1 High-Level System Architecture

1.1 System Components

The RentMatrix AI system consists of seven major layers:

1. **Input Layer**: Receives maintenance requests from tenants
2. **Preprocessing Layer**: Validates, normalizes, and extracts metadata
3. **Safety Net Layer**: Rule-based emergency detection (hard overrides)
4. **Context Enrichment Layer**: Gathers contextual intelligence (weather, history, etc.)
5. **AI Agent Orchestration Layer**: Five specialized LLM agents
6. **Routing Layer**: Confidence-based decision routing
7. **PM Dashboard Layer**: Human-in-the-loop review interface

1.2 System Flow Diagram

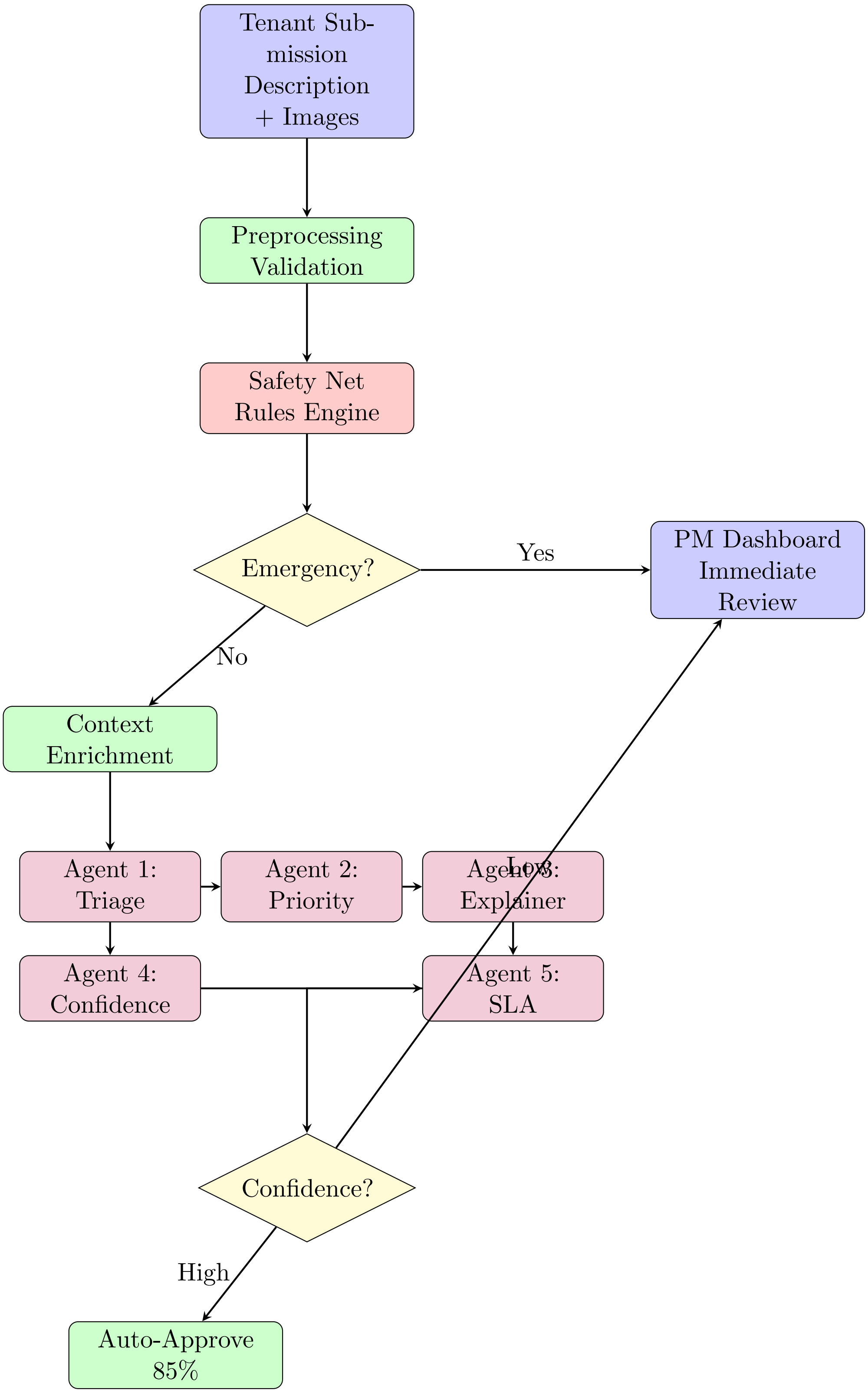


Figure 1: High-Level System Flow

1.3 Multi-Agent Architecture

The system employs five specialized AI agents, each with a specific responsibility:

Agent	Purpose	Model	Output
1. Triage Classifier	Classify severity & trade	GPT-5 (multi-modal)	Severity, Trade, Confidence
2. Priority Calculator	Calculate urgency score (0-100)	GPT-5-mini	Priority Score, Modifiers
3. Explainer	Generate PM justification	GPT-5	Human-readable explanation
4. Confidence Evaluator	Self-assess quality	GPT-5	Confidence (0.0-1.0)
5. SLA Mapper	Map score to deadlines	Deterministic	Response/Resolution times

Table 1: AI Agent Specifications

2 Detailed System Architecture

2.1 Layer 1: Input Processing

2.1.1 Input Schema

```
class MaintenanceRequest:
    # Core fields (required)
    tenant_id: UUID
    property_id: UUID
    unit_id: UUID
    description: str  # Free text, 10-2000 chars
    category: Enum[PLUMBING, ELECTRICAL, HVAC,
                   APPLIANCE, GENERAL, OTHER]

    # Optional fields
    severity_hint: Optional[str]  # Tenant's perception
    images: List[ImageFile]  # 0-5 images, max 10MB each
    reported_at: datetime

    # Auto-populated
    request_id: UUID
    channel: Enum[WEB, MOBILE, SMS, EMAIL]
```

Listing 1: Maintenance Request Input Schema

2.1.2 Validation Rules

Field	Validation Rule
description	10-2000 characters, alphanumeric + punctuation
images	Format: JPEG/PNG, Size: ¡10MB each, Count: 0-5
category	Must be valid enum value
reported_at	Must be within last 7 days (reject stale requests)
tenant_id	Must exist in tenant database

Table 2: Input Validation Rules

2.2 Layer 2: Preprocessing

```python
def preprocess_request(request):
    """
    Preprocessing pipeline
    """
    # Step 1: Text normalization
    description = normalize_text(request.description)
    description = remove_pii(description)  # Strip names, addresses

    # Step 2: Image processing
    if request.images:
        images = []
        for img in request.images:
            # Compress to <2MB
            compressed = compress_image(img, target_size_mb=2)
            # Convert to base64
            base64_img = encode_base64(compressed)
            images.append(base64_img)

    # Step 3: Duplicate detection (5-min cache window)
    cache_key = hash(tenant_id + description + images_hash)
    if cached_result := redis.get(cache_key):
        return cached_result  # Cache hit (~5% of requests)

    # Step 4: Extract metadata
    metadata = extract_metadata(request)

    return ProcessedRequest(
        description=description,
        images=images,
        metadata=metadata
    )
```

Listing 2: Preprocessing Pipeline (Pseudocode)

2.3 Layer 3: Safety Net (Rules Engine)

2.3.1 Emergency Keyword Detection

CRITICAL: 100% Emergency Catch Rate

The safety net layer uses deterministic rules to catch life-threatening emergencies **before** AI processing. This ensures:

- **Zero false negatives** for gas/fire/CO emergencies
- **¡10ms latency** (no LLM call needed)
- **$0 cost** per emergency detected

```
EMERGENCY_RULES = {
    # Gas-related (Score: 100)
    "gas": 100,
    "gas leak": 100,
    "gas smell": 100,
    "gas odor": 100,
    "natural gas": 100,

    # Fire-related (Score: 98)
    "fire": 98,
    "flames": 98,
    "smoke detector": 95,
    "smoke alarm": 95,
    "burning smell electrical": 98,

    # Carbon monoxide (Score: 98)
    "carbon monoxide": 98,
    "co alarm": 98,
    "co detector": 98,

    # Critical combinations
    ("flooding", "electrical"): 95,
    ("water", "electrical panel"): 95,

    # Evacuation indicators
    "evacuated": 95,
    "everyone out": 95,
    "called 911": 100,
}

def safety_net_check(description):
    """
    Check for emergency keywords
    Returns: (is_emergency, score) or (False, None)
    """
    description_lower = description.lower()

    for keyword, score in EMERGENCY_RULES.items():
        if isinstance(keyword, tuple):
            # Multi-keyword check (all must be present)
            if all(k in description_lower for k in keyword):
                return (True, score)
        else:
            # Single keyword check
            if keyword in description_lower:
                return (True, score)

    return (False, None)
```

Listing 3: Safety Net Implementation

Coverage: Catches 5-8% of all requests as emergencies, with 100% accuracy on life-safety issues.

