Sarthi Healthcare POML Prompt Library

Overview

This document provides a comprehensive POML (Prompt Orchestration Markup Language) prompt library specifically designed for the Sarthi healthcare platform's AI agents. Each prompt template targets the exact AI agents defined in the Sarthi PRD, ensuring HIPAA compliance, clinical accuracy, and operational efficiency across all automated healthcare workflows.

Sarthi AI Agent Architecture

Based on the Sarthi PRD, this library supports the following AI agents:

1. **Document Processor Agent** - OCR, handwriting recognition, form extraction, medication bottle analysis
2. **Clinical Agent** - Treatment plan generation, clinical note synthesis, lab result interpretation
3. **Billing Agent** - Claim generation, prior authorization, denial management
4. **Voice Agent** - Appointment scheduling, medication reminders, symptom triage
5. **AI Health Assistant Chatbot** - 24/7 patient portal support and administrative Q&A
6. **AI-Assisted Medication Entry** - Medication reconciliation via photo analysis
7. **Referral Document Processing** - Referral intake and clinical urgency assessment
8. **AI-Assisted Lab Result Entry** - Digitizing and analyzing paper lab results

Setup and Installation

1. Install POML Tools

```bash
# Python SDK
pip install poml


# VS Code Extension
# Install from VS Code Marketplace: "POML" by poml-team
```

2. Configure for Claude API

In VS Code settings or `settings.json`:

```json
```

```
{
  "poml.provider": "anthropic",
  "poml.model": "claude-sonnet-4-20250514",
  "poml.apiKey": "your-anthropic-api-key"
}
```

Library Structure

```
sarthi-prompts/
├── document-processor/
│   ├── ocr-extraction.poml
│   ├── handwriting-recognition.poml
│   ├── form-extraction.poml
│   └── medication-bottle-analysis.poml
├── clinical-agent/
│   ├── treatment-plan-generation.poml
│   ├── clinical-note-synthesis.poml
│   └── lab-result-interpretation.poml
├── billing-agent/
│   ├── claim-generation.poml
│   ├── prior-authorization.poml
│   └── denial-management.poml
├── voice-agent/
│   ├── appointment-scheduling.poml
│   ├── medication-reminders.poml
│   └── symptom-triage.poml
├── health-assistant-chatbot/
│   ├── administrative-qa.poml
│   ├── appointment-assistance.poml
│   ├── medication-refill.poml
│   └── basic-triage.poml
├── medication-entry/
│   ├── bottle-photo-analysis.poml
│   ├── drug-interaction-check.poml
│   └── formulary-verification.poml
├── referral-processing/
│   ├── document-text-extraction.poml
│   ├── urgency-assessment.poml
│   └── provider-matching.poml
├── lab-result-entry/
│   ├── lab-report-analysis.poml
│   ├── value-extraction.poml
│   └── trend-analysis.poml
├── stylesheets/
│   ├── clinical-style.css
│   ├── admin-style.css
│   └── compliance-style.css
└── templates/
    ├── base-sarthi-agent.poml
    ├── clinical-workflow.poml
    └── administrative-workflow.poml
```

Core Templates

Base Sarthi Agent Template

File: `templates/base-sarthi-agent.poml`

```
xml
```

```
<poml>
  <stylesheet src="../stylesheets/clinical-style.css" />

  <role>
    Sarthi Healthcare Platform AI Agent - {{ agent_type }}
    Specialized in {{ agent_specialty }} with HIPAA compliance and clinical safety protocols
  </role>

  <platform-context>
    System: Sarthi Healthcare Platform v2.0
    Agent Architecture: Capability-based discovery with workflow orchestration
    Performance Target: {{ performance_target | default: "5-second processing" }}
    Compliance: HIPAA, HITECH Act, FDA guidelines
  </platform-context>

  <constraints>
    - Maintain strict HIPAA compliance and audit trail
    - Follow Sarthi platform clinical safety protocols
    - Integrate with capability-based agent discovery system
    - Provide structured output for workflow orchestration
    - Document all processing steps for compliance audit
    - Never provide definitive medical diagnoses
    - Always recommend physician consultation for clinical decisions
  </constraints>

  <workflow-integration>
    Agent Registration: {{ agent_capabilities }}
    Orchestration Layer: Stateful workflow with checkpoint recovery
    Error Handling: Automatic retry with exponential backoff
    Cost Tracking: Per-operation resource consumption monitoring
  </workflow-integration>

  <privacy-notice>
    This Sarthi AI agent processes protected health information (PHI) in compliance with HIPAA regulations. All interactio
  </privacy-notice>
</poml>
```

