent capability enables sophisticated monitoring workflows where each agent specializes in different aspects of the monitoring process.

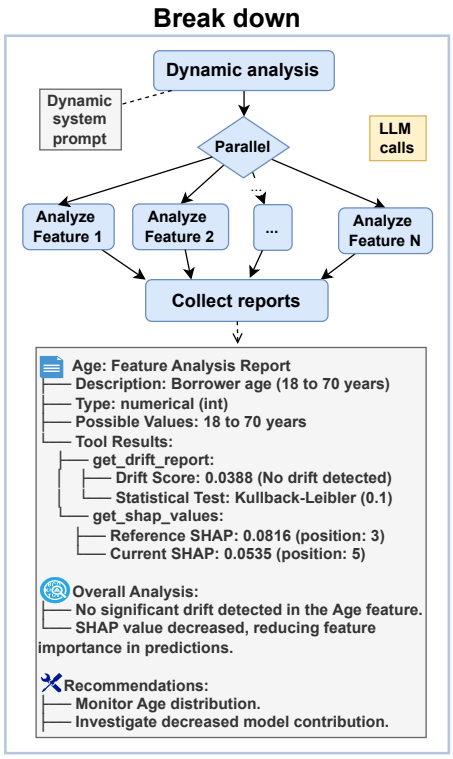

Figure 3: Break Down step analyzes features in parallel using LLM calls.

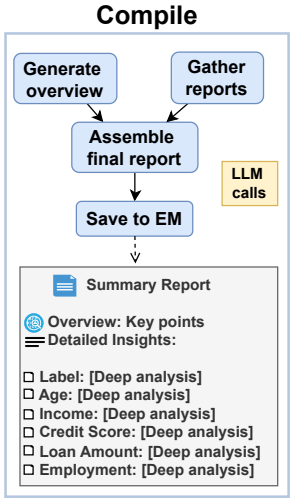

Figure 4: Compile step generates the final comprehensive report.

4 EXPERIMENTAL SETUP

4.1 Models and Datasets

4.1.1 LLM Models. We evaluated four LLMs to compare their reasoning capabilities for interpreting monitoring tool outputs:

- **Llama-3.2-1b**: A 1 billion parameter model, representing smaller, more resource-efficient LLMs.
- **Llama3-8b**: An 8 billion parameter model with an 8,192 token context window, representing medium-sized LLMs.
- **Llama3-70b**: A 70 billion parameter model with an 8,192 token context window, representing large-scale LLMs.
- **GPT-4o-mini**: A multimodal model with a larger context window, representing state-of-the-art performance. The exact model size has not been published.

All models were used with a temperature setting of 0 to ensure consistency in responses. This range of models allows us to evaluate the performance of our approach across different model sizes and architectures, from resource-efficient options to state-of-the-art performers. The inclusion of Llama-3.2-1b enables us to assess the effectiveness of our approach on smaller models, which are often preferred in resource-constrained environments. The comparison across these models provides insights into how our cognitive architecture scales with different LLM capabilities and sizes.

4.1.2 Datasets. We curated three datasets representing diverse domains and drift scenarios, each with varying levels of complexity. These datasets were synthetically generated using GPT-4o and subsequently refined to ensure realism and consistency. The refinement process addressed potential issues such as nonsensical drifts, ensured correct value ranges (e.g., for age), and maintained appropriate type interdependencies and distributions. Detailed information about the dataset generation and refinement process is available in the supplementary file. The datasets are as follows:
Loan Default Prediction (Easy): A financial dataset with 10 features (7 numerical, 3 categorical). This dataset is considered easy due to its straightforward feature set and clear relationships between variables such as income, credit score, and loan default probability. Features include Age (18-70), Income ($20K-$150K), Credit Score (300-850), Loan Amount ($1K-$50K), and categorical variables like Home Ownership (Rent/Own/Mortgage).
Eligibility Simulation (Medium): A dataset with 5 features (2 numerical, 3 categorical). It is classified as medium difficulty due to the interplay between socioeconomic factors and eligibility criteria, requiring more nuanced interpretation. The numerical features are designed with realistic ranges based on socioeconomic indicators.
Chronic Condition Prediction (Difficult): A healthcare dataset with 10 features (6 numerical, 4 categorical). This dataset is considered difficult due to the complex interactions between various health indicators, lifestyle factors, and socioeconomic variables in predicting chronic conditions. Numerical features include health metrics and lifestyle indicators with clinically relevant ranges.