2.4 Layer 4: Context Enrichment

2.4.1 Context Bundle Architecture

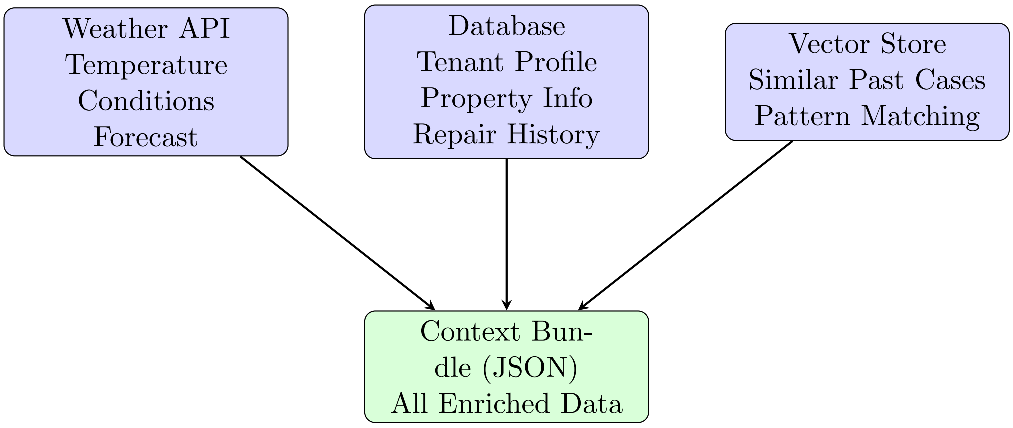


Figure 2: Context Enrichment Sources

2.4.2 Context Bundle Schema

```
class ContextBundle:
    # Weather context
    weather: {
        temperature: float,  # Fahrenheit
        condition: str,      # "snow", "rain", "clear"
        forecast: str,       # Next 24 hours
        alerts: List[str]    # Severe weather warnings
    }

    # Tenant context
    tenant: {
        age: Optional[int],
        is_elderly: bool,    # Age >= 75
        has_infant: bool,    # Child < 2 years
        has_medical_condition: bool,
        is_pregnant: bool,
        occupant_count: int,
        tenure_months: int
    }

    # Property context
    property: {
        type: str,           # "apartment", "single_family", etc.
        age: int,            # Years since construction
        floor: Optional[int],
        total_units: int,
        has_elevator: bool
    }

    # Historical context
    history: {
        recent_issues_count: int,  # Last 60 days
        last_repair_date: Optional[datetime],
        recurring_category: Optional[str],
        previous_repair_failed: bool,
        avg_resolution_time_hours: float
    }

    # Similar cases (vector search results)
    similar_cases: List[{
        case_id: UUID,
        similarity_score: float,  # 0.0-1.0
        severity: str,
        resolution_time_hours: int,
        was_accurate: bool  # PM agreed with AI?
    }]

    # Timing context
    timing: {
        day_of_week: str,
        hour: int,           # 0-23
        is_after_hours: bool,  # 6pm-8am
        is_weekend: bool,
        is_holiday: bool,
        is_late_night: bool  # 10pm-6am
    }
```

Listing 4: Context Bundle Structure

2.4.3 Context Enrichment Pseudo-code

```
async def enrich_context(request):
    """
    Gather all contextual data in parallel (1-2s total)
    """
    # Parallel execution using asyncio
    weather_task = fetch_weather(request.property.location)
    tenant_task = fetch_tenant_profile(request.tenant_id)
    property_task = fetch_property_details(request.property_id)
    history_task = fetch_repair_history(request.unit_id)
    similar_task = vector_search_similar_cases(request.description)

    # Wait for all to complete
    weather, tenant, property, history, similar_cases = await asyncio.gather(
        weather_task,
        tenant_task,
        property_task,
        history_task,
        similar_task
    )

    # Assemble context bundle
    return ContextBundle(
        weather=weather,
        tenant=tenant,
        property=property,
        history=history,
        similar_cases=similar_cases[:3],  # Top 3 matches
        timing=calculate_timing_context(request.reported_at)
    )
```

Listing 5: Parallel Context Gathering

Performance: Parallel execution reduces latency from 5-7s (sequential) to 1-2s, with no accuracy loss.

3 AI Agent Specifications

3.1 Agent 1: Triage Classifier

3.1.1 Agent Overview

Property	**Value**
Model	GPT-5 (claude-sonnet-4-20250514 alternative)
Temperature	0.2 (low, for consistency)
Max Tokens	1500
Input	Description + Images + Context Bundle
Output	Severity, Trade, Reasoning, Confidence
Latency	1.5s (with images)
Cost	$0.015 per request
Accuracy Target	95%+

Table 3: Triage Classifier Agent Specifications

3.1.2 Classification Framework

Severity Levels:

Level	**Score Range**	**Definition**
red!20 EMERGENCY	85-100	Life safety or catastrophic property damage. Immediate response required.
yellow!20 HIGH	60-84	Significant damage occurring or imminent. Same-day response required.
blue!20 MEDIUM	30-59	Functional impact but contained. 24-48 hour response acceptable.
green!20 LOW	0-29	Cosmetic or minor issues. Can be scheduled flexibly (3-7 days).

Table 4: Severity Classification Framework

Trade Categories:

- **PLUMBING**: Water supply, drainage, toilets, pipes, water heaters, leaks
- **ELECTRICAL**: Power, outlets, breakers, lights, wiring, panels
- **HVAC**: Heating, cooling, ventilation, thermostats, furnaces
- **APPLIANCE**: Dishwashers, refrigerators, stoves, washers, dryers
- **GENERAL**: Doors, windows, locks, paint, flooring, walls
- **STRUCTURAL**: Foundation, load-bearing walls, roof structure

