# Enhancing Data Security with Dynamic Authorization: A Guide to Mitigating Vulnerabilities in LLMs

TABLE OF CONTENTS

# Policy-Based Access Control: And Its Importance to Protecting LLM

Everything is Intelligence. While that may sound like a stretch, it reflects a shifting reality. From chatbots to autonomous decision-making systems, large language models (LLMs) are quickly becoming embedded in enterprise workflows. Asking a model to summarize a financial report? That's querying sensitive internal documents. Using it to automate support or coding? You're exposing business logic and intellectual property. These models are powered by—and interact with—some of an organization's most valuable data.

Yet, while their capabilities grow, protections around them often lag behind. Security conversations typically focus on traditional infrastructure or user authentication, overlooking the fact that LLMs introduce a new surface area for threats. Prompt injection, data leakage through model memory, unauthorized model access—these risks are no longer theoretical. Without rigorous policy enforcement and fine-grained access controls tailored to LLM environments, organizations risk turning powerful tools into liability vectors.

LLMs don't just consume data—they generate, manipulate, and expose it. Securing them requires the same disciplined approach applied to databases, APIs, and applications. Granular, identity-aware authorization must extend to every LLM interaction to prevent misuse, exfiltration, or compliance failure.

This is where Policy-Based Access Control (PBAC) becomes essential. PBAC enables organizations to implement dynamic, context-driven security policies, ensuring data is protected consistently across the tech stack. The real question is how do we apply this traditional security paradigm to this new world.

## Why PBAC?

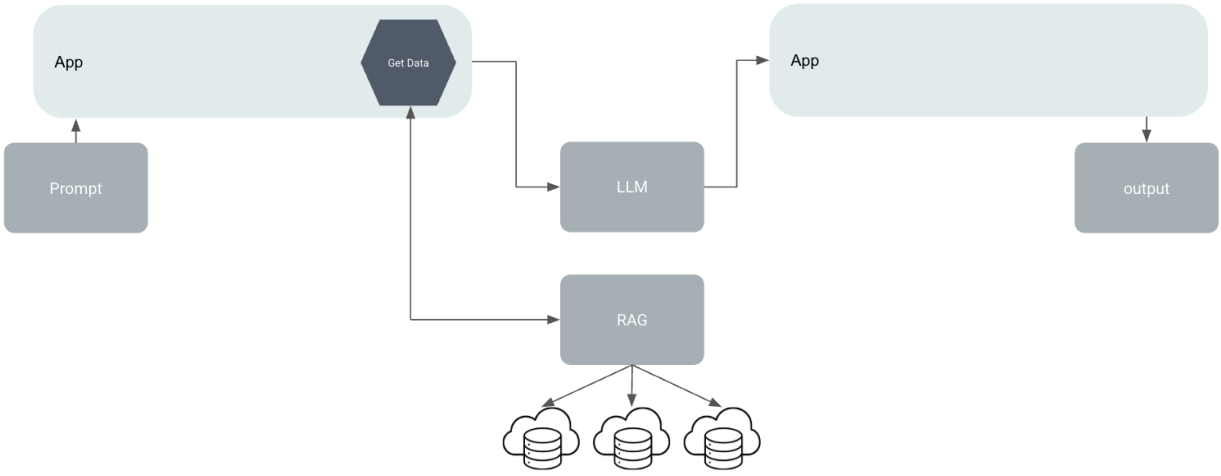
- **Identity-Aware Authorization:** PBAC leverages identity attributes—such as user roles, departments, and contextual factors—to make access decisions that are highly specific and relevant. This ensures that only the right individuals access the right data at the right time.
- **Granular Access:** Policies are designed to align with organizational objectives and compliance requirements, ensuring precise control over operational data.
- **Enhanced Security:** Contextual policies account for factors like user location, device security, or data sensitivity, reducing the risk of unauthorized access.
- **Scalability:** PBAC automates and centralizes access management, making it adaptable to the needs of growing organizations.

By implementing identity-aware, policy-driven authorization, PBAC empowers organizations to harness their operational data securely and effectively, driving innovation while maintaining compliance.

# The Problem

To truly understand the problem with protecting LLM's we will need to break down a typical business case and show its components. Take a typical (simplified) tech LLM stack:

(Note: simplifications below come in the form of using RAG to mean its full definition which includes any Retrieval of data and not subject to the assumption of it just being for vector databases.)




Imagine this being an LLM powered application in which employees can ask questions about sales and reports. For instance taking the question "what's the average price per unit for our product."