Each dataset was designed to exhibit realistic distribution drifts, challenging the monitoring capabilities of our system and baselines across different complexity levels. It is important to note that while the original datasets contain 1,000 samples each, our monitoring tools utilize only 100 samples for drift calculation and feature attribution. This approach ensures that the sample size does not significantly impact the monitoring process, allowing for efficient analysis regardless of the original dataset size.

4.2 Evaluation Methodology

Our evaluation follows the MMLU (Measuring Massive Multitask Language Understanding) approach [5], using four key metrics and a structured evaluation process:

(1) **Accuracy (↑)**: Percentage of correct answers to multi-choice questions, measuring report quality.
(2) **Unknown Ratio (↓)**: Percentage of "I DON'T KNOW" responses, indicating information gaps.
(3) **Tokens (↓)**: Number of tokens generated, measuring conciseness and efficiency.
(4) **Time (↓)**: Elapsed time for report generation, affected by model latency.

All metrics are reported with mean and standard deviation across multiple runs.

The evaluation consists of five steps:

(1) **Ground Truth Creation**: Comprehensive reports using raw data and monitoring tools, including executive summaries, dataset synopses, and detailed analyses.
(2) **Question Generation**: 39 multi-choice questions per dataset created by GPT-4o, covering various aspects of the reports.
(3) **Report Generation**: Reports generated using CAMA and baseline methods across all LLM models.
(4) **Evaluation**: GPT-4o as impartial judge answers pre-defined questions for each report.
(5) **Metric Calculation**: Comparison of GPT-4o's answers against ground truth for accuracy and unknown ratio, plus token and time measurements.

This systematic approach ensures fair comparison across methods and models, with robust and consistent performance assessment.

4.3 Comparison Methods

We compared CAMA against six alternative methods:

- Standard (I/O): Direct input-output prompting.
- Chain of Thought (CoT): Prompting with intermediate reasoning steps.
- Reflection: Iterative self-correction approach.
- ReAct: Reasoning and acting framework.
- Self-Discover: Autonomous reasoning structure generation.
- Plan & Execute: Structured planning and execution approach.

4.4 Implementation Details

Our implementation executed LLM models across two distinct experimental setups:

- **Large Language Models API**: We integrated GROQ[2], OpenRouter[3], and OpenAI APIs to access large-scale language model inference capabilities.
- **Small Language Models API**: To access inference capabilities of CPU-optimized smaller language models, we leveraged the Intel® Optimized Stack on a Lenovo cluster with dual Intel® Xeon® Platinum 8360Y processors @ 2.40GHz and 256 GB RAM, running Ubuntu 22.04 LTS. This setup utilized the Intel® AI Analytics Toolkit (via Docker image

[2]https://groq.com/
[3]https://openrouter.ai/

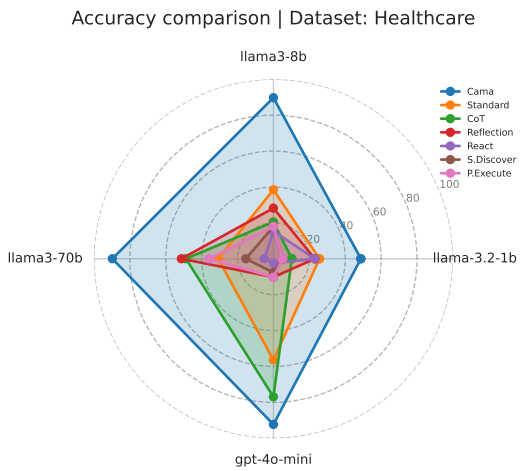

Figure 5: Accuracy comparison for the Healthcare dataset across different models and methods.

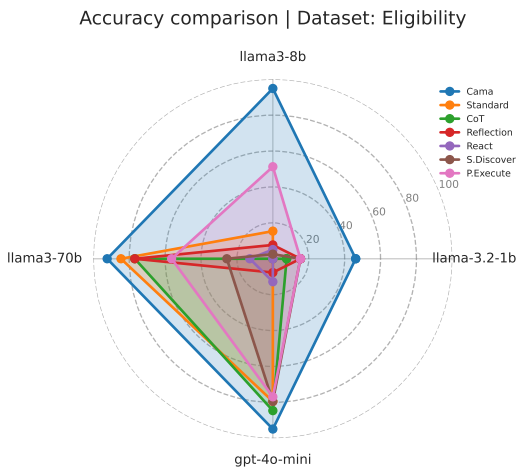

Figure 6: Accuracy comparison for the Eligibility dataset across different models and methods.

intel/oneapi-aikit:devel-ubuntu22.04[4]) and Intel® Extension for PyTorch[5] 1.12.100+cpu.