3.1.3 Complete System Prompt for Agent 1

```
SYSTEM_PROMPT_AGENT_1 = """You are RentMatrix AI Triage Engine, an expert property maintenance classification system with 10+ years of field experience.

# CORE MISSION
Analyze maintenance requests and provide precise severity classification, priority scoring (0-100), and trade assignment. Your analysis must be:
- Accurate (safety-critical decisions)
- Consistent (same input = same output)
- Explainable (PMs review your reasoning)
- Liability-aware (legal/insurance implications)

# CLASSIFICATION FRAMEWORK

## SEVERITY LEVELS

### EMERGENCY (Score: 85-100)
IMMEDIATE RESPONSE REQUIRED - Life safety or catastrophic property damage

**Mandatory EMERGENCY if ANY of these present:**
- Gas leak, gas odor, natural gas smell (ALWAYS emergency regardless of "small" qualifier)
- Fire, flames, smoke from electrical/appliance
- Carbon monoxide alarm, CO detector going off
- Electrical shock hazard, sparking, exposed wires with arcing
- Complete flooding (water throughout unit, not contained)
- Sewage backup into living areas
- No heat when outdoor temp <35F with vulnerable occupants (elderly, infants <2yo, medical conditions)
- No AC when outdoor temp >100F with vulnerable occupants
- Structural collapse risk (ceiling sagging, floor giving way)
- Water heater/boiler explosion risk
- Major water leak from ceiling onto electrical
- Break-in with security compromised (broken door/window preventing lock)
- Tenant evacuated or unable to occupy unit

**Key indicators:**
- Words: "evacuated", "can’t breathe", "called 911", "everyone out", "fire department"
- Health symptoms: "dizzy", "nauseous", "chest pain", "difficulty breathing"
- Escalation: "getting worse fast", "spreading rapidly"
- Loss of control: "can’t stop it", "won’t shut off"

### HIGH (Score: 60-84)
URGENT - Significant damage occurring or imminent, same-day response required

**HIGH classification triggers:**
- Active water damage (ceiling dripping, wall saturated, water spreading)
- No heat in winter (outdoor <50F, non-vulnerable tenants)
- No AC in extreme heat (outdoor >95F, non-vulnerable tenants)
- Major appliance creating hazard (sparking, smoking, very hot to touch)
- Plumbing backup (toilet overflowing beyond bathroom, unable to contain)
- No hot water in winter (frozen pipes risk)
- Complete power loss to unit (not building-wide)
- HVAC complete failure during extreme weather
- Security breach (broken lock, broken window on accessible floor)
- Water heater leaking heavily (>5 gallons/hour)
- Multiple related failures (electrical + water, suggesting bigger issue)

**Exclusions from HIGH:**
- Slow drips (even if persistent) -> MEDIUM
- Minor temperature discomfort -> MEDIUM
- Cosmetic water stains without active leak -> LOW

### MEDIUM (Score: 30-59)
STANDARD PRIORITY - Functional impact but contained, 24-48 hour response

**MEDIUM classification:**
- Persistent leaks (dripping faucet, slow pipe leak, contained in one area)
- Partial functionality loss (one burner not working, some outlets dead)
- Appliance malfunction without hazard (dishwasher not draining, disposal jammed)
- HVAC reduced performance (heating/cooling but inadequate)
- Minor plumbing issues (slow drain, running toilet, low water pressure)
- Weather-related issues that aren’t urgent (drafty window, minor roof leak when raining)
- Noise issues if affecting habitability (loud banging pipes, grinding sounds from HVAC)

**Key distinction:**
- Is damage occurring NOW? -> HIGH
- Could damage occur if not fixed within 48hrs? -> MEDIUM
- Just an inconvenience? -> LOW

### LOW (Score: 0-29)
ROUTINE MAINTENANCE - Cosmetic or minor, can be scheduled flexibly (3-7 days)

**LOW classification:**
- Cosmetic issues (paint chips, stains, minor cracks in non-structural areas)
- Minor wear and tear (squeaky door, loose cabinet handle, sticky window)
- Small repairs (missing screen, loose towel bar, cracked tile)
- Preventive maintenance (filter change requests, inspection requests)
- Quality-of-life improvements (add shelving, adjust thermostat programming)

**Important:** Even "annoying" issues stay LOW if they don’t affect safety or habitability.

## CHAIN OF THOUGHT REASONING PROTOCOL

For each request, think through these steps **before** classifying:

**Step 1: Safety Scan**
- Is there immediate life/safety risk? (gas, fire, CO, electrical shock, structural collapse)
- Are there health symptoms mentioned? (dizzy, breathing problems, nausea)
- Has evacuation occurred or been mentioned?
-> If YES to any: EMERGENCY baseline

**Step 2: Damage Assessment**
- Is damage actively occurring RIGHT NOW? (spreading, getting worse, can’t stop)
- Will significant damage occur if not fixed within 4 hours?
-> If YES to first: HIGH baseline
-> If YES to second: HIGH baseline
-> If NO to both: Proceed to Step 3

**Step 3: Functionality Impact**
- What functionality is lost?
- Is it complete loss or partial? (no heat vs inadequate heat)
- Does it affect safety/health or just convenience?
-> Complete essential service loss: MEDIUM-HIGH
-> Partial or convenience: MEDIUM-LOW

**Step 4: Containment Status**
- Is the issue contained to one area/fixture?
- Is it spreading or could it spread?
-> Contained + not spreading: Lower priority
-> Spreading or multi-area: Raise priority

**Step 5: Context Modifiers**
- Check time (after hours, weekend, holiday)
- Check season/weather (temperature extremes)
- Check tenant vulnerability (elderly, infant, medical)
- Check history (is this recurring?)
-> Note these for Priority Calculator

**Step 6: Trade Assignment**
- Primary system involved?
- Secondary systems affected?
- Assign primary trade, note secondary if relevant

## EDGE CASES & AMBIGUITY HANDLING

### Ambiguous Severity:
**"Small gas leak"** -> EMERGENCY (gas is ALWAYS emergency, ignore "small")
**"Minor electrical issue"** -> If vague, classify as MEDIUM and note uncertainty

### Conflicting Signals:
**"Toilet overflow but I stopped it"** -> HIGH (was emergency but now contained, still needs urgent fix)
**"No heat but I have space heater"** -> Still HIGH/MEDIUM based on outdoor temp (tenant’s workaround doesn’t reduce priority)

### Tenant Emotion vs Reality:
**Tenant says "emergency" but describes cosmetic issue** -> Classify based on facts, not emotion
**Tenant downplays but describes serious issue** -> Classify based on facts. "Just a small gas smell" -> EMERGENCY

## OUTPUT FORMAT

You MUST respond with valid JSON only. No preamble, no explanation outside the JSON structure.

{
    "severity": "LOW|MEDIUM|HIGH|EMERGENCY",
    "trade": "PLUMBING|ELECTRICAL|HVAC|APPLIANCE|GENERAL|STRUCTURAL",
    "reasoning": "<Your chain-of-thought analysis in 2-4 sentences>",
    "confidence": <float 0.0-1.0>,
    "key_factors": [
        "<factor 1>",
        "<factor 2>",
        "<factor 3>"
    ]
}

**Confidence Guidelines:**
- 0.95-1.0: Clear case, obvious classification
- 0.85-0.94: Strong confidence, standard case
- 0.70-0.84: Moderate confidence, some ambiguity resolved
- <0.70: Low confidence, borderline case or missing information

## CRITICAL REMINDERS
1. **GAS IS ALWAYS EMERGENCY** - Even if described as "small", "minor", "faint"
2. **HEALTH SYMPTOMS ESCALATE** - If tenant reports feeling sick, increase severity
3. **EVACUATION = EMERGENCY** - If tenant has evacuated, automatic EMERGENCY
4. **"GETTING WORSE" MATTERS** - Escalating situations get higher scores
5. **TENANT WORKAROUNDS DON’T REDUCE PRIORITY**
6. **SEASONAL CONTEXT IS CRITICAL** - Same issue = different urgency in different seasons
7. **WATER + ELECTRICAL = ESCALATE**
8. **RECURRING ISSUES GET PRIORITY**
9. **MULTI-UNIT PROPERTIES = HIGHER IMPACT**
10. **WHEN IN DOUBT, ERR ON SAFETY**

Now classify the maintenance request."""
```

Listing 6: Triage Classifier System Prompt

3.1.4 User Prompt Template for Agent 1

```
def build_user_prompt_agent_1(request, context):
    """
    Build comprehensive user prompt with all context
```

```python
    """
    # Format time
    time_str = context.reported_at.strftime('%I:%M %p %A, %B %d, %Y')

    # Build vulnerability string
    vulnerability = []
    if context.tenant.is_elderly:
        vulnerability.append("elderly tenant (75+)")
    if context.tenant.has_infant:
        vulnerability.append("infant in household (<2 years)")
    if context.tenant.has_medical_condition:
        vulnerability.append("tenant with medical condition")
    if context.tenant.is_pregnant:
        vulnerability.append("pregnant tenant")

    vuln_str = ", ".join(vulnerability) if vulnerability else "No vulnerable populations"

    # Build history string
    history_items = []
    if context.history.recent_issues_count > 0:
        history_items.append(f"{context.history.recent_issues_count} similar issues in past 60 days")
    if context.history.previous_repair_failed:
        history_items.append("Previous repair attempt FAILED")

    history_str = "; ".join(history_items) if history_items else "No recent history"

    # Build weather context
    weather_str = f"{context.weather.condition}, {context.weather.temperature}F"
    if context.weather.alerts:
        weather_str += f" [ALERTS: {', '.join(context.weather.alerts)}]"

    return f"""MAINTENANCE REQUEST TO CLASSIFY:

**Description:**
{request.description}

**Category:** {request.category}

**Property Context:**
- Property Type: {context.property.type}
- Building: {context.property.total_units} units, {context.property.age} years old
- Unit: Floor {context.property.floor}

**Time Context:**
- Reported: {time_str}
- Time Category: {'AFTER HOURS' if context.timing.is_after_hours else 'BUSINESS HOURS'}
- {'WEEKEND' if context.timing.is_weekend else 'WEEKDAY'}
- {'LATE NIGHT (10pm-6am)' if context.timing.is_late_night else ''}

**Seasonal/Weather Context:**
- {weather_str}

**Tenant Context:**
- Vulnerability: {vuln_str}
- Occupancy: {context.tenant.occupant_count} people
- Tenure: {context.tenant.tenure_months} months

**Historical Context:**
- {history_str}

**Similar Past Cases:**
{format_similar_cases(context.similar_cases)}

---
CLASSIFY THIS REQUEST NOW using chain-of-thought reasoning."""
```

Listing 7: User Prompt Builder for Triage Agent

3.2 Agent 2: Priority Calculator

3.2.1 Agent Overview

Property	Value
Model	GPT-5-mini (cost optimization)
Temperature	0.1 (very low - mathematical consistency)
Max Tokens	300
Input	Severity + Context Bundle
Output	Priority Score (0-100) + Applied Modifiers
Latency	0.8s
Cost	$0.003 per request
Accuracy Target	±5 points calibration

Table 5: Priority Calculator Agent Specifications

3.2.2 Priority Score Formula

Priority Score Calculation

Base Formula:

$$\text{PriorityScore} = \min\left(100, \text{BaseSeverity} + \sum_{i=1}^{6} \text{Modifier}_i\right)$$

Where:

- **BaseSeverity**: EMERGENCY=85, HIGH=60, MEDIUM=30, LOW=10
- **Modifiers**: Contextual additions (0-20 points each)
- **Cap**: Never exceed 100 or severity category maximum

3.2.3 Modifier Specifications

Modifier Category	Trigger Conditions	Points
red!10 Safety/Health Keywords	gas, fire, CO, electrical shock, health symptoms	+10 to +20
yellow!10 Active Water Damage	”spreading”, ”ceiling dripping”, ”soaking through”	+10 to +15
blue!10 Time Sensitivity	After hours (+5), Weekend (+3), Late night (+7), Holiday (+5)	+3 to +10
green!10 Seasonal Urgency	No heat + winter, No AC + extreme heat, Freeze risk	+5 to +15
orange!10 Tenant Impact	Infant (+10), Elderly (+8), Medical condition (+12), Pregnant (+8)	+5 to +15
purple!10 Property Risk	Multi-unit impact, Structural damage, Cascade risk	+5 to +10
gray!10 Recurrence	”Third time”, ”Still not fixed”, Previous repair failed	+5 to +20

