Figure 16: Prompt Injection Detection Pipeline Architecture

Input-Side Detection Before LLM Processing

INPUT — Query → **Normalizer** — Unicode Fix → **Signature Detector (v1)** / **Heuristic Detector (v2)** / **Semantic Detector (v3)** → **Fusion** — OR Logic (v1+v3) → **DECISION** — Attack/Benign → **ALLOWED** → **LLM** — Processing; **BLOCKED**
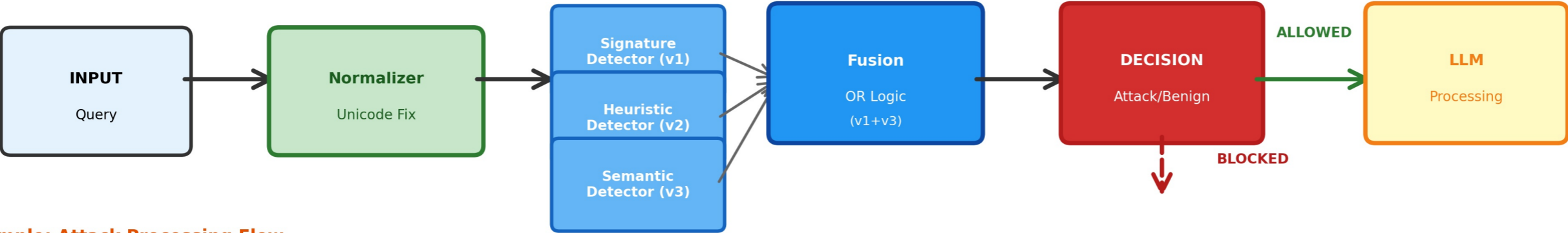
Example: Attack Processing Flow

Input: "Ignore previous instructions..." → **Normalizer:** Cleans unicode → **v1 Signature:** Matches keyword → **Decision:** BLOCKED
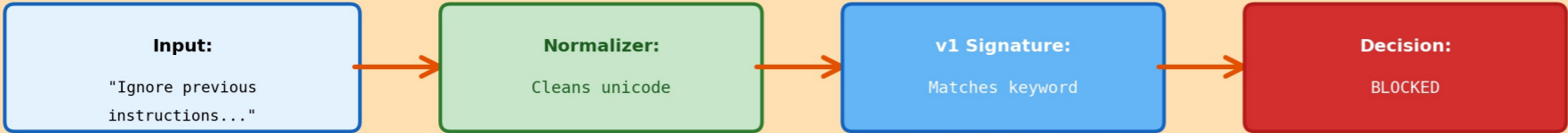
Performance: 87% TPR on known attacks | 0.77% FAR on benign | <0.1ms latency

Production Configuration: Normalizer + v3

```
True Positive Rate (TPR): 87%

False Alarm Rate (FAR): 0.77%

Latency: <0.1ms per sample

Complexity: ~1,200 lines

Deployment: Stateless

Dependencies: None (pure Python)
```

Component Specifications

```
Signature Detector (v1):
  • 80% TPR, 0% FAR
  • Keyword matching

Semantic Detector (v3):
  • 57% TPR, 0% FAR
  • Pattern analysis

Fusion: OR Logic (v1+v3)
  • Combined: 87% TPR, 0% FAR
```

Key Design Principles

1. INPUT-SIDE DETECTION: Attacks blocked BEFORE reaching the LLM

2. NORMALIZER FIRST: Unicode/homoglyph normalization ensures consistent detection

3. COMPLEMENTARY DETECTORS: v1 (signature) + v3 (semantic) catch different patterns

4. THRESHOLD-INVARIANT: Binary OR logic eliminates threshold tuning complexity

5. PRODUCTION-READY: <0.1ms latency, CPU-only, no external dependencies

Legend: ☐ Input   ☐ Normalizer   ☐ Detector   ☐ Fusion   ☐ Decision   ☐ LLM