We also leveraged other key technologies and libraries:

- **Cognitive Architecture**: Implemented using Langchain[6] and Langgraph[7] for flexible and efficient workflow management.
- **Data Representation**: Docarray[8] was employed for handling unstructured data representations.
- **Monitoring Tools**: We utilized Alibi Detect[9] for calculating drift scores and feature attributions.

All methods were implemented using Python 3.9, ensuring consistency in environment and dependencies for fair comparisons

4.5 Reproducibility

To facilitate reproducibility, we have released our codebase[10], including dataset generation scripts, model implementations, evaluation pipelines, and configuration files for all experiments.

5 RESULTS AND DISCUSSION

Our proposed method, CAMA, demonstrates superior performance across all evaluated scenarios, as shown in Table 1 and Figures 5, 6, and 7. These results provide a comprehensive comparison of our approach against other established methods across different datasets and model sizes.

5.1 Performance Analysis

CAMA consistently outperforms all other methods across different model sizes and datasets:

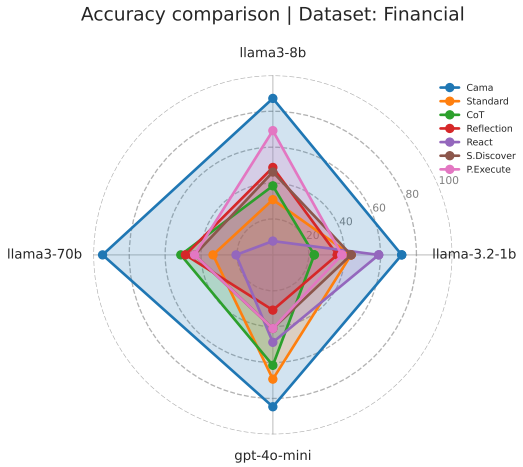

Figure 7: Accuracy comparison for the Financial dataset across different models and methods.

- **Accuracy:** CAMA achieves the highest accuracy for all models, with particularly impressive results for larger models. For llama3-70b, CAMA reaches 92.3% ± 1.5% accuracy, significantly outperforming the next best method (CoT and Reflection, both at 59.0% ± 9.0%).
- **Unknown Ratio:** Our method maintains the lowest unknown ratios, reaching 0.0% for both llama3-70b and gpt-4o-mini, indicating comprehensive and informative reports.
- **Tokens and Time:** While CAMA generally generates more tokens compared to simpler methods, the trade-off in performance is substantial. The increased token count is justified by the significant gains in accuracy and reduction in unknown responses.

The radar plots (Figures 5, 6, and 7) demonstrate CAMA's superior performance across all datasets and model sizes. In each plot,

[4]https://hub.docker.com/r/intel/oneapi-aikit
[5]https://github.com/intel/intel-extension-for-pytorch
[6]https://www.langchain.com/
[7]https://langchain-ai.github.io/langgraph/
[8]https://docs.docarray.org/
[9]https://github.com/SeldonIO/alibi-detect
[10]https://github.com/gusseppe/cognitive_architecture_checker/