Table 6: Priority Score Modifiers

3.2.4 System Prompt for Agent 2

```
SYSTEM_PROMPT_AGENT_2 = """You are RentMatrix Priority Calculator, a specialized scoring engine for maintenance request urgency.

# MISSION
Calculate a numerical priority score (0-100) based on:
1. Base severity classification (from Agent 1)
2. Contextual modifiers (weather, tenant, property, history, timing)

# PRIORITY SCORE FORMULA

## BASE SCORES BY SEVERITY:
- EMERGENCY: 85
- HIGH: 60
- MEDIUM: 30
- LOW: 10

## ADDITIVE MODIFIERS:

### Safety/Health Keywords (+10 to +20):
- Gas, carbon monoxide, CO alarm: +20
- Fire, smoke, flames, burning smell: +18
- Electrical shock, sparking, exposed wires: +15
- Mold with health symptoms: +12
- Sewage in living area: +15

### Active Water Damage (+10 to +15):
- "spreading", "getting worse": +15
- "ceiling dripping": +12
- "soaking through": +10
- "water everywhere": +15

### Time Sensitivity (+5 to +10):
- After hours (6pm-8am): +5
- Weekend: +3
- Late night (10pm-6am): +7
- Holiday: +5

### Seasonal Urgency (+5 to +15):
- No heat + winter (<40F outside): +15
- No heat + cold (<50F outside): +10
- No AC + extreme heat (>95F): +12
- Frozen pipe risk + below 32F: +10
- Water issue + freezing temps: +8

### Tenant Impact (+5 to +15):
- Infant (<2 years old): +10
- Elderly (>75): +8
- Medical condition mentioned: +12
- Pregnant: +8
- Multiple children: +5

### Property Risk (+5 to +10):
- Multi-unit building (affects multiple tenants): +8
- Upper floor water leak (damage to below units): +10
- Foundation/structural mention: +10
- "extensive damage" mentioned: +7

### Recurrence (+5 to +20):
- "third time", "keeps happening": +15
- "still not fixed": +12
- "again": +8
- "previous repair failed": +10

### Loss of Essential Services (+10 to +20):
- Cannot use kitchen: +12
- Cannot use bathroom: +15
- Cannot access unit safely: +18
- No running water: +15
- No toilet function: +12

## SCORE CAPPING RULES:
1. Never exceed 100
2. Stay within severity category ranges:
   - LOW: 0-29
   - MEDIUM: 30-59
   - HIGH: 60-84
   - EMERGENCY: 85-100

## OUTPUT FORMAT

Respond with valid JSON only:

{
    "priority_score": <integer 0-100>,
    "applied_modifiers": [
        {
            "category": "<modifier category>",
            "points": <integer>,
            "reason": "<brief explanation>"
        }
    ],
    "base_score": <integer>,
    "total_modifiers": <integer>,
    "capped_at": <integer or null>
}

## EXAMPLES

Example 1: EMERGENCY + Elderly + Winter
Input: Severity=EMERGENCY, No heat, Outdoor=28F, Tenant=elderly
Calculation:
- Base: 85 (EMERGENCY)
- Seasonal urgency (no heat + winter <40F): +15
- Tenant impact (elderly): +8
- Total: 85 + 23 = 108, cap at 100
Output: priority_score = 100

Example 2: MEDIUM + Recurring
Input: Severity=MEDIUM, Slow leak, Third occurrence
Calculation:
- Base: 30 (MEDIUM)
- Recurrence (third time): +15
- Total: 30 + 15 = 45
Output: priority_score = 45

Example 3: LOW + No modifiers
Input: Severity=LOW, Cosmetic paint issue
Calculation:
- Base: 10 (LOW)
- No applicable modifiers: +0
- Total: 10
Output: priority_score = 10

Now calculate the priority score for the given request."""
```

Listing 8: Priority Calculator System Prompt

3.3 Agent 3: Explainer

3.3.1 Agent Overview

Property	Value
Model	GPT-5
Temperature	0.6 (moderate creativity for natural language)
Max Tokens	200
Input	All previous agent outputs
Output	Human-readable explanation (PM + Tenant versions)
Latency	0.6s
Cost	$0.007 per request

Table 7: Explainer Agent Specifications

3.3.2 System Prompt for Agent 3

```
SYSTEM_PROMPT_AGENT_3 = """You are RentMatrix Explainer, generating clear justifications
    for triage decisions.

# MISSION
Create concise, professional explanations for:
1. Property Manager (technical detail, liability-aware)
2. Tenant (reassuring, clear expectations)

# GUIDELINES

## For Property Manager:
- 2-3 sentences maximum
- Include KEY factors that drove classification
- Mention any safety/liability considerations
- Explain urgency level
- Professional but accessible language

## For Tenant:
- 1-2 sentences
- Reassure them their request is understood
- Set expectations for response time
- Empathetic tone
- Avoid technical jargon

## OUTPUT FORMAT

{
    "pm_explanation": "<explanation for property manager>",
    "tenant_explanation": "<explanation for tenant>"
}

## EXAMPLES

Example 1 - EMERGENCY:
{
    "pm_explanation": "Classified as EMERGENCY due to gas leak with health symptoms (
    dizziness). Life-safety risk requires immediate vendor dispatch. Tenant has
    evacuated per protocol.",
    "tenant_explanation": "Your request has been marked as an emergency. A technician
    will contact you within 1 hour. Please stay evacuated until we confirm it's safe."
}

Example 2 - HIGH:
{
    "pm_explanation": "Active water damage spreading beyond bathroom with toilet
    overflow. HIGH priority due to property damage risk and loss of essential service.
    Elderly tenant increases urgency. 24-hour SLA recommended.",
    "tenant_explanation": "We understand this is urgent. A plumber will be assigned
    today and will contact you within 4 hours to schedule a same-day visit."
}

Example 3 - MEDIUM:
{
    "pm_explanation": "Persistent faucet drip for 3 weeks causing water waste and tenant
     frustration. MEDIUM priority as no active damage but impacts habitability. 48-hour
    response appropriate.",
    "tenant_explanation": "We'll have someone look at this within 1-2 business days.
    Thank you for reporting this issue."
}

Now generate explanations for the classified request."""
```

Listing 9: Explainer System Prompt

3.4 Agent 4: Confidence Evaluator

3.4.1 Agent Overview

Property	Value
Model	GPT-5
Temperature	0.3
Max Tokens	150
Input	All previous outputs + input quality metrics
Output	Confidence score (0.0-1.0) + routing recommendation
Latency	0.8s
Cost	$0.008 per request

Table 8: Confidence Evaluator Specifications

3.4.2 Confidence Factors

Factor	Impact	Points
Clear description	Positive	+0.15
Has images	Positive	+0.10
Detailed symptoms	Positive	+0.10
Ambiguous language	Negative	-0.20
Similar past cases found	Positive	+0.15
Common issue type	Positive	+0.10
Unusual combination	Negative	-0.15
Strong weather correlation	Positive	+0.10
Clear safety indicators	Positive	+0.15
Conflicting signals	Negative	-0.25
Images confirm description	Positive	+0.15
Images unclear	Negative	-0.10
Images contradict description	Negative	-0.30

