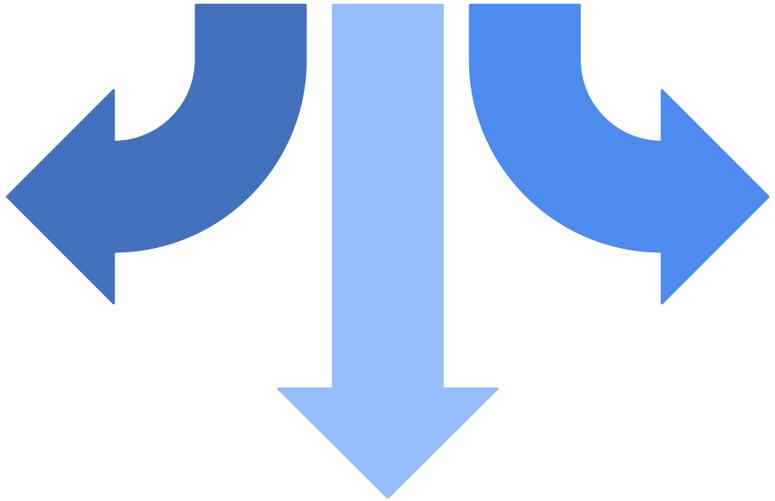
Protection using Prompt Shield

Design for Integrating Prompt Shield with Caching & Deduplication

Since your system involves **Azure LLM** and a **UI service**, we need to strategically place the **Prompt Shield** to protect against prompt injection and adversarial queries while also optimizing performance by reducing redundant checks.

Where should the Prompt Shield be placed for optimal protection and performance?



Azure LLM Integration

Placing the shield here protects the language model from adversarial queries, ensuring the integrity of AI outputs.

UI Service Protection

This choice focuses on safeguarding user interactions, preventing malicious inputs from affecting the system.

Performance Optimization

Strategically reducing redundant checks enhances system efficiency while maintaining security.

1. Architectural Placement of Prompt Shield

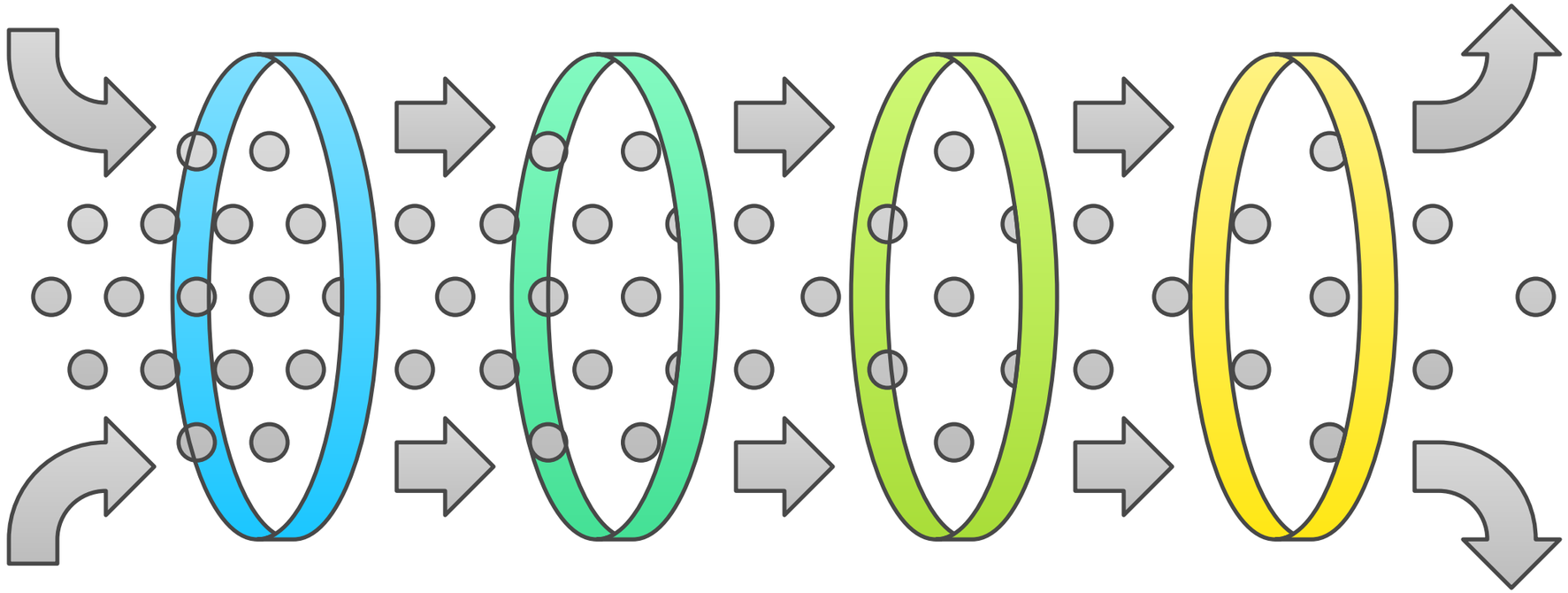
Components Involved:

- **UI Service** → Sends user queries to backend.
- **Azure LLM** → Generates responses for user queries.
- **Prompt Shield** → Filters malicious or unwanted prompts.
- **Prompt Shield Wrapper with Caching** → Avoids unnecessary duplicate checks.

Proposed Placement:

- Place the **Prompt Shield Wrapper before** queries reach **Azure LLM**.
- Ensure **deduplication logic** is implemented within the wrapper.
- Use **cache-based similarity detection** to avoid rechecking prompts with Prompt Shield.

Optimizing Query Processing



Send to Backend

Queries are transmitted to backend systems

Filter Prompts

Malicious prompts are filtered out

Cache and Deduplicate

Redundant checks are avoided

Generate Responses

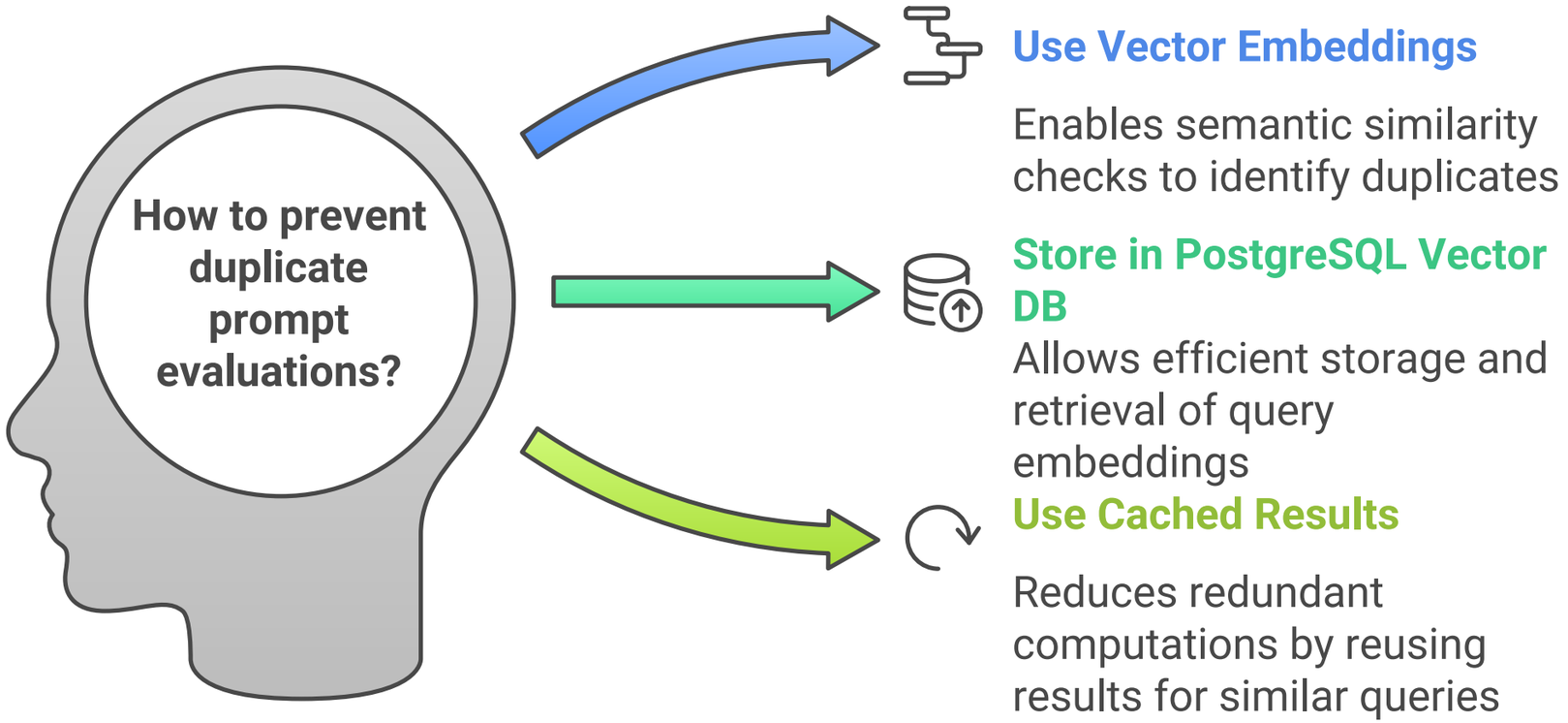
LLM creates responses for queries

2. Design Flow with Caching and Similarity Detection

3. Implementing Similarity-Based Caching

To prevent **duplicate prompt evaluations**, we can:

- Use **vector embeddings** (e.g., **Azure AI Search**) to check **semantic similarity**.
- Store embeddings of past queries in **PostgreSQL Vector DB**. (pgvector)
- If a new query has **high cosine similarity** (e.g., >90%) with an existing query → Use cached result instead of hitting Prompt Shield.