Table 1: Average Performance Comparison Across All Datasets

Model	Metric	CAMA	Standard	CoT	Reflection	React	S.Discover	P.Execute
llama-3.2-1b	**Accuracy (↑)**	**55.6 ± 8.1**	23.1 ± 12.6	13.7 ± 4.8	24.8 ± 6.0	27.4 ± 17.2	21.4 ± 11.5	19.7 ± 9.9
	Unknown (↓)	**35.9 ± 5.4**	76.9 ± 12.6	85.5 ± 4.5	70.1 ± 8.9	69.2 ± 19.4	77.8 ± 12.3	80.3 ± 9.9
	Tokens (↓)	24.9 ± 5.2	4.9 ± 0.2	**3.8 ± 0.8**	14.2 ± 0.5	23.6 ± 9.6	17.3 ± 0.7	23.7 ± 7.0
	Time (↓)	5.6 ± 0.3	**1.5 ± 0.6**	2.0 ± 0.4	19.7 ± 16.2	35.2 ± 23.5	61.0 ± 27.7	46.6 ± 2.2
llama3-8b	**Accuracy (↑)**	**90.6 ± 2.3**	28.2 ± 6.8	19.7 ± 11.1	28.2 ± 11.8	9.4 ± 3.1	22.2 ± 12.8	46.1 ± 15.0
	Unknown (↓)	**0.9 ± 0.9**	70.1 ± 8.2	80.3 ± 11.1	68.4 ± 12.8	89.8 ± 3.9	76.1 ± 13.4	47.9 ± 16.6
	Tokens (↓)	19.8 ± 4.1	5.0 ± 0.1	**4.8 ± 0.2**	10.0 ± 3.3	5.7 ± 1.3	20.2 ± 3.2	34.9 ± 3.7
	Time (↓)	5.2 ± 0.3	**1.1 ± 0.1**	3.6 ± 2.6	11.7 ± 7.8	6.6 ± 4.2	22.7 ± 11.1	38.3 ± 12.0
llama3-70b	**Accuracy (↑)**	**92.3 ± 1.5**	49.6 ± 17.5	59.0 ± 9.0	59.0 ± 9.0	12.8 ± 4.4	28.2 ± 8.2	45.3 ± 6.0
	Unknown (↓)	**0.0 ± 0.0**	42.7 ± 21.4	35.9 ± 12.8	32.5 ± 12.6	85.5 ± 3.1	66.7 ± 8.3	50.4 ± 7.6
	Tokens (↓)	19.1 ± 2.2	**4.0 ± 0.9**	4.3 ± 1.0	12.3 ± 1.7	7.1 ± 1.5	19.5 ± 3.4	37.3 ± 25.8
	Time (↓)	72.2 ± 11.7	**12.5 ± 5.2**	35.3 ± 6.5	122.2 ± 13.3	21.2 ± 2.2	123.6 ± 29.9	249.0 ± 128.9
gpt-4o-mini	**Accuracy (↑)**	**90.6 ± 3.1**	68.4 ± 6.7	74.3 ± 6.8	16.3 ± 7.3	21.4 ± 14.0	42.7 ± 20.7	42.7 ± 19.2
	Unknown (↓)	**0.0 ± 0.0**	29.1 ± 6.0	19.7 ± 9.5	82.9 ± 8.1	75.2 ± 15.2	56.4 ± 20.0	51.3 ± 16.0
	Tokens (↓)	20.2 ± 4.0	4.8 ± 0.8	**4.5 ± 1.1**	13.6 ± 2.3	7.8 ± 1.7	22.6 ± 1.8	495.9 ± 10.8
	Time (↓)	27.2 ± 5.5	**19.1 ± 3.6**	23.9 ± 2.9	57.0 ± 2.9	25.9 ± 3.4	101.1 ± 37.0	340.3 ± 64.1

CAMA's accuracy (represented by the blue area) consistently extends further from the center than other methods, indicating higher accuracy across all scenarios.

5.2 Comparative Analysis

CAMA's performance contrasts sharply with other approaches:

- **Standard (I/O):** Shows moderate accuracy but high unknown ratios, particularly for smaller models.
- **Chain of Thought (CoT):** Performs well with larger models but struggles with smaller ones, suggesting its effectiveness depends on model size.
- **Reflection:** Shows inconsistent performance across model sizes and datasets.
- **ReAct:** Performs poorly across all model sizes, indicating potential issues with its reasoning and action framework in this context.
- **Self Discover:** Demonstrates low accuracy and high unknown ratios, suggesting difficulties in autonomous reasoning structure generation for this task.
- **Plan & Execute:** Shows moderate performance with some models but struggles with consistency across datasets and model sizes.

These results reveal that while some methods (like CoT) work better with larger models, CAMA consistently outperforms all methods across model sizes and datasets. This suggests that our cognitive architecture effectively leverages the strengths of various models while providing additional structure and guidance.

5.3 Model Size Impact

The impact of model size on performance is evident:

- **llama-3.2-1b:** Even with this smaller model, CAMA achieves 55.6% ± 8.1% accuracy, significantly outperforming other methods.
- **llama3-8b:** CAMA's performance jumps to 90.6% ± 2.3% accuracy, demonstrating substantial improvement with increased model size.
- **llama3-70b and gpt-4o-mini:** These larger models show CAMA's peak performance, with accuracies of 92.3% ± 1.5% and 90.6% ± 3.1% respectively.