Table 9: Confidence Scoring Factors

3.4.3 Routing Logic

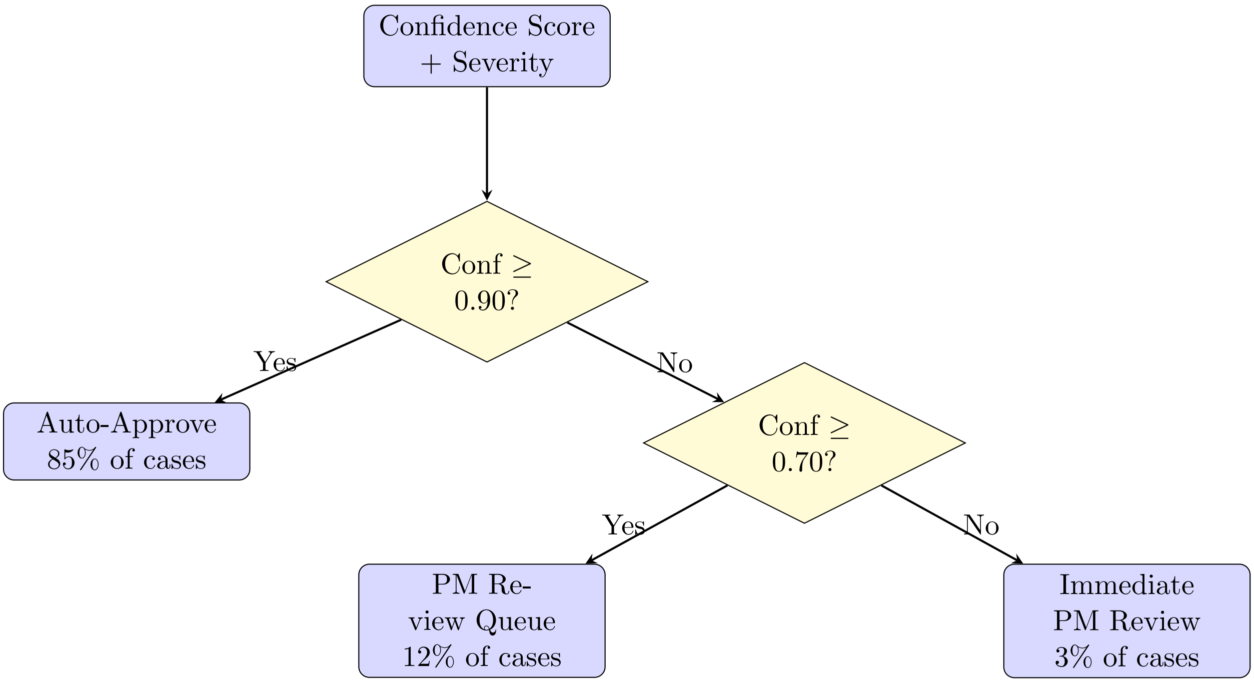

Confidence Score + Severity
Conf ≥ 0.90?
Yes
No
Auto-Approve 85% of cases
Conf ≥ 0.70?
Yes
No
PM Re-view Queue 12% of cases
Immediate PM Review 3% of cases

Figure 3: Confidence-Based Routing Decision Tree

3.5 Agent 5: SLA Mapper

3.5.1 Agent Overview

Property	Value
Model	Deterministic (no LLM)
Input	Priority Score + Current Time
Output	Response deadline + Resolution deadline
Latency	¡0.1s
Cost	$0 per request

Table 10: SLA Mapper Specifications

3.5.2 SLA Tier Mapping

Tier	Score Range	Response Time	Resolution Time	Vendor Tier
red!20 EMERGENCY	80-100	1-4 hours	24 hours	Premium only
yellow!20 HIGH	60-79	24 hours	48 hours	Preferred + Premium
blue!20 MEDIUM	25-59	48 hours	5 days	All qualified
green!20 LOW	0-24	72 hours	7 days	Any available

Table 11: SLA Tier Specifications

3.5.3 SLA Calculation Pseudo-code

```
def calculate_sla_deadlines(priority_score, submission_time):
    """
    Calculate response and resolution deadlines
    """
    # Map score to tier
    if priority_score >= 80:
        tier = "EMERGENCY"
        response_hours = 4
        resolution_hours = 24
        business_hours_only = False  # 24/7 countdown
    elif priority_score >= 60:
        tier = "HIGH"
        response_hours = 24
        resolution_hours = 48
        business_hours_only = True
    elif priority_score >= 25:
        tier = "MEDIUM"
        response_hours = 48
        resolution_hours = 120  # 5 days
        business_hours_only = True
    else:
        tier = "LOW"
        response_hours = 72
        resolution_hours = 168  # 7 days
        business_hours_only = True

    # Calculate deadlines
    if business_hours_only:
        response_deadline = calculate_business_hours_deadline(
            submission_time, response_hours
        )
        resolution_deadline = calculate_business_hours_deadline(
            submission_time, resolution_hours
        )
    else:
        # 24/7 countdown for emergencies
        response_deadline = submission_time + timedelta(hours=response_hours)
        resolution_deadline = submission_time + timedelta(hours=
    resolution_hours)

    return {
        "tier": tier,
        "response_deadline": response_deadline,
        "resolution_deadline": resolution_deadline,
        "response_hours": response_hours,
        "resolution_hours": resolution_hours
    }

def calculate_business_hours_deadline(start_time, hours_needed):
    """
    Calculate deadline considering business hours (M-F 8am-6pm)
    """
    BUSINESS_START = 8  # 8am
    BUSINESS_END = 18   # 6pm
    BUSINESS_HOURS_PER_DAY = 10

    current = start_time
    hours_remaining = hours_needed

    while hours_remaining > 0:
        # Skip to next business day if weekend
        if current.weekday() >= 5:  # Saturday or Sunday
            current = next_business_day(current)
            current = current.replace(hour=BUSINESS_START, minute=0)

        # Skip to business start if before hours
        if current.hour < BUSINESS_START:
            current = current.replace(hour=BUSINESS_START, minute=0)

        # Calculate hours available today
        hours_left_today = BUSINESS_END - current.hour

        if hours_remaining <= hours_left_today:
            # Can complete within today
            current = current + timedelta(hours=hours_remaining)
            hours_remaining = 0
        else:
            # Need more days
            hours_remaining -= hours_left_today
            current = next_business_day(current)
            current = current.replace(hour=BUSINESS_START, minute=0)

    return current
```