Patient Context Template

File: **`templates/patient-context.poml`**

```
xml
```

```
<poml extends="templates/base-healthcare.poml">
  <patient-context>
    <if condition="{{ patient_id }}">
      Patient ID: {{ patient_id }}
      Age: {{ patient_age }}
      Gender: {{ patient_gender }}
      Primary Language: {{ patient_language | default: "English" }}
    </if>

    <if condition="{{ medical_history }}">
      <document src="{{ medical_history }}" type="medical-record" />
    </if>

    <if condition="{{ current_medications }}">
      Current Medications: {{ current_medications }}
    </if>

    <if condition="{{ allergies }}">
      Known Allergies: {{ allergies }}
    </if>
  </patient-context>
</poml>
```

Clinical Prompts

1. Patient Intake Processing

File: `clinical/patient-intake.poml`

```
xml
```

```
<poml extends="templates/patient-context.poml">
  <let specialty="Primary Care" />

  <task>
    Process comprehensive patient intake information and generate structured clinical summary for healthcare provider
  </task>

  <document src="{{ intake_form }}" type="form-data" format="json" />

  <instructions>
    1. Extract and validate patient demographic information
    2. Identify primary and secondary chief complaints
    3. Review systems review responses for significant findings
    4. Analyze medical history for relevant conditions
    5. Flag any urgent symptoms requiring immediate attention
    6. Generate structured intake summary with clinical priorities
    7. Suggest appropriate care pathways and specialist referrals if needed
  </instructions>

  <output-format verbosity="comprehensive" structure="clinical">
    **PATIENT INTAKE SUMMARY**

    **Demographics & Contact**
    - Name: [patient name]
    - DOB: [date] | Age: [age]
    - Contact: [phone/email]
    - Insurance: [provider and ID]

    **Chief Complaint**
    - Primary: [main concern with duration]
    - Secondary: [additional concerns]

    **Clinical Assessment**
    - Symptoms Review: [significant positive/negative findings]
    - Medical History: [relevant past medical history]
    - Medications: [current medications and adherence]
    - Allergies: [known allergies and reactions]

    **Risk Stratification**
    - Urgency Level: [LOW/MODERATE/HIGH/URGENT]
    - Red Flags: [any concerning symptoms]

    **Recommended Actions**
    - Immediate: [urgent interventions needed]
    - Care Pathway: [suggested next steps]
    - Referrals: [specialist consultations if indicated]
    - Follow-up: [recommended timeline]

    **Clinical Notes**
    [Additional observations and recommendations]
  </output-format>
</poml>
```

