

# AI TICKET PROCESSOR - TECHNICAL OVERVIEW (QUICK REFERENCE)

**For:** Engineering Team Review

**Version:** 2.2

**Status:** Production-Ready MVP

**Last Updated:** November 6, 2025

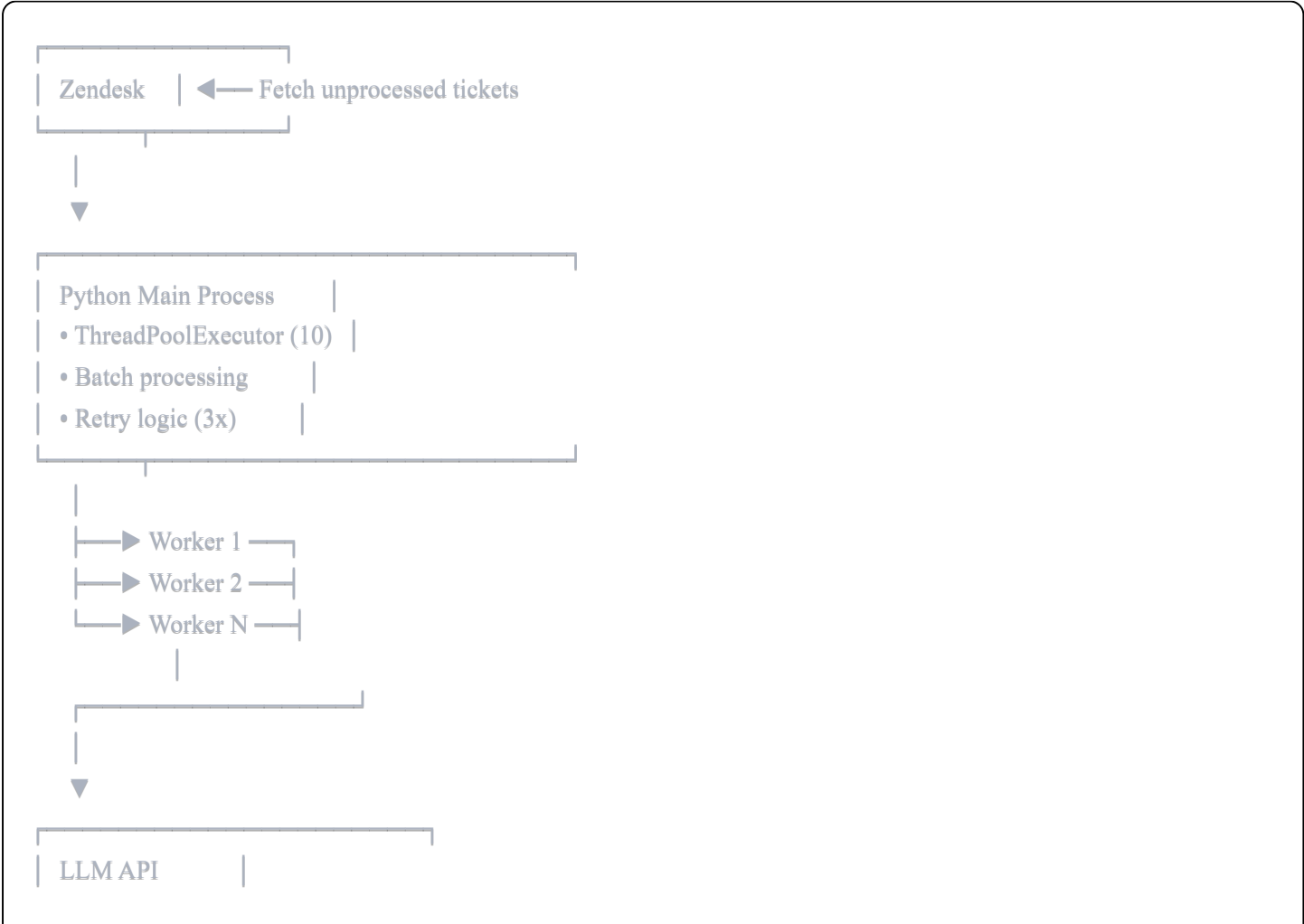
## WHAT IS THIS?