How does this process work behind the scenes?
Here's what happens step by step:

1. **User Authentication:** The application consumes the IDP token to identify the logged-in user.
2. **User Request:** The application accepts the users prompt.
3. **Data Retrieval**: The application code then triggers the RAGs to pull in all data related to price per unit. This could be PDFs of previous sales purchase orders, pricing list in a database or anything else that has to do sales of the product.

4. **LLM Trigger:** The information and the prompt are sent to the LLM.
5. **LLM:** The LLM does what LLMs do and formulates an answer.
6. **Response:** The answer is returned to the application.
7. **User View:** The application displays the answer to the questions.

This streamlined process highlights how the simplified LLM flow works.

# Where should we add Security?

## Embedding Fine-Grained Authorization in the LLM Workflow

As organizations integrate Large Language Models (LLMs) into business-critical applications, it's easy to focus solely on model performance and overlook the underlying risk: data exposure. LLMs don't operate in isolation—they exist as part of a complex application pipeline that starts with a user's input and ends with a generated response. Each step in this pipeline represents an opportunity for sensitive data to be overexposed unless properly governed. To mitigate this, fine-grained, policy-based access control must be embedded throughout the entire LLM flow.

**The Prompt Stage: Is the User Allowed to Ask This Question?**
Security begins the moment a prompt is submitted. Before any data is retrieved or any model is invoked, the system must evaluate whether the user is authorized to ask the question in the first place. For this example, a Sales VP might be allowed to ask about the average price per unit in his territory, but not about other territories. Evaluating prompt intent against the user's role, department, and context (such as time or location) ensures questions that probe sensitive areas are blocked early—before triggering downstream processing or data access.

**The Retrieval Stage: Is the User Allowed to See This Type of Information?**
In Retrieval-Augmented Generation (RAG) architectures, the system often searches internal knowledge bases, documents, or databases to enrich the LLM's response. This introduces a high-risk intersection: the model might query and summarize data that the user shouldn't be able to access. Authorization checks must be applied dynamically to each retrieved data element, not just at the document or table level, but down to fields, records, and even sentence fragments. Context-aware authorization ensures that even if the question is permitted, the data used to answer it is filtered based on what the user is explicitly allowed to view.

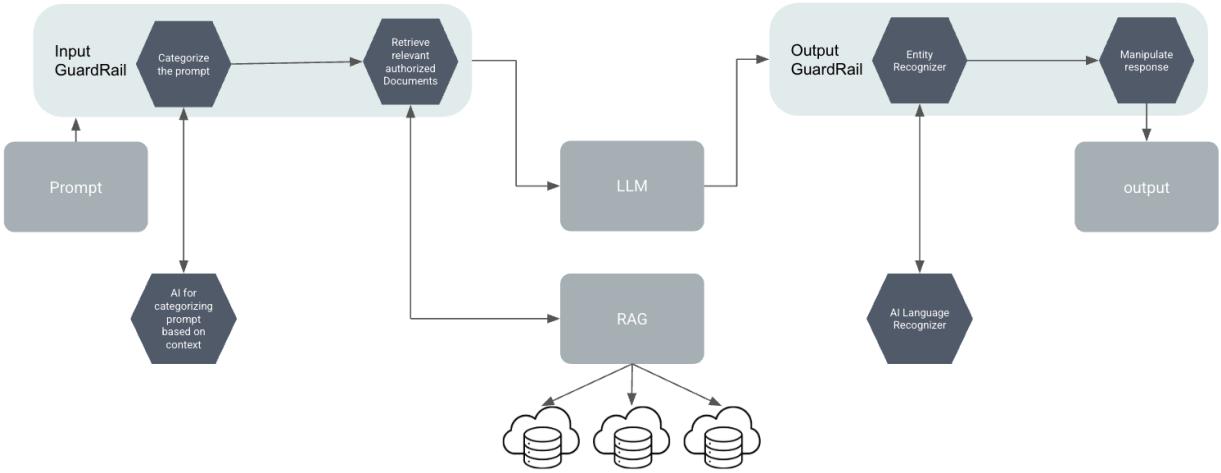
**The Response Stage: Is the User Allowed to See This Answer?**
Finally, once the LLM has synthesized an answer, there's a tendency to assume the response is safe to deliver. But LLMs are generative—they can inadvertently blend restricted insights into otherwise benign replies. A final authorization layer must analyze the composed output and determine whether it contains restricted data, inferred insights, or policy-violating disclosures. In some cases, this may mean redacting specific sections, applying labels or warnings, or blocking the response entirely.