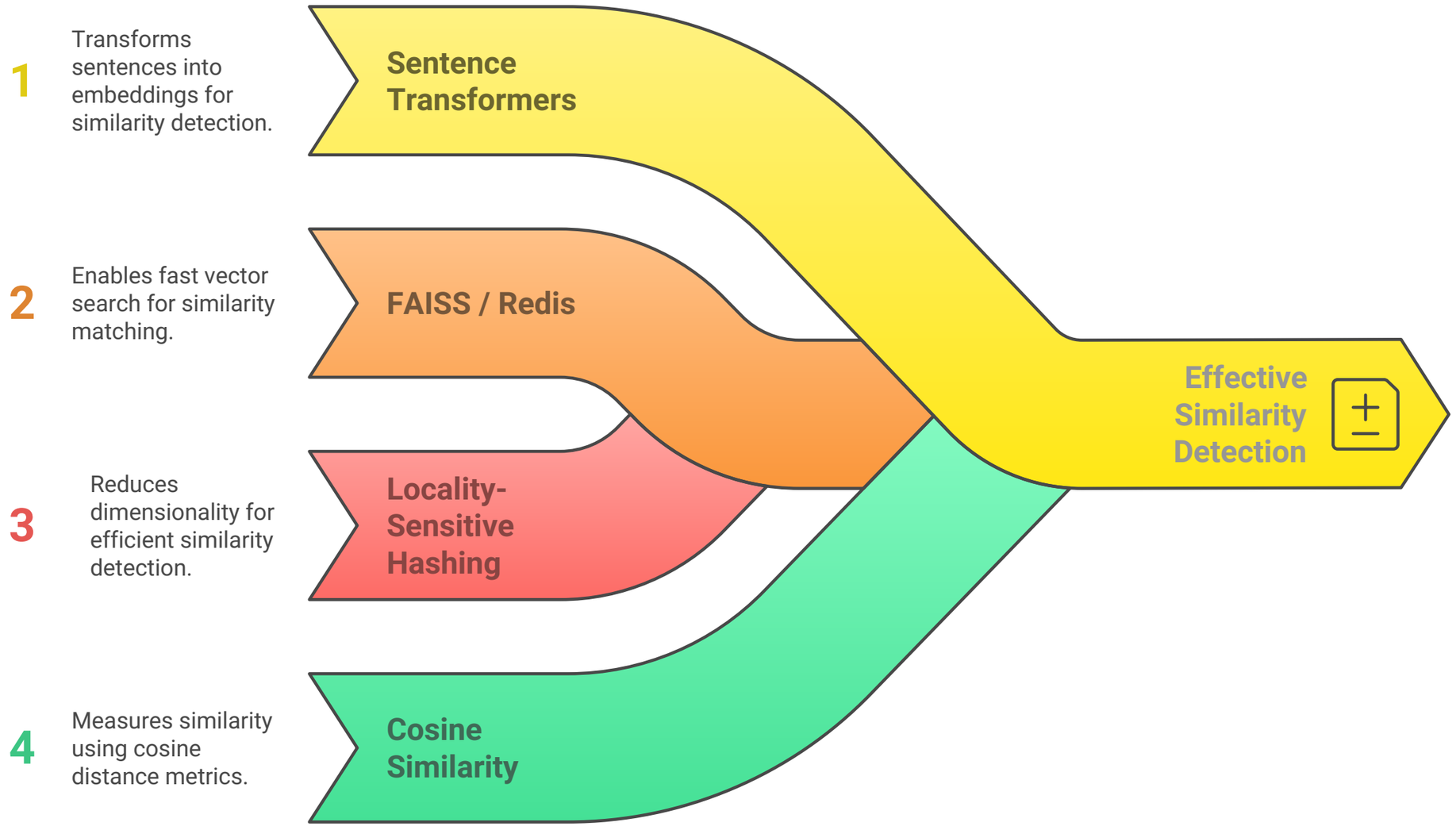


Tech Stack for Similarity Detection:

- **Sentence Transformers (SBERT)**

- **Locality-Sensitive Hashing (LSH)**
- **Cosine Similarity via Scikit-learn / Azure AI Search**

Unified Similarity Detection Framework

