Here is the rewritten **Orgo v2 Examples Utilisation** document, incorporating the requested corrections:

## **Orgo v2 Examples Utilisation**

This section provides detailed examples of how Orgo can be utilized across various scenarios. It now includes expanded multi-organization workflows with Redis/RabbitMQ task queue integration and demonstrates how escalations are managed in distributed setups.

### **1. Purpose**

**Objective:**

* Demonstrate practical applications of Orgo in diverse organizational contexts.
* Illustrate the use of Redis/RabbitMQ for task queue management and distributed escalation handling.

**Outcome:**

* Clear, actionable examples showcasing Orgo’s capabilities in handling multi-organization workflows and distributed task escalations.

### **2. Multi-Organization Workflows**

#### **Example 1: Maintenance Requests Across Schools and Hospitals**

**Scenario:** A maintenance issue arises at a school and escalates to district-level management if unresolved. Similarly, hospital maintenance issues escalate to the operations manager.

**Workflow Steps:**

1. **Trigger:**
   * A school secretary emails maintenance@school.org or a hospital staff member emails maintenance@hospital.org.

**Task Queuing with Redis:** import redis

queue = redis.StrictRedis(host="localhost", port=6379)

def enqueue\_task(task\_data):

queue.rpush("task\_queue", task\_data)

* + Incoming requests are queued asynchronously for processing.

1. **Routing Logic:**

Rule engine routes tasks based on organizational type:  
 - condition: "organization == 'school'"

action:

route\_to: "district\_maintenance@school.org"

escalate\_after: "2 hours"

- condition: "organization == 'hospital'"

action:

route\_to: "ops\_manager@hospital.org"

escalate\_after: "1 hour"

1. **Escalation Handling in a Distributed Setup:**

Escalation tasks are pushed to RabbitMQ for durability:  
 import pika

connection = pika.BlockingConnection(pika.ConnectionParameters("localhost"))

channel = connection.channel()

channel.queue\_declare(queue="escalation\_queue", durable=True)

def escalate\_task(task\_data):

channel.basic\_publish(exchange="",

routing\_key="escalation\_queue",

body=task\_data,

properties=pika.BasicProperties(delivery\_mode=2))

1. **Resolution:**
   * Task status is updated in Redis, and a summary email is sent upon resolution.

#### **Example 2: Distributed Escalation for IT Support**

**Scenario:** An IT issue at a corporate office escalates to regional and national IT leads if unresolved within designated timeframes.

**Workflow Steps:**

1. **Trigger:**
   * Employee emails it\_support@corporate.com with a system issue.
2. **Task Queue Integration:**

Task is enqueued in Redis for immediate processing:  
 queue.rpush("it\_tasks", {"task\_id": 101, "status": "pending"})

1. **Escalation Across Regions with RabbitMQ:**

Regional IT leads are notified for unresolved tasks:  
 - condition: "task\_unresolved\_time > '4 hours'"

action:

notify: "regional\_lead@corporate.com"

- condition: "task\_unresolved\_time > '8 hours'"

action:

notify: "national\_lead@corporate.com"

RabbitMQ ensures reliable escalation notifications:  
 def notify\_escalation(recipient, task\_id):

message = f"Task {task\_id} escalated to {recipient}"

channel.basic\_publish(exchange="",

routing\_key="escalation\_queue",

body=message)

1. **Monitoring Escalation Chain:**
   * Real-time dashboards track escalation status using Redis TTL (Time-To-Live) for task expiry.

### **3. Dynamic Escalation Handling**

Redis and RabbitMQ are used to manage escalations dynamically:

* **Redis for Immediate Actions:**
  + Stores task data with TTL to monitor time-based triggers.
* **RabbitMQ for Persistent Notifications:**
  + Guarantees delivery of escalated tasks to higher authorities.

#### **Escalation Workflow Example:**

Task data is stored in Redis with an expiration timer:  
 queue.set("task\_123", "pending", ex=7200) # Expires in 2 hours

If the timer expires without resolution, RabbitMQ escalates:  
 escalate\_task({"task\_id": 123, "escalate\_to": "manager@organization.com"})

### **4. Use Cases for Multi-Organization Workflows**

#### **Use Case 1: Regional Maintenance Network**

* **Scenario:** Maintenance requests from multiple schools are tracked regionally.
* **Solution:** Redis queues tasks for each school, and RabbitMQ escalates unresolved tasks to district managers.

#### **Use Case 2: Healthcare System Workflow**

* **Scenario:** Lab results are delayed, triggering escalations.
* **Solution:** Redis monitors task resolution times, and RabbitMQ escalates delays to supervisors and hospital administrators.

#### **Use Case 3: Distributed Corporate IT Support**

* **Scenario:** IT tasks span multiple regions.
* **Solution:** Redis handles initial task distribution, while RabbitMQ manages escalations across regional and national levels.

### **5. Future Enhancements**

* **Integration with Advanced Analytics:**
  + Use Elastic Stack to monitor task queues and escalations in real time.
* **AI-Powered Escalation Predictions:**
  + Analyze task patterns to predict potential escalations and proactively resolve issues.

### **6. Summary**

This section expands on Orgo's multi-organization workflows, integrating Redis for caching and RabbitMQ for persistent task queue management. The examples demonstrate effective escalation handling in distributed setups, ensuring reliability and efficiency across diverse organizational contexts. Let me know if further refinements are required.