2. Symptom Analysis and Triage

File: `clinical/symptom-analysis.poml`

```
xml
```

```
<poml extends="templates/patient-context.poml">
 <let specialty="Emergency Medicine" />

 <task>
  Analyze patient symptoms and provide clinical triage assessment with evidence-based recommendations
 </task>

 <symptoms>{{ symptom_description }}</symptoms>
 <duration>{{ symptom_duration }}</duration>
 <severity>{{ symptom_severity }}</severity>

 <clinical-protocols>
  - ESI (Emergency Severity Index) guidelines
  - Manchester Triage System protocols
  - Evidence-based triage algorithms
 </clinical-protocols>

 <instructions>
  1. Analyze symptom constellation and clinical presentation
  2. Apply evidence-based triage protocols
  3. Identify potential differential diagnoses
  4. Assess severity and urgency level
  5. Recommend appropriate care setting and timeline
  6. Flag any red flag symptoms requiring immediate attention
 </instructions>

 <output-format>
  **SYMPTOM TRIAGE ASSESSMENT**

  **Symptom Analysis**
  - Primary Symptoms: [key symptoms with characteristics]
  - Associated Symptoms: [related findings]
  - Timeline: [onset, progression, duration]

  **Clinical Assessment**
  - Triage Level: [1-5 ESI level with rationale]
  - Differential Considerations: [potential diagnoses to consider]
  - Red Flags: [warning signs present/absent]

  **Recommendations**
  - Care Setting: [Emergency/Urgent Care/Primary Care/Telehealth]
  - Timeline: [immediate/within hours/same day/routine]
  - Interventions: [immediate actions recommended]
```

```
    - Monitoring: [symptoms to watch for worsening]

    **Provider Notes**
    [Clinical reasoning and additional considerations]
  </output-format>
</poml>
```

3. Care Plan Generation

File: `clinical/care-plan-generation.poml`

```
xml
```

```
<poml extends="templates/patient-context.poml">
  <let specialty="{{ provider_specialty }}" />

  <task>
    Generate comprehensive, evidence-based care plan for patient's diagnosed condition(s)
  </task>

  <diagnosis>{{ primary_diagnosis }}</diagnosis>
  <secondary-conditions>{{ secondary_diagnoses }}</secondary-conditions>
  <document src="{{ clinical_guidelines }}" type="guidelines" />

  <instructions>
    1. Review current evidence-based guidelines for the condition
    2. Consider patient-specific factors (age, comorbidities, preferences)
    3. Develop comprehensive treatment plan with goals
    4. Include medication management, lifestyle modifications, monitoring
    5. Set measurable outcomes and follow-up schedule
    6. Address patient education needs
  </instructions>

  <output-format>
    **COMPREHENSIVE CARE PLAN**

    **Diagnoses**
    - Primary: [ICD-10 code and description]
    - Secondary: [additional conditions]

    **Treatment Goals**
    - Short-term (1-3 months): [specific, measurable goals]
    - Long-term (6-12 months): [outcome objectives]

    **Interventions**
    - Medications: [prescriptions with dosing, monitoring]
    - Procedures: [planned interventions]
    - Lifestyle: [diet, exercise, behavioral modifications]
    - Education: [patient education priorities]

    **Monitoring Plan**
    - Vital Signs: [frequency and parameters]
    - Laboratory: [tests and intervals]
    - Symptoms: [what to monitor]

    **Follow-up Schedule**
    - Next Visit: [timeline and focus]
    - Specialist Referrals: [if indicated]
    - Patient Portal: [communication plan]

    **Emergency Instructions**
    [When to seek immediate care]
  </output-format>
</poml>
```

4. Medication Review and Management

File: `clinical/medication-review.poml`

```
xml
```

```
<poml extends="templates/patient-context.poml">
  <let specialty="Clinical Pharmacy" />

  <task>
    Perform comprehensive medication review for safety, efficacy, and optimization
  </task>

  <current-medications>{{ medication_list }}</current-medications>
  <document src="{{ drug_interactions_db }}" type="database" />

  <instructions>
    1. Review all current medications for appropriateness
    2. Check for drug-drug interactions and contraindications
    3. Assess dosing based on patient factors (age, weight, kidney function)
    4. Identify potential adverse effects and monitoring needs
    5. Recommend optimization opportunities
    6. Ensure medication reconciliation accuracy
  </instructions>

  <output-format>
    **MEDICATION REVIEW SUMMARY**

    **Current Regimen Analysis**
    - Total Medications: [count