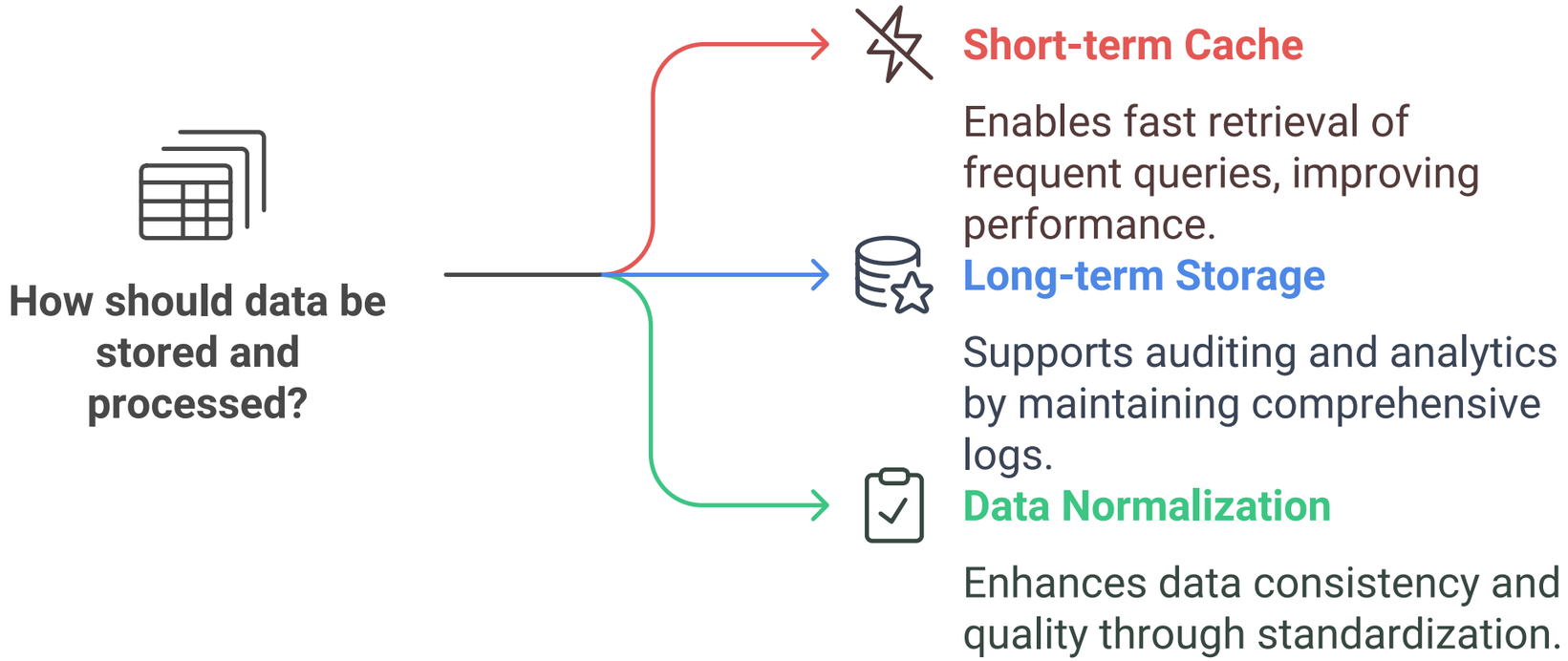


4. Cache Implementation Strategy

- **Short-term Cache (MemoryDB/Redis)**: Store frequent queries for fast retrieval.
- **Long-term Storage (SQL/NoSQL)**: Maintain logs for auditing and analytics.