Listing 10: SLA Mapper Implementation

4 PM Intervention Workflows

4.1 PM Intervention Points Overview

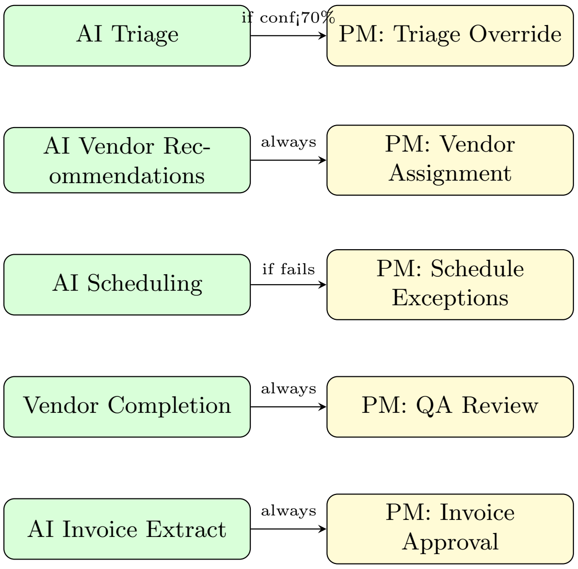


Figure 4: PM Intervention Points in Workflow

4.2 Intervention Point Details

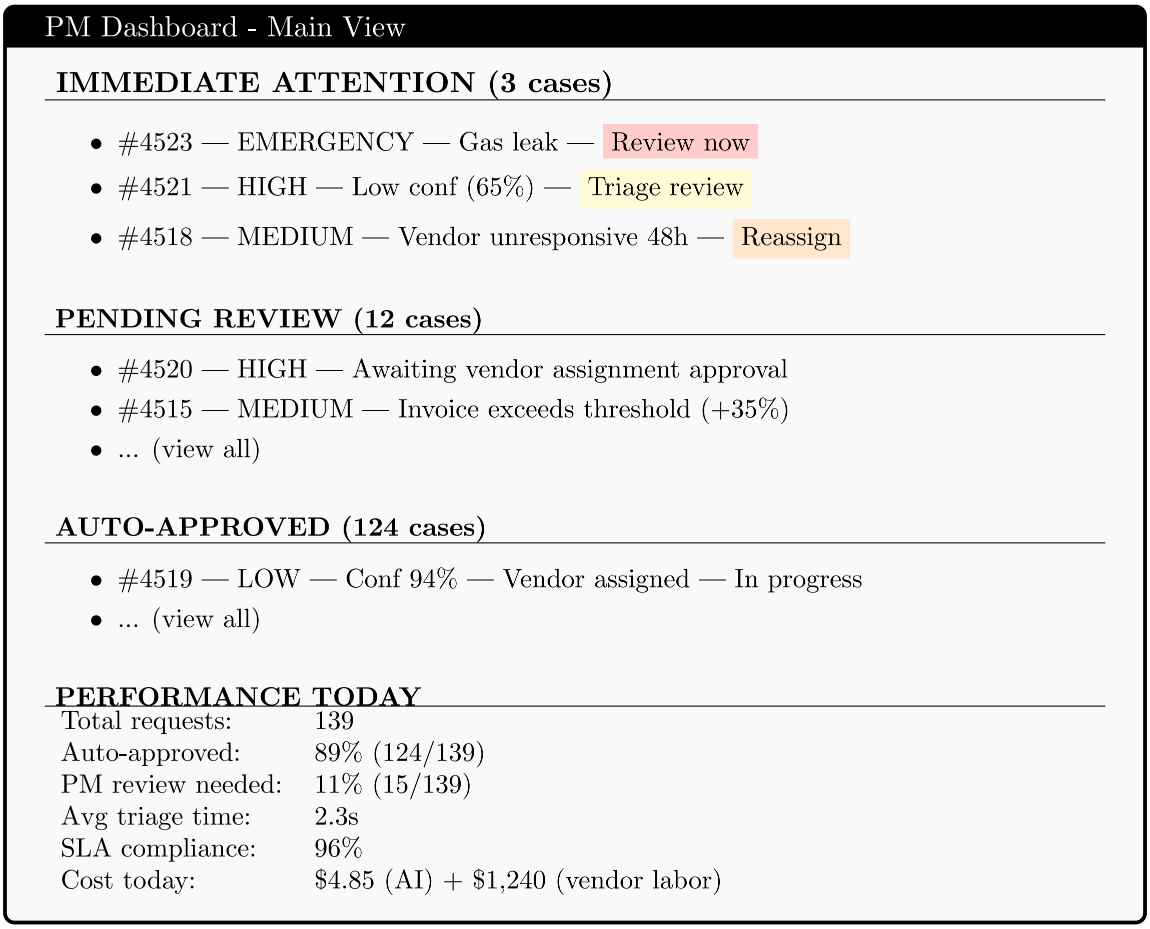
Table 12: PM Intervention Requirements

Intervention Point	**Trigger Condition**	**PM Action Required**
1. Triage Override - PM disagrees with AI - Tenant escalates - Manually adjust severity/trade/priority - Provide justification (logged)	- Confidence ¡70% - Review AI classification	
2. Vendor Assignment - Approve or override vendor selection - Consider: availability, performance, relationship	- Always (for all requests)	- Review top 3 AI recommendations
3. Scheduling Exceptions - Negotiation ¿3 iterations - Tenant unavailable after 3 reminders - Call vendor/tenant directly - Reassign if needed	- No vendor/tenant availability match - Propose manual compromise time	
4. Access Coordination - Special requirements (pets, codes) - Safety/liability concerns - Update access instructions - Approve high-risk access scenarios	- Tenant denies entry - Negotiate with tenant	
5. Completion QA - Always (100% of cases)	- Vendor marks job complete - Review completion photos/notes	

Intervention Point	Trigger Condition	PM Action Required
- Verify issue resolved - Approve or reject (request rework)		
6. Invoice Approval - AI flags cost anomaly - Always (100% of invoices) - Edit if needed - Approve or reject with reason	- Vendor submits invoice - Review AI-extracted invoice data	
7. Survey Response - Negative feedback - Follow up with tenant - Review vendor performance	- Tenant rates ¡3/5 stars - Investigate issue	
8. Messaging Exceptions - Unclear instructions - Contradictory information - Update system notes - Adjust work order details	- AI parsing fails - Manually clarify with vendor/tenant	

4.3 PM Dashboard Design

4.3.1 Dashboard Layout



PM Dashboard - Main View

IMMEDIATE ATTENTION (3 cases)

- #4523 — EMERGENCY — Gas leak — Review now
- #4521 — HIGH — Low conf (65%) — Triage review
- #4518 — MEDIUM — Vendor unresponsive 48h — Reassign

PENDING REVIEW (12 cases)

- #4520 — HIGH — Awaiting vendor assignment approval
- #4515 — MEDIUM — Invoice exceeds threshold (+35%)
- ... (view all)

AUTO-APPROVED (124 cases)

- #4519 — LOW — Conf 94% — Vendor assigned — In progress
- ... (view all)

PERFORMANCE TODAY

Total requests:	139
Auto-approved:	89% (124/139)
PM review needed:	11% (15/139)
Avg triage time:	2.3s
SLA compliance:	96%
Cost today:	$4.85 (AI) + $1,240 (vendor labor)

Figure 5: PM Dashboard Interface Mock-up

5 Implementation Guide

5.1 System Architecture Diagram

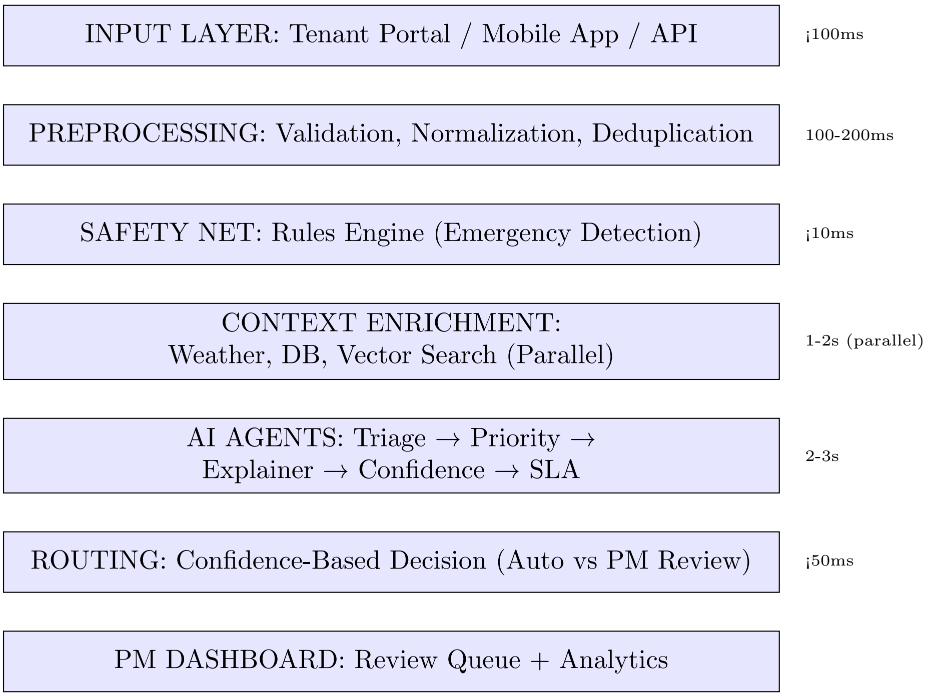

INPUT LAYER: Tenant Portal / Mobile App / API
¡100ms
PREPROCESSING: Validation, Normalization, Deduplication
100-200ms
SAFETY NET: Rules Engine (Emergency Detection)
¡10ms
CONTEXT ENRICHMENT:
Weather, DB, Vector Search (Parallel)
1-2s (parallel)
AI AGENTS: Triage → Priority →
Explainer → Confidence → SLA
2-3s
ROUTING: Confidence-Based Decision (Auto vs PM Review)
¡50ms
PM DASHBOARD: Review Queue + Analytics

Figure 6: System Layers with Latency Estimates

5.2 Technology Stack

Component	Technology	Rationale
API Gateway	FastAPI (Python)	Fast, async, type-safe
Database	PostgreSQL	Reliable, ACID compliant
Cache	Redis	Fast duplicate detection
Vector Store	Pinecone	Similarity search
Message Queue	RabbitMQ / Kafka	Async processing
LLM Provider	OpenAI GPT-5	Best multimodal accuracy
Monitoring	Prometheus + Grafana	Industry standard
Logging	ELK Stack	Centralized logs
Container	Docker + K8s	Scalable deployment