This trend suggests that while CAMA benefits from larger models, it can still provide significant improvements even with smaller, more resource-efficient models.

5.4 Dataset-specific Performance

Figures 5, 6, and 7 illustrate CAMA's consistent superior performance across different datasets:

- **Healthcare:** CAMA shows a clear advantage, particularly with llama3-8b and llama3-70b models.
- **Eligibility:** The performance gap is most pronounced in this dataset, with CAMA significantly outperforming other methods across all model sizes.
- **Financial:** While the performance gap narrows for some models (particularly gpt-4o-mini), CAMA still maintains a clear lead.

These dataset-specific results demonstrate CAMA's versatility and robustness across different domains and data complexities.

5.5 Ablation Study

To evaluate the importance of each component in our proposed architecture, we conducted an ablation study using the financial dataset and the llama3-8b model. Table 2 presents the results.

The ablation study reveals several crucial insights:

Table 2: Ablation Study on financial dataset using llama3-8b

Configuration	Accuracy (%)
Full Pipeline	88.30
w/o Refactor	20.51
w/o Break Down	23.08
w/o Compile	69.23

- **Component Interdependence:** Removing any single component results in a significant performance decrease, with accuracy dropping by at least 19 percentage points (in the case of removing Compile). This demonstrates the interdependence of all components in our pipeline.
- **Critical Components:** The Refactor and Break Down components appear most critical, with their removal causing accuracy drops more than 67.8 and 65.2 percentage points, respectively. Refactor's importance stems from its context-aware filtering that provides essential feature representations, while Break Down enables focused analysis of individual features, reducing cognitive load on LLMs. This aligns with our design philosophy of improving data representation and detailed analysis before synthesis.
- **Compile Importance:** While less impactful than Refactor and Break Down, the Compile step still contributes significantly to the overall performance, improving accuracy by 19.07 percentage points.
- **Pipeline Robustness:** The substantial accuracy drop when removing any component indicates that each step is crucial for optimal performance. This justifies the complexity of our full pipeline and highlights the importance of maintaining all components for best results.

5.6 Discussion

The superior performance of CAMA across different model sizes and metrics underscores the effectiveness of our adaptive cognitive architecture. Several factors contribute to these outcomes:

(1) **Effective use of memory modules:** Our method's structured use of Procedural, Episodic, Semantic, and Working Memory ensures efficient retention and utilization of relevant information throughout the analysis process.
(2) **Feature engineering-inspired approach:** The Refactor, Break Down, and Compile steps allow our method to handle complex scenarios and evolving data effectively, as evidenced by the significant performance drops when any of these components are removed.
(3) **Adaptability across model sizes:** CAMA's consistent performance improvements across different model sizes (from llama-3.2-1b to llama3-70b) demonstrate its ability to effectively leverage the strengths of various LLMs while providing additional guidance and structure.
(4) **Performance-token trade-off:** While CAMA's approach consumes more tokens due to feature-specific context preservation and independent reasoning chains, this design choice is justified by the significant performance gains across all metrics.

The ablation study further confirms the importance of each component in our pipeline, demonstrating that the full architecture is necessary to achieve optimal performance.

6 CONCLUSION

We presented CAMA, an adaptive cognitive architecture for monitoring ML models using LLMs. Leveraging multiple memory types, our method automates monitoring and reporting, providing accurate and actionable insights. Experimental results demonstrate CAMA's superior performance across various metrics, model sizes, and datasets. With llama3-70b, CAMA achieved 92.3% ± 1.5% accuracy, outperforming the next best method by 33.3 percentage points. For llama3-8b, CAMA's accuracy (90.6% ± 2.3%) surpassed the next best method by 44.5 percentage points. Even with llama-3.2-1b, CAMA showed robust performance (55.6% ± 8.1% accuracy). The three-step process of Refactor, Break Down, and Compile enables efficient evaluations, adapting to diverse data drift scenarios. Our ablation study highlights the necessity of each component in our pipeline. CAMA's consistent superior performance across Healthcare, Eligibility, and Financial datasets demonstrates its versatility and adaptability. This, combined with its effectiveness across various model sizes, positions CAMA as a powerful tool for enhancing ML model monitoring in production environments, contributing to more reliable and transparent AI systems.