Normalization can involve:

- Lowercasing
- Removing special characters
- Lemmatization

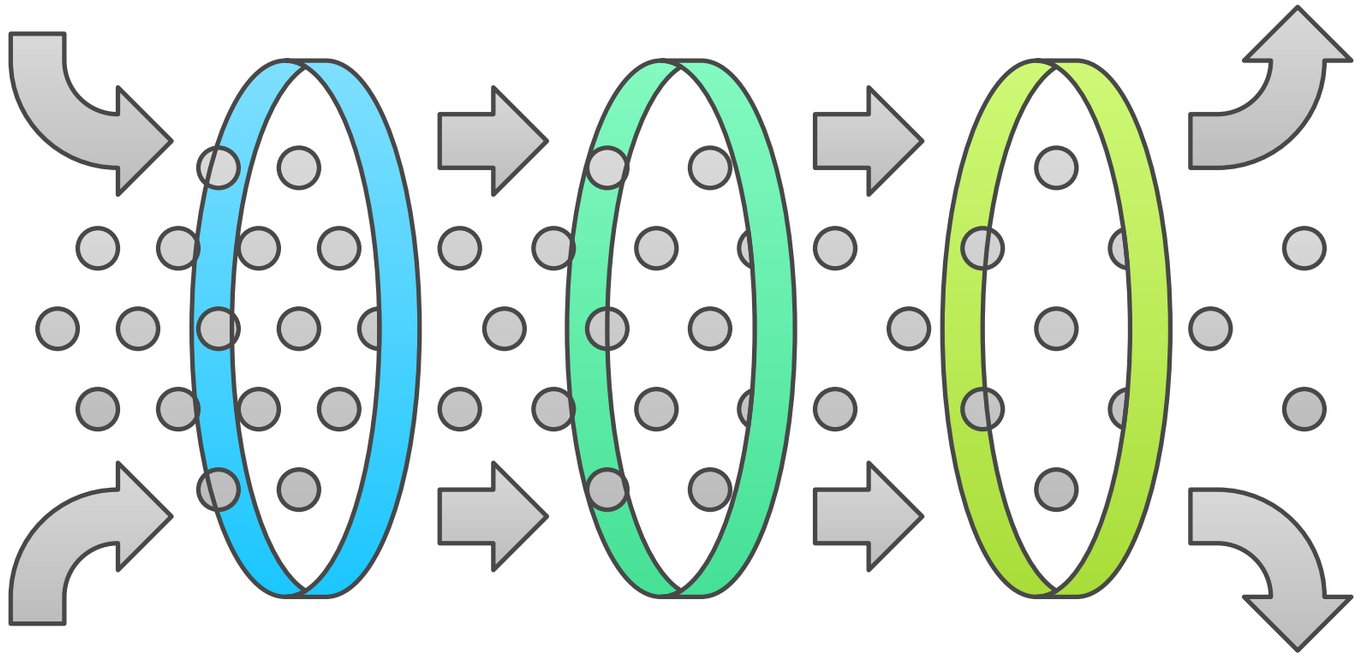


5. Security Considerations

- Ensure **Prompt Shield can block adversarial variations** of cached prompts.

- **Log all bypassed queries** for later review.

Enhancing Prompt Security



Block Adversarial Variations

Shielding against modified prompts

Rate-limit Requests

Controlling request flow

Log Bypassed Queries

Recording exceptions for review

Key Benefits of This Design

✅ **Reduces Load**: Avoids redundant Prompt Shield checks for similar queries.✅ **Improves Performance**: Cached responses ensure lower latency.✅ **Enhances Security**: Still maintains strict filtering for adversarial inputs.✅ **Scalable**: Can extend cache expiration or increase similarity thresholds dynamically.

Unified System Efficiency