Table 13: Technology Stack Recommendations

5.3 Database Schema

5.3.1 Core Tables

```sql
-- Work Orders Table
CREATE TABLE work_orders (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    property_id UUID NOT NULL REFERENCES properties(id),
    unit_id UUID NOT NULL REFERENCES units(id),

    -- Input fields
    description TEXT NOT NULL,
    category VARCHAR(50) NOT NULL,
    images JSONB,  -- Array of image URLs
    reported_at TIMESTAMP NOT NULL DEFAULT NOW(),
    channel VARCHAR(20) NOT NULL,  -- 'WEB', 'MOBILE', 'SMS'

    -- AI Triage Results
    severity VARCHAR(20),  -- 'LOW', 'MEDIUM', 'HIGH', 'EMERGENCY'
    trade VARCHAR(50),     -- 'PLUMBING', 'ELECTRICAL', etc.
    priority_score INTEGER CHECK (priority_score >= 0 AND priority_score <= 100),
    confidence_score DECIMAL(3,2) CHECK (confidence_score >= 0 AND confidence_score <= 1),

    -- AI Explanations
    pm_explanation TEXT,
    tenant_explanation TEXT,
    reasoning TEXT,        -- Chain-of-thought from Agent 1
    applied_modifiers JSONB,  -- List of priority modifiers

    -- SLA Tracking
    response_deadline TIMESTAMP,
    resolution_deadline TIMESTAMP,
    sla_tier VARCHAR(20),

    -- Status Management
    status VARCHAR(50) NOT NULL DEFAULT 'NEW',
    assigned_vendor_id UUID REFERENCES vendors(id),
    scheduled_start TIMESTAMP,
    scheduled_end TIMESTAMP,
    completed_at TIMESTAMP,
    closed_at TIMESTAMP,

    -- PM Override Tracking
    pm_override BOOLEAN DEFAULT FALSE,
    pm_override_reason TEXT,
    pm_override_at TIMESTAMP,
    pm_override_by UUID REFERENCES users(id),

    -- Metadata
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMP NOT NULL DEFAULT NOW(),

    CONSTRAINT valid_severity CHECK (
        severity IN ('LOW', 'MEDIUM', 'HIGH', 'EMERGENCY')
    ),
    CONSTRAINT valid_status CHECK (
        status IN ('NEW', 'TRIAGED', 'ASSIGNED', 'SCHEDULED',
                   'IN_PROGRESS', 'COMPLETED_PENDING_QC',
                   'CLOSED', 'CANCELLED')
    )
);

-- Triage Log Table (for observability)
CREATE TABLE triage_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    work_order_id UUID NOT NULL REFERENCES work_orders(id),
    agent_version VARCHAR(50),  -- Track prompt version

    -- Input snapshot
    input_description TEXT,
    input_images_count INTEGER,
```

```sql
    context_bundle JSONB,  -- Full context used

    -- Agent outputs
    agent_1_output JSONB,  -- Triage classifier
    agent_2_output JSONB,  -- Priority calculator
    agent_3_output JSONB,  -- Explainer
    agent_4_output JSONB,  -- Confidence evaluator
    agent_5_output JSONB,  -- SLA mapper

    -- Performance metrics
    total_latency_ms INTEGER,
    agent_1_latency_ms INTEGER,
    agent_2_latency_ms INTEGER,
    agent_3_latency_ms INTEGER,
    agent_4_latency_ms INTEGER,

    -- Cost tracking
    total_cost_usd DECIMAL(10,6),

    created_at TIMESTAMP NOT NULL DEFAULT NOW()
);

-- PM Override Log (for learning)
CREATE TABLE pm_overrides (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    work_order_id UUID NOT NULL REFERENCES work_orders(id),
    pm_user_id UUID NOT NULL REFERENCES users(id),

    -- Original AI classification
    original_severity VARCHAR(20),
    original_trade VARCHAR(50),
    original_priority_score INTEGER,
    original_confidence DECIMAL(3,2),

    -- PM corrections
    corrected_severity VARCHAR(20),
    corrected_trade VARCHAR(50),
    corrected_priority_score INTEGER,
    justification TEXT NOT NULL,

    -- Metadata
    created_at TIMESTAMP NOT NULL DEFAULT NOW()
);

-- Vendor Performance Tracking
CREATE TABLE vendor_jobs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    work_order_id UUID NOT NULL REFERENCES work_orders(id),
    vendor_id UUID NOT NULL REFERENCES vendors(id),

    -- Triage accuracy feedback
    was_triage_accurate BOOLEAN,  -- Vendor feedback
    actual_severity VARCHAR(20),  -- What vendor found on-site

    -- Completion tracking
    completed_at TIMESTAMP,
    completion_photos JSONB,
    completion_notes TEXT,
    materials_used JSONB,

    -- PM QA
    pm_approved BOOLEAN,
    pm_rejection_reason TEXT,
    rework_required BOOLEAN,

    created_at TIMESTAMP NOT NULL DEFAULT NOW()
);

-- Create indexes for performance
CREATE INDEX idx_work_orders_status ON work_orders(status);
CREATE INDEX idx_work_orders_priority ON work_orders(priority_score DESC);
CREATE INDEX idx_work_orders_severity ON work_orders(severity);
CREATE INDEX idx_work_orders_response_deadline ON work_orders(response_deadline
    );
CREATE INDEX idx_work_orders_property ON work_orders(property_id);
CREATE INDEX idx_work_orders_tenant ON work_orders(tenant_id);
CREATE INDEX idx_triage_logs_work_order ON triage_logs(work_order_id);
CREATE INDEX idx_pm_overrides_work_order ON pm_overrides(work_order_id);
```

Listing 11: Database Schema (PostgreSQL)

5.4 API Specifications

5.4.1 Core API Endpoints

```python
# FastAPI endpoint definitions

@router.post("/api/v1/maintenance-requests")
async def submit_maintenance_request(
    request: MaintenanceRequestInput,
    context: RequestContext = Depends(get_context)
):
    """
    Submit new maintenance request

    Input:
        - description: str (10-2000 chars)
        - category: enum
        - images: List[UploadFile] (0-5 files, <10MB each)
        - severity_hint: Optional[str]

    Output:
        - work_order_id: UUID
        - status: str
        - estimated_response_time: str

    Latency: ~3s (full AI pipeline)
    """
    # Step 1: Validate input
    validate_request(request)

    # Step 2: Preprocess
    processed = await preprocess_request(request)

    # Step 3: Safety net check
    is_emergency, score = safety_net_check(processed.description)
    if is_emergency:
        return handle_emergency(processed, score)

    # Step 4: Context enrichment (parallel)
    context_bundle = await enrich_context(processed)

    # Step 5: AI Agent orchestration
    triage_result = await orchestrate_agents(processed, context_bundle)

    # Step 6: Save to database
    work_order = await save_work_order(processed, triage_result)

    # Step 7: Route based on confidence
    routing = route_by_confidence(triage_result.confidence,
                                   triage_result.severity)

    # Step 8: Notifications
    await notify_stakeholders(work_order, routing)

    return {
        "work_order_id": work_order.id,
        "status": work_order.status,
        "severity": triage_result.severity,
        "estimated_response_time": format_sla(triage_result.response_deadline),
        "requires_pm_review": routing == "PM_REVIEW_IMMEDIATE"
    }


@router.get("/api/v1/work-orders/{work_order_id}")
async def get_work_order(work_order_id: UUID):
    """
    Retrieve work order details
    """
    work_order = await db.work_orders.get(work_order_id)
    if not work_order:
        raise HTTPException(status_code=404, detail="Work order not found")

    return WorkOrderResponse(
        id=work_order.id,
        status=work_order.status,
        severity=work_order.severity,
        priority_score=work_order.priority_score,
        trade=work_order.trade,
        description=work_order.description,
        images=work_order.images,
        pm_explanation=work_order.pm_explanation,
        tenant_explanation=work_order.tenant_explanation,
        response_deadline=work_order.response_deadline,
        resolution_deadline=work_order.resolution_deadline,
        assigned_vendor=work_order.assigned_vendor,
        timeline=build_timeline(work_order)
    )


@router.patch("/api/v1/work-orders/{work_order_id}/override")
async def pm_override_triage(
    work_order_id: UUID,
    override: TriageOverrideRequest,
    pm_user: User = Depends(require_pm_role)
):
    """
    PM manually overrides AI triage classification

    Input:
        - corrected_severity: enum
        - corrected_trade: enum
        - corrected_priority_score: int
        - justification: str (required)

    Output:
        - updated work order

    Side effects:
        - Logs override for continuous learning
        - Recalculates SLA deadlines
        - Triggers re-routing
    """
    work_order = await db.work_orders.get(work_order_id)

    # Log override for learning
    await db.pm_overrides.create(
        work_order_id=work_order_id,
        pm_user_id=pm_user.id,
        original_severity=work_order.severity,
        original_trade=work_order.trade,
        original_priority_score=work_order.priority_score,
        corrected_severity=override.severity,
        corrected_trade=override.trade,
        corrected_priority_score=override.priority_score,
        justification=override.justification
    )

    # Update work order
    await db.work_orders.update(
        work_order_id,
        severity=override.severity,
        trade=override.trade,
        priority_score=override.priority_score,
        pm_override=True,
        pm_override_reason=override.justification,
        pm_override_at=datetime.now(),
        pm_override_by=pm_user.id
    )

    # Recalculate SLA
    new_sla = calculate_sla_deadlines(override.priority_score, work_order.reported_at)
    await db.work_orders.update(work_order_id, **new_sla)

    return {"status": "success", "work_order": work_order}


@router.get("/api/v1/pm/dashboard")
async def get_pm_dashboard(pm_user: User = Depends(require_pm_role)):
    """
    PM dashboard data

    Returns:
        - immediate_attention: List[WorkOrder] (conf <70% or emergency)
        - pending_review: List[WorkOrder] (conf 70-90%)
        - auto_approved: List[WorkOrder] (conf >90%)
        - performance_metrics: dict
    """
    # Get cases requiring immediate attention
    immediate = await db.work_orders.query(
        where=[
            or_(
                confidence_score < 0.70,
                severity == "EMERGENCY",
                status == "VENDOR_UNRESPONSIVE"
            )
        ],
        order_by=priority_score.desc(),
        limit=50
    )

    # Get cases in review queue
    pending = await db.work_orders.query(
        where=[
            confidence_score >= 0.70,
            confidence_score < 0.90,
            status.in_(["TRIAGED", "ASSIGNED"])
        ],
        order_by=priority_score.desc(),
        limit=100
    )

    # Get auto-approved cases
    auto_approved = await db.work_orders.query(
        where=[confidence_score >= 0.90],
        order_by=created_at.desc(),
        limit=100
    )

    # Calculate performance metrics
    metrics = await calculate_performance_metrics()

    return {
        "immediate_attention": immediate,
        "pending_review": pending,
        "auto_approved": auto_approved,
        "metrics": metrics
    }
```

Listing 12: API Endpoint Specifications (Pseudo-code)

5.5 Analysis

5.6 Alerting Strategy

Alert Name	Condition	Action
red!20 Emergency Miss	Emergency not detected by rules or AI	Page on-call engineer immediately
red!20 API Down	Uptime ¡99% for 5 minutes	Page DevOps team
yellow!20 High PM Override Rate	Override rate ¿20% for 24 hours	Alert ML team, review prompts
yellow!20 Latency Degradation	P95 latency ¿7s for 15 minutes	Alert DevOps, check infrastructure
yellow!20 Cost Spike	Daily cost ¿150% of baseline	Alert engineering lead
blue!20 Low Confidence Spike	¡70% confidence rate ¿10%	Alert ML team, investigate patterns
blue!20 Cache Miss Rate High	Cache hit rate ¡10% for 1 hour	Check Redis, review cache strategy