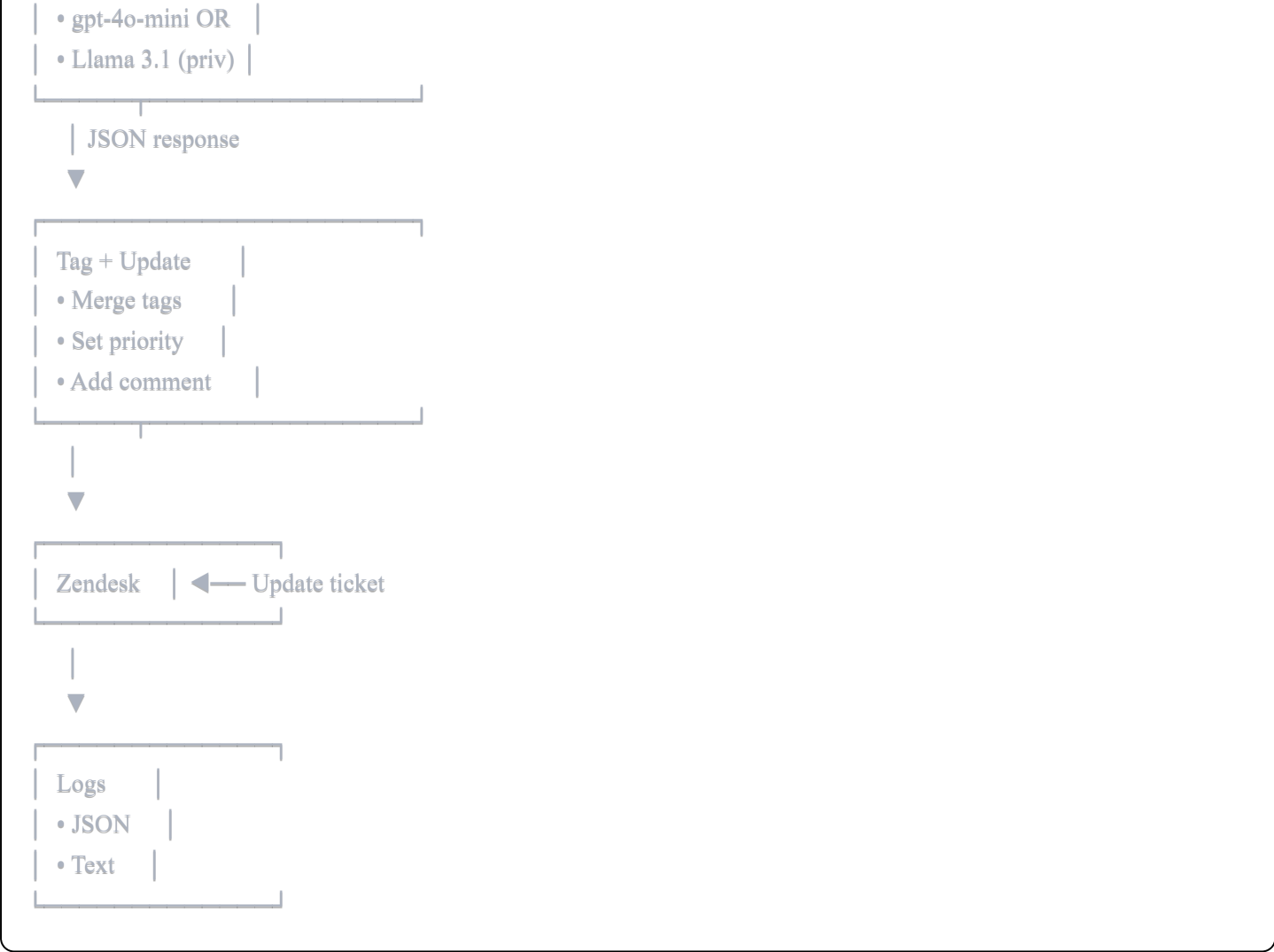
An automated system that reads Zendesk tickets, analyzes them with AI, and updates them with:

- Smart tags (bug, feature, refund, etc.)
- Priority levels (high, medium, low)
- Internal AI analysis comments
- Sentiment detection

**Result:** 99.1% reduction in manual triage time (5 min → 3.5 sec per ticket)

## SYSTEM ARCHITECTURE (ONE PAGE)





## TECH STACK

| Component   | Technology             | Why                            |
|-------------|------------------------|--------------------------------|
| Language    | Python 3.11+           | Best for ML/AI, rich libraries |
| LLM         | gpt-4o-mini            | Fast, cheap (\$0.001/ticket)   |
| Concurrency | ThreadPoolExecutor     | Simple, effective for I/O      |
| API Client  | requests + retry       | Industry standard              |
| Dashboard   | Streamlit              | Quick to build, easy to deploy |
| Automation  | Windows Task Scheduler | Zero cost, reliable            |
| Logging     | JSON + text            | Structured + human-readable    |

## CODE STRUCTURE

```
ai-ticket-processor/
├── ai_ticket_processor.py  # Main entry (orchestration)
├── zendesk_client.py      # Zendesk API wrapper
├── llm_client.py          # LLM API wrapper
└── processor.py           # Core processing logic
```

```
|— tag_manager.py      # Tag merge safety
|— logger.py          # Logging utilities
|— dashboard.py       # Streamlit dashboard
|— dashboard_utils.py # Dashboard data
|— config.py          # Configuration
```