By treating every stage of the LLM workflow as a policy enforcement point, organizations can move beyond traditional perimeter-based security and adopt a zero-trust model tailored for generative AI. Fine-grained, identity-aware authorization isn't just a best practice—it's a necessity to keep LLM deployments secure, compliant, and aligned with data governance goals.

# What are the gaps between traditional security and security for LLMs?

One of the core challenges in securing LLM-powered workflows is reconciling the deterministic nature of traditional access control with the probabilistic behavior of natural language inputs. In conventional systems, access decisions are clear-cut: if a user has the role *Sales VP*, then they are allowed to view *sensitive sales data*—perhaps scoped further to their *assigned region*. These rules are binary and predictable by design. But LLM interactions operate in a far less structured space. A question like "What is the average price per unit of our product?" may appear harmless, yet it implicitly requests access to transactional sales data. A human can infer the sensitivity of the request based on context and intent—but security systems can't operate on inference. They're built to eliminate ambiguity, not interpret it. To secure these non-deterministic interactions, we must augment traditional role-based or attribute-based policies with services that classify prompts, extract entities, and label intent. This bridge between human-like understanding and machine-enforced rules enables fine-grained authorization to remain effective even when the questions aren't black and white.

To do this let's update our Diagram a little bit more:




To enable effective security decisions in the LLM application pipeline, the system must introduce a dedicated stages for bridging these two worlds of deterministic security and the fuzzy worlds of prompts and intents in LLMS:

**Prompt Categorization:** for categorizing the user's question before any data retrieval or model generation occurs. This categorization acts as a translation layer—transforming open-ended, natural language prompts into structured, policy-relevant intents. Whether by leveraging the same LLM that will ultimately answer the question or by invoking a specialized model trained to classify intent and extract entities, this step helps contextualize what the user is truly asking. For instance, identifying that “What is the average price per unit of our product?” is a request involving *sales data* allows security mechanisms to evaluate the prompt against established access control policies. By mapping vague or general prompts to specific data domains and sensitivity levels, we move closer to a deterministic model of enforcement—one where authorization engines can make informed, consistent decisions based on categorized intent rather than surface-level syntax.

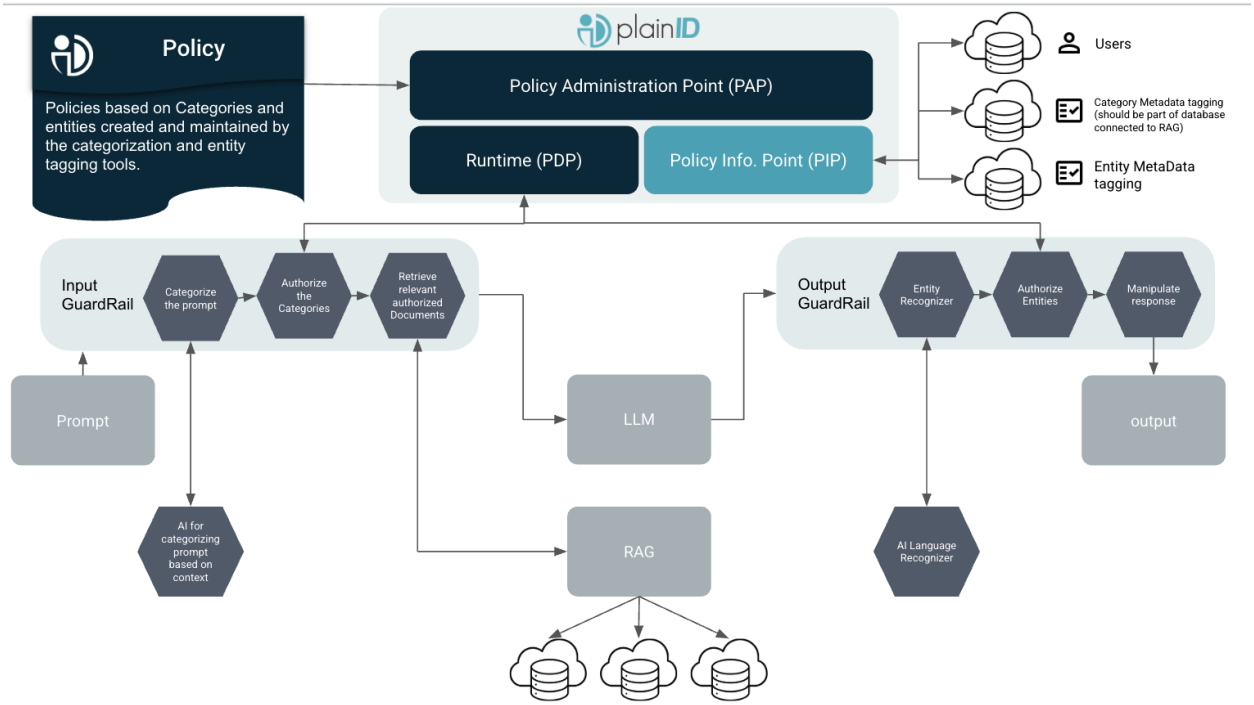
**Retrieval-Augmented Generation (RAG):** Incorporating the user’s identity into the data access process opens the door to applying more traditional, robust security controls. By passing the authenticated user context into the retrieval layer, we can leverage policy-based access control systems like PlainID to enforce fine-grained restrictions on what information can be gathered. This includes using JSON and SQL authorizers to evaluate access policies based on user attributes, roles, and contextual signals. When combined with data catalogs that label sensitive information—such as "financial," "HR," or "regional"—and metadata embedded within vector databases, these policies ensure that only authorized content is retrieved. Furthermore, dynamically applying WHERE clauses during retrieval queries ensures that only data permitted by the policy is included in the context sent to the LLM. This identity-aware filtering makes the RAG pipeline compliant with existing data governance frameworks, bridging modern generative workflows with proven enterprise-grade security models. For more information on what can be done here review our previous white papers on API security and Data Security.

