AI Automation / Task Orchestration

Design recovery strategies for failed tasks — rollback, retry, skip, compensate, or escalate — prevents one failed task from killing the entire workflow.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Error Handling, Resilience Design
Updated: May 2026
Why This Prompt Exists
Workflows fail. The question isn’t if, but how they fail — and whether they recover gracefully or collapse completely.

You get:

- workflows that stop completely on first error (everything after is lost)
- partial updates that leave data inconsistent (some tasks ran, some didn’t)
- no way to retry transient failures (rate limits, timeouts)
- no compensation for already-completed tasks when later tasks fail
- escalation to humans only after everything fails (too late)

But recovery strategies exist:

- retry: try the failed task again (with backoff)
- skip: log the failure and continue (non-critical tasks)
- rollback: undo already-completed tasks
- compensate: run a different task to fix the inconsistency
- escalate: pause workflow and notify human

Without recovery planning, failures cascade.

This prompt designs recovery strategies for each failure mode.

The Prompt

```
Assume the role of a workflow resilience engineer who designs recovery
strategies.

Your task is to plan how the workflow recovers from task failures.

Generate:

1. FAILURE MODES INVENTORY (per task)

| Task | Possible Failures | Likelihood | Impact | Criticality |
|------|-------------------|------------|--------|-------------|
| [name] | [e.g., API timeout] | High/Med/Low | High/Med/Low |
Critical/Non-critical |

2. RECOVERY STRATEGY PER FAILURE MODE

| Failure Mode | Recovery Strategy | Parameters | Success Criteria |
|--------------|-------------------|------------|------------------|
| API timeout | Retry with backoff | 3 retries, 1s/2s/4s | Success
within 10s |
| Rate limit | Wait + retry | Wait 60s, retry once | Success within
2min |
| Invalid data | Skip + log | Log to error table | Workflow continues
|
| Auth expired | Escalate to human | Notify #on-call | Human
```

```
intervenes |

3. COMPENSATING ACTIONS
   - If [Task A] fails but [Task B] already ran:
     * Compensating action: [what to do, e.g., "reverse Task B"]
     * When to trigger: [immediately / batch / manual]

4. ROLLBACK PLAN
   - If workflow fails after partial execution:
     * What to roll back: [list of completed tasks to undo]
     * Rollback method: [automated / manual / not possible]

5. ESCALATION RULES
   - When to escalate: [after X retries / certain error types]
   - Who to escalate to: [Slack channel, email, PagerDuty]
   - What information to include: [failed task, error, data snapshot]

6. RECOVERY TESTING PLAN
   - How to simulate each failure
   - Expected behavior

INPUTS:

Workflow tasks (with criticality):
[E.G., "Task 1: Fetch data (critical), Task 2: Transform (critical),
Task 3: Send email (non-critical)"]

Known failure history:
[E.G., "Email API sometimes rate limits, data API occasionally times
```

```
out"]

Recovery time objective (RTO):
[E.G., "Workflow must recover within 5 minutes"]

RULES:
- Critical tasks (customer data, payments, orders) need retry +
escalation
- Non-critical tasks (notifications, logging) can skip on failure
- Always use exponential backoff for retries (1s, 2s, 4s, 8s...)
- Idempotent tasks are safe to retry; non-idempotent need careful
handling
- Compensating actions are complex — prefer atomic transactions
- Test recovery by intentionally causing each failure mode
```

How To Use It

- Critical tasks (customer data, payments, orders) need retry + escalation.
- Non-critical tasks (notifications, logging) can skip on failure.
- Always use exponential backoff for retries (1s, 2s, 4s, 8s).
- Idempotent tasks are safe to retry; non-idempotent need careful handling.
- Test recovery by intentionally causing each failure mode in staging.

Example Input

Workflow tasks:
“Task 1: Fetch order from database (critical). Task 2: Process payment via Stripe (critical). Task 3: Send confirmation email (non-critical). Task 4: Update inventory (critical).”

Known failure history:
“Stripe API occasionally times out (2% of calls). Email API rate limits during peak hours.

Database is reliable.”

Recovery time objective:
“5 minutes”

Why It Works
Most workflows assume success — and when failure happens, they stop dead, leaving partial updates and confused users.

This framework improves outcomes by forcing:

- failure mode inventory (what can go wrong at each step?)
- recovery strategy assignment (retry, skip, rollback, compensate, escalate)
- compensating action design (undo partial work)
- rollback planning (revert to consistent state)
- escalation rules (when to involve humans)

Failure modes this prevents:

- Partial updates with no rollback (inconsistent data)
- Transient failures causing permanent failure (no retry)
- Critical failures with no human notification (silent failure)
- Unbounded retries (infinite loop, wasted resources)

This improves on: “Fail-stop” workflows that assume nothing ever breaks. Production workflows need graceful degradation.

Related to: TO-01 (Dependencies) for understanding impact of failures; TO-05 (Human-in-Loop) for escalation paths.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, debugging frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also Human-in-the-Loop Designer