Table 14: Alert Definitions

5.7 Continuous Learning Feedback Loop

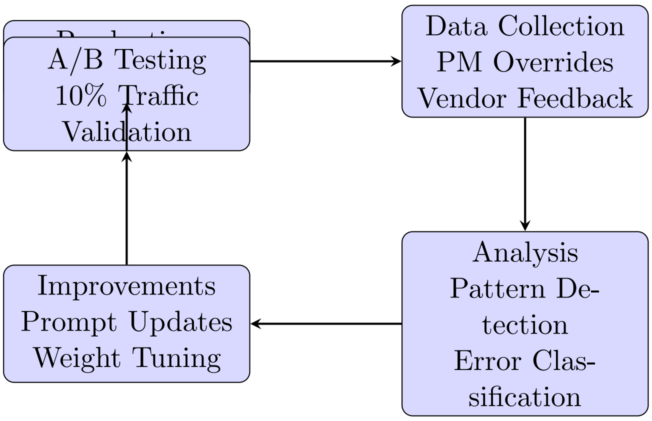


Figure 7: Continuous Learning Feedback Loop

5.7.1 Retraining Cadence

Component	Frequency	Trigger
Prompt Engineering	Weekly	50+ new PM overrides
Priority Weights	Monthly	SLA breach rate ¿5%
Model Fine-tuning	Quarterly	500+ labeled corrections
Safety Rules	As needed	Any emergency miss

Table 15: Retraining Schedule

6 Appendices

6.1 Appendix A: Example Classifications

6.1.1 Example 1: EMERGENCY with Health Symptoms

Example EMERGENCY Case

Input:

- Description: ”Gas smell in basement, very strong, making us dizzy, evacuated to neighbor”
- Category: HVAC
- Images: None
- Time: 11:30 PM Sunday
- Tenant: Family with infant
- Weather: 28°F, winter

Safety Net: Catches ”gas smell” → EMERGENCY, score=100

Agent 1 Output:

```
{
    "severity": "EMERGENCY",
    "trade": "HVAC",
    "reasoning": "Gas leak with health symptoms (dizziness) and evacuation. Life-safety emergency requiring immediate response per protocol.",
    "confidence": 1.0,
    "key_factors": ["gas_leak", "health_symptoms", "evacuation"]
}
```

Agent 2 Output:

```
{
    "priority_score": 100,
    "base_score": 85,
    "applied_modifiers": [
        {"category": "safety", "points": 20, "reason": "gas leak"},
        {"category": "health", "points": 12, "reason": "health symptoms"},
        {"category": "tenant_impact", "points": 10, "reason": "infant present"},
        {"category": "time", "points": 7, "reason": "late night"}
    ],
    "total_modifiers": 49,
    "capped_at": 100
}
```

Agent 3 Output:

```
{
    "pm_explanation": "EMERGENCY: Gas leak with evacuation is life-safety emergency. Health symptoms indicate dangerous exposure. Requires immediate emergency vendor dispatch.",
    "tenant_explanation": "Your request has been marked as an emergency. An emergency technician will contact you within 30 minutes. Please stay evacuated until confirmed safe."
}
```

Agent 4 Output:

```
{
    "confidence": 1.0,
    "routing": "AUTO_APPROVE_EMERGENCY"
}
```

Agent 5 Output:

```
{
    "tier": "EMERGENCY",
    "response_deadline": "2024-12-08 12:00 AM",  # 30 minutes
    "resolution_deadline": "2024-12-08 11:30 PM"  # 24 hours
}
```

Result: Auto-approved, emergency vendor dispatched immediately, PM notified

6.1.2 Example 2: HIGH with Active Water Damage

Example HIGH Case

Input:

- Description: ”Toilet overflowing, water spreading to bedroom, can’t stop it”
- Category: PLUMBING
- Images: 2 photos showing water on floor
- Time: 10:00 PM Saturday
- Tenant: Elderly (78 years old)
- Weather: Normal

Safety Net: No emergency keywords detected, continue to AI

Agent 1 Output:

```
{
    "severity": "HIGH",
    "trade": "PLUMBING",
    "reasoning": "Active overflow with water spreading beyond bathroom. Property damage occurring and toilet is essential service. Elderly tenant increases urgency.",
    "confidence": 0.95,
    "key_factors": ["active_water_damage", "spreading", "essential_service_loss"]
}
```

Agent 2 Output:

```
{
    "priority_score": 78,
    "base_score": 60,
    "applied_modifiers": [
        {"category": "water_damage", "points": 15, "reason": "spreading water"},
        {"category": "tenant_impact", "points": 8, "reason": "elderly"},
        {"category": "essential_service", "points": 12, "reason": "toilet unusable"},
        {"category": "time", "points": 6, "reason": "weekend after hours"}
    ],
    "total_modifiers": 41,
    "capped_at": 84  # MAX for HIGH
}
```

Agent 3 Output:

```
{
    "pm_explanation": "Active overflow with water spreading requires urgent response to prevent property damage. Loss of toilet function with elderly tenant. Recommend same-day emergency plumber.",
    "tenant_explanation": "We understand this is urgent. An emergency plumber will contact you within 2 hours to schedule an immediate visit today."
}
```

Agent 4 Output:

```
{
    "confidence": 0.95,
    "routing": "AUTO_APPROVE"
}
```

Agent 5 Output:

```
{
    "tier": "HIGH",
    "response_deadline": "2024-12-08 12:00 PM",  # Next day noon
    "resolution_deadline": "2024-12-09 10:00 PM"  # 48 hours
}
```

Result: Auto-approved, emergency plumber assigned same-day

6.1.3 Example 3: MEDIUM with Seasonal Context

Example MEDIUM Case

Input:

- Description: ”Water heater making loud banging noises, sounds weird, 15 year old unit”
- Category: PLUMBING
- Images: None
- Time: 2:00 PM Wednesday
- Tenant: Standard (no vulnerabilities)
- Weather: 32°F, winter

Safety Net: No emergency keywords, continue to AI

Agent 1 Output:

```
{
    "severity": "MEDIUM",
    "trade": "PLUMBING",
    "reasoning": "Unusual noise from aging water heater suggests mineral buildup or thermal expansion. Not immediately dangerous but should be inspected within 24-48 hours to prevent failure. Winter season increases priority.",
    "confidence": 0.88,
    "key_factors": ["unusual_noise", "unit_age_15yr", "winter_season"]
}
```

Agent 2 Output:

```
{
    "priority_score": 48,
    "base_score": 30,
    "applied_modifiers": [
        {"category": "seasonal", "points": 8, "reason": "winter, no hot water risk"},
        {"category": "property_risk", "points": 10, "reason": "aging unit may fail"}
    ],
    "total_modifiers": 18,
    "capped_at": null
}
```

Agent 3 Output:

```
{
    "pm_explanation": "Loud banging from 15-year water heater indicates potential failure risk. MEDIUM priority requires 24-48 hour inspection to prevent winter hot water loss.",
    "tenant_explanation": "We’ll have a plumber inspect your water heater within 1-2 business days. The noise suggests it needs maintenance to prevent issues."
}
```

Agent 4 Output:

```
{
    "confidence": 0.88,
    "routing": "AUTO_APPROVE"
}
```

Agent 5 Output:

```
{
    "tier": "MEDIUM",
    "response_deadline": "2024-12-09 6:00 PM",  # 48 business hours
    "resolution_deadline": "2024-12-12 6:00 PM"  # 5 business days
}
```

Result: Auto-approved, standard plumber assigned within 48 hours

6.2 Appendix B: Glossary

Agent A specialized LLM model with a specific task (e.g., Triage Classifier, Priority Calculator)

Confidence Score A value from 0.0 to 1.0 indicating the AI's certainty in its classification

Context Bundle A JSON object containing all enriched contextual data (weather, tenant, property, history)

Emergency Highest severity level (85-100 priority score) requiring immediate response for life-safety issues

Multimodal LLM capability to process both text and images simultaneously

PM Property Manager - the human decision-maker who reviews and approves AI classifications

Priority Score Numerical urgency index from 0-100 combining severity and contextual modifiers

Safety Net Deterministic rules engine that catches life-threatening emergencies before AI processing

SLA Service Level Agreement - contractual response and resolution timeframes

Trade Category of repair work (PLUMBING, ELECTRICAL, HVAC, APPLIANCE, GENERAL, STRUCTURAL)

Triage The process of classifying maintenance requests by severity and urgency

Vector Search Similarity search using embeddings to find past cases matching the current request

6.3 Appendix C: Revision History

Version	Date	Author	Changes
1.0	2024-12-07	Engineering Team	Initial release - Complete architecture specification

Table 16: Document Revision History