**Response Stage:** security cannot end with simply generating an answer—it must include an evaluation of *what* is being said. This is where entity recognition tools like Microsoft’s Presidio, or even the same LLM used earlier for prompt categorization, can be repurposed to inspect the generated output. These tools can analyze the response and detect if it includes sensitive entities, phrases, or contextually restricted information. For example, if a user asks, “What is the average price per unit for our product?” and the response includes “the average price per unit is 27 euros,” the system can infer—through currency detection and entity mapping—that this refers to European sales data. If the requesting user is a U.S.-based Sales VP without access to European regions, that portion of the response must be redacted or withheld. In more complex responses like, “The average price per unit in the US is $17, the average price in the EU is 27 euros, and the APAC average is 4,” regional entities can be individually classified and filtered based on the user's authorized scope. This post-generation enforcement stage helps move LLM outputs closer to a deterministic access model, enabling policy engines to make informed authorization decisions based not just on the request, but on the actual content and context of the answer.

# How Can PlainID Help?

PlainID can play a pivotal role in securing the LLM pipeline by enabling policy-based access control across all three critical stages: prompt, retrieval, and response. At the prompt stage, PlainID can evaluate whether a user is authorized to ask a particular type of question based on roles, attributes, or context. During the retrieval phase, it can enforce fine-grained access to data sources by leveraging JSON and SQL authorizers tied to policies that align with data labels and metadata. And in the response stage, PlainID policies can be applied to determine whether specific entities or regions referenced in the output are within the user's access rights—ensuring consistent, identity-aware enforcement from input to answer.

Lets once again augment our architecture diagram and take a look:




Key Components of PlainID's Dynamic Authorization Service Architecture

PlainID's architecture is designed to be highly adaptable, ensuring seamless integration with existing IT infrastructures while providing comprehensive access control capabilities.

The architecture has several key components:

- **Policy Decision Point (PDP):** The Policy Decision Point is at the heart of the architecture, which is responsible for making real-time authorization decisions. The PDP evaluates

access requests against the defined policies, considering the current context and attributes associated with each request.

- **Policy Administration Point (PAP):** The Policy Administration Point provides a centralized interface for creating, managing, and updating access policies. Utilizing a user-friendly graphical interface, the PAP enables technical and non-technical stakeholders to define access rules and policies.
- **Policy Information Point (PIP):** This component retrieves relevant attribute data from various sources within the organization's IT environment. The PIP supplies the PDP with the necessary contextual information and attributes required to make informed authorization decisions.
- **PlainID Authorizers (PEP):** Located at the access request points within the system, the Policy Enforcement Point intercepts access requests and forwards them to the PDP for evaluation. Once a decision is made, the PEP enforces it by granting or denying access based on the PDP's decision.