7 LIMITATIONS AND FUTURE WORK

While CAMA demonstrates significant improvements in ML model monitoring, our current study has some limitations. The evaluation was conducted on a specific set of datasets and LLM architectures, which may limit the generalizability of our findings. Additionally, the computational requirements and processing times of our method in comparison to simpler approaches warrant further investigation. Future research should focus on expanding the evaluation of CAMA across a broader range of LLM architectures, monitoring scenarios, and domains to establish wider applicability. Investigating techniques to optimize the implementation for reduced computational overhead and processing time could enhance CAMA's efficiency. These advancements would contribute to making CAMA more robust and adaptable for diverse real-world ML monitoring scenarios, ultimately leading to more reliable ML systems.

ACKNOWLEDGMENTS

We thank Lenovo for providing the technical infrastructure to run the experiments in this paper. This work was partially supported by Lenovo and Intel® as part of the Lenovo AI Innovators University Research program, by Spanish Ministry of Science (MICINN), the Research State Agency (AEI) and European Regional Development Funds (ERDF/FEDER) under contracts PID2019-107255GB-C22 and PID2021-126248OB-I00, and by the Generalitat de Catalunya (AGAUR) under contract 2021-SGR-00478.

REFERENCES

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901.

[2] Justin Cosentino, Anastasiya Belyaeva, Xin Liu, Nicholas A. Furlotte, Zhun Yang, Chace Lee, Erik Schenck, Yojan Patel, Jian Cui, Logan Douglas Schneider, Robby Bryant, Ryan G. Gomes, Allen Jiang, Roy Lee, Yun Liu, Javier Perez, Jameson K. Rogers, Cathy Speed, Shyam Tailor, Megan Walker, Jeffrey Yu, Tim Althoff, Conor Heneghan, John Hernandez, Mark Malhotra, Leor Stern, Yossi Matias, Greg S. Corrado, Shwetak Patel, Shravya Shetty, Jiening Zhan, Shruthi Prabhakara, Daniel McDuff, and Cory Y. McLean. 2024. Towards a Personal Health Large Language Model. arXiv:2406.06474 [cs.AI] https://arxiv.org/abs/2406.06474

[3] Bradley Eck, Duygu Kabakci-Zorlu, Yan Chen, France Savard, and Xiaowei Bao. 2022. A monitoring framework for deployed machine learning models with supply chain examples. In *2022 IEEE International Conference on Big Data (Big Data)*. 2231–2238. https://doi.org/10.1109/BigData55660.2022.10020394

[4] Florian Heinrichs. 2023. Monitoring Machine Learning Models: Online Detection of Relevant Deviations. arXiv:2309.15187 [cs.LG] https://arxiv.org/abs/2309.15187

[5] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring Massive Multitask Language Understanding. arXiv:2009.03300 [cs.CY] https://arxiv.org/abs/2009.03300

[6] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2024. Large language models are zero-shot reasoners. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS'22)*. Curran Associates Inc., Red Hook, NY, USA, Article 1613, 15 pages.

[7] Zachary C. Lipton, Yu-Xiang Wang, and Alex Smola. 2018. Detecting and Correcting for Label Shift with Black Box Predictors. arXiv:1802.03916 [cs.LG] https://arxiv.org/abs/1802.03916

[8] Scott M. Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 4768–4777.

[9] Stephan Rabanser, Stephan Günnemann, and Zachary C. Lipton. 2019. Failing loudly: an empirical study of methods for detecting dataset shift. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, Article 125, 13 pages.

[10] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: language agents with verbal reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS'23)*. Curran Associates Inc., Red Hook, NY, USA, Article 377, 19 pages.

[11] Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. 2024. Cognitive Architectures for Language Agents. arXiv:2309.02427 [cs.AI] https://arxiv.org/abs/2309.02427

[12] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL] https://arxiv.org/abs/2302.13971

[13] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. arXiv:2305.04091 [cs.CL] https://arxiv.org/abs/2305.04091

[14] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. Finetuned Language Models Are Zero-Shot Learners. arXiv:2109.01652 [cs.CL] https://arxiv.org/abs/2109.01652

[15] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903* (2022).

[16] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL] https://arxiv.org/abs/2210.03629

[17] Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, Denny Zhou, Swaroop Mishra, and Huaixiu Steven Zheng. 2024. Self-Discover: Large Language Models Self-Compose Reasoning Structures. arXiv:2402.03620 [cs.AI] https://arxiv.org/abs/2402.03620