#### Dependencies:

```
|— requests          # HTTP client
|— openai            # OpenAI SDK
|— streamlit         # Dashboard
|— pandas            # Data processing
|— plotly            # Charts
```

## 🔑 KEY DESIGN DECISIONS

### 1. Parallel Processing with ThreadPoolExecutor

#### Why not asyncio?

- Simpler to implement
- Good enough for I/O-bound tasks
- 10 workers = 9.7x speedup

#### Why not multiprocessing?

- Overhead too high for I/O
- Thread-safe requests library

### 2. JSON-Based Logging (No Database)

#### Why no PostgreSQL?

- Simpler deployment (no DB setup)
- JSON logs readable by dashboard
- Easy to back up and migrate

#### When to add DB?

- Multi-client SaaS deployment
- Need complex queries
- 10 concurrent clients

### 3. Fetch-Merge-Update Pattern

python

```
# WRONG (overwrites existing tags)
zendesk.update(ticket_id, tags=["ai_processed", "ai_bug"])

# RIGHT (preserves existing tags)
current = zendesk.get_ticket(ticket_id)
merged = current['tags'] + ["ai_processed", "ai_bug"]
zendesk.update(ticket_id, tags=merged)
```

**Why:** Preserves Zendesk's built-in ML tags

## 4. Structured LLM Prompts

```
Analyze this ticket and return ONLY valid JSON:
{
  "summary": "...",
  "root_cause": "bug|feature|refund|other",
  "urgency": "high|medium|low",
  "sentiment": "positive|neutral|negative"
}
```

**Why:**

- Enforces structure (no free text)
- Easy to validate
- Prevents hallucination

## 5. Circuit Breaker for API Calls

```
python

@zendesk_breaker
def update_ticket(ticket_id, **kwargs):
    # If 5 failures in a row, stop calling API
    # Wait 60 seconds, then try again
    pass
```

**Why:** Prevents cascading failures



## PERFORMANCE CHARACTERISTICS

### Current (10 workers)

| Metric        | Value           |
|---------------|-----------------|
| Throughput    | 167 tickets/min |
| Latency (avg) | 3.5 sec/ticket  |

| Metric        | Value          |
|---------------|----------------|
| Latency (p95) | 5.2 sec        |
| Success rate  | 99.8%          |
| Cost          | \$0.001/ticket |

## Bottlenecks

1. LLM API latency (2-3s) ← Dominates
2. Zendesk API latency (0.5-1s)
3. Network I/O

## Not bottlenecks:

- CPU (minimal processing)
- Memory (small payloads)
- Disk I/O (logs only)

## Scalability

### Vertical (more workers):

- 10 workers: 167/min (current)
- 20 workers: 320/min
- 50 workers: 700/min ← **Zendesk rate limit**

### Horizontal (multiple instances):

- Need Redis for ticket claiming
- Avoid duplicate processing
- Can scale to 10,000+ tickets/hour

## SECURITY

### Data Flow

#### Public LLM (default):

Zendesk → [Python] → OpenAI API → [Python] → Zendesk

└────────── TLS 1.3 ─────────┘

#### Private LLM (enterprise):

Zendesk → [Python] → [Llama 3.1 on GCP/On-prem] → [Python] → Zendesk

└────────────────── Never leaves VPC ─────────────────┘

Security Controls

| Control               | Implementation                 |
|-----------------------|--------------------------------|
| Encryption at Rest    | AES-256 (optional for logs)    |
| Encryption in Transit | TLS 1.3 (all API calls)        |
| API Keys              | Environment variables (.env)   |
| PII Redaction         | Regex patterns (SSN, CC, etc.) |
| Access Control        | IAM roles / service accounts   |
| Audit Trail           | 12-month log retention         |

Compliance

- **GDPR:** Private LLM keeps data in EU
- **HIPAA:** Private LLM + DLP + BAA
- **SOC 2:** Encryption + audit trail + IAM

 ERROR HANDLING

Retry Strategy

```
python

def call_with_retry(func, attempts=3):
    for i in range(attempts):
        try:
            return func()
        except RateLimitError:
            sleep(2 ** i) # 1s, 2s, 4s
        except NetworkError:
            sleep(1)
```

Fallback Strategy

If LLM fails after 3 retries:

1. Use rule-based analysis (keyword matching)
2. Tag with `ai_fallback`
3. Continue processing (don't halt)

# Circuit Breaker

If API fails 5 times in a row:

- 1. Stop calling API (fail-fast)
- 2. Wait 60 seconds
- 3. Try again (half-open)
- 4. If success, resume (closed)

## MONITORING

### Metrics (SLIs/SLOs)

| Metric        | SLI   | SLO   | Alert |
|---------------|-------|-------|-------|
| Success Rate  | 99.9% | 99.5% | <99%  |
| Latency (p95) | 10s   | 15s   | >15s  |
| Error Rate    | 0.1%  | 0.5%  | >0.5% |

### Health Check

```
python
def health_check():
    return {
        "status": "healthy|degraded|unhealthy",
        "checks": {
            "zendesk_api": "ok|error",
            "llm_api": "ok|error",
            "disk_space": "ok|warning|critical"
        },
        "metrics": {
            "success_rate_24h": 0.998,
            "avg_latency_24h": 3.5
        }
    }
```

### Alerts

- Slack webhook for warnings
- PagerDuty for critical
- Email for daily summaries

## DEPLOYMENT OPTIONS

## Option 1: Windows (Current)

Windows 10/11

- └─ Task Scheduler (every 10 min)
  - └─ python ai\_ticket\_processor.py
    - └─ Logs to logs/ directory

**Pros:** Zero cost, simple

**Cons:** Single machine, not scalable

## Option 2: GCP Cloud Run

Cloud Scheduler (every 10 min)

- └─ Triggers Cloud Run
  - └─ Docker container
    - └─ Runs processor
      - └─ Logs to Cloud Storage

**Cost:** ~\$8/month

**Pros:** Scalable, managed

**Cons:** Requires GCP knowledge

## Option 3: Kubernetes (Future)

CronJob (every 10 min)

- └─ Spawns Job pod
  - └─ Runs processor
    - └─ Logs to persistent volume

**Best for:** Multi-client SaaS



## TESTING

### Unit Tests

python

```
def test_llm_analysis():
    client = LLMClient("gpt-4o-mini", api_key)
    result = client.analyze("Bug", "Error 500")
    assert result['root_cause'] == 'bug'
    assert result['urgency'] in ['high', 'medium', 'low']
```

### Integration Tests

python

```
def test_end_to_end():
    processor = TicketProcessor(zendesk, llm, dry_run=True)
    result = processor.process_ticket(test_ticket)
    assert result['status'] == 'success'
```

## Load Tests

```
bash

# Simulate 1000 concurrent tickets
locust -f tests/load_test.py --users 100
```

**Target:** 99% success rate at 1000 tickets

## 🚀 FUTURE ARCHITECTURE

### Event-Driven (Real-time)

```
Zendesk Webhook
├── Message Queue (RabbitMQ)
│   └── Worker Pool
│       └── Process instantly (not every 10 min)
```

### Microservices

```
API Gateway
├── Ticket Service (fetch)
├── Analysis Service (LLM)
├── Tagging Service (update)
└── Notification Service (alerts)
```

### ML Pipeline

```
Tickets → Human QA → Fine-tune Llama → A/B Test → Deploy
└── Feedback Loop ───┘
```

## 💡 QUESTIONS FOR TEAM REVIEW

### Architecture

1. **Is ThreadPoolExecutor the right choice?** Or should we use asyncio?
2. **JSON logs vs PostgreSQL?** When to switch?
3. **Any concerns about rate limiting?**

## Security

4. Is DLP (PII redaction) sufficient? Or need more?
5. Should we add encryption at rest by default?
6. Any compliance requirements we missed?

## Performance

7. Is 167 tickets/min acceptable? Need faster?
8. Should we cache LLM responses? For duplicate tickets?
9. When to add horizontal scaling?

## Features

10. Which next feature is highest priority?
  - Webhooks (real-time)
  - Private LLM
  - Multi-platform (Freshdesk)
  - SaaS dashboard

## Deployment

11. Cloud (GCP) or on-prem?
12. Who manages infrastructure?

---

## NEXT STEPS

1. Read full spec: [AI\\_TICKET\\_PROCESSOR\\_TECHNICAL\\_SPEC.md](#) (35k words)
2. Review code: Clone repo and test locally
3. Provide feedback: GitHub issues or team meeting
4. Prioritize features: Vote on roadmap

---

## ADDITIONAL DOCS

- Full Technical Spec: [AI\\_TICKET\\_PROCESSOR\\_TECHNICAL\\_SPEC.md](#)
  - API Documentation: [docs/API.md](#)
  - Deployment Guide: [docs/DEPLOYMENT.md](#)
  - Troubleshooting: [docs/TROUBLESHOOTING.md](#)
-

**Contact:** Madhan Karthick ([madhan1787@gmail.com](mailto:madhan1787@gmail.com))

**Document End**