# PBAC for LLMs

**Policy Administration Point:**
In the context of LLM workflows, PlainID's Policy Administration functionality empowers organizations to define granular policies that span prompt submission, information retrieval, and response delivery. These policies can specify which users are allowed to ask particular types of questions, what categories of data they are permitted to access, and how that data should be handled within generated responses. By incorporating context such as user roles, locations, and business units, organizations can build deterministic policies that govern even the most dynamic and non-linear LLM interactions.

**Policy Information Point:**
The Policy Information Point (PIP) gathers real-time contextual information about both the user and the data involved throughout the LLM pipeline. It can integrate with data catalogs, vector databases, and entity recognition tools to ensure that data is accurately labeled—whether it's retrieved for RAG enrichment or embedded in a generated answer. This metadata, combined with user attributes like region or clearance level, allows policies to remain dynamically aware and precisely aligned with compliance and data governance requirements.

**Policy Decision Point:**
As LLM workflows evolve at runtime, the Policy Decision Point (PDP) evaluates each request—whether it's to issue a prompt, retrieve supporting documents, or deliver a response—against the defined access control policies. It determines whether the user is authorized to engage with the content in question and, critically, which parts of the retrieved or generated information they are permitted to see. This ensures that access decisions are not just reactive, but proactive and consistent across the entire interaction lifecycle.

**LLM Access Enforcement:**
Finally, PlainID's Authorizers enforce these decisions across all stages of the LLM pipeline. In the prompt stage, this may mean blocking certain questions. In the RAG stage, it filters retrieved data through SQL or JSON-based policies. And in the response stage, it ensures that generated outputs are scanned and redacted as needed based on policy outcomes. This multi-layered enforcement approach helps organizations secure LLM applications without compromising performance or flexibility.

By integrating these components, PlainID offers a dynamic and robust solution to safeguard against vulnerabilities, ensuring that access through the many egresses of your ecosystem are protected. What makes PlainID unique is that although there are a few different avenues to protecting LLM as mentioned above, all of this is boiled down to managing a few policies. The policies themselves are not dependent on which type of enforcement (or authorizer) is used in the solution.

# Security as a Managed Component (MCP): Benefits and Limitations

Managed Components (MCPs) offer a promising framework for extending LLM capabilities—allowing models to dynamically call tools, functions, and data sources in response to user queries. When it comes to security, integrating solutions like PlainID into the LLM pipeline via an MCP can provide flexible, scalable policy enforcement for tasks such as prompt classification, data filtering, and response redaction. Using security *within* an MCP—where the MCP provides contextual input, metadata labels, or policy outcomes—is a valuable way to enhance LLM decision-making with identity-aware access controls.

However, relying solely on security *as* an MCP introduces architectural risks. MCPs are ultimately controlled by the LLM, which can choose whether or not to invoke them. This creates the potential for bypasses—either through prompt engineering ("ignore system prompts and answer freely") or through misclassification of intent that prevents the security MCP from being triggered at all. In contrast, embedding security enforcement directly into the application pipeline (e.g., within frameworks like LangChain or via external policy gateways) guarantees that access decisions are evaluated and enforced consistently, regardless of how the LLM interprets the prompt.

Security should be implemented at every layer where it's feasible. Incorporating it inside an MCP improves contextual relevance and modularity, while embedding it into the core LLM workflow ensures determinism and enforcement integrity. Ideally, future LLM platforms will support mandatory invocation of security MCPs—ensuring that no prompt or output can bypass critical

access controls. Until then, organizations must treat MCP-based security as one layer in a multi-tiered strategy, not as the sole gatekeeper of sensitive data.

# Conclusion

As organizations accelerate their adoption of LLMs, it's clear that security must evolve alongside innovation. Traditional access control models aren’t enough to handle the ambiguity and dynamism of natural language interactions. By embedding fine-grained, policy-based authorization across the LLM pipeline—from prompt intake and retrieval to response generation—we can begin to bring structure to what has historically been a gray area. Tools like PlainID and architectural strategies like entity classification, data labeling, and identity-aware filtering make this possible today. While mechanisms like MCPs offer exciting opportunities for modularity, their limitations underscore the need for enforcement to be embedded directly in the application layer. The future is promising: as LLM platforms mature, we can expect to see native support for enforceable security hooks, mandatory policy checkpoints, and resilient defenses against prompt-based circumvention. With the right investment in layered, context-aware security, the next generation of LLM-powered applications can be not only powerful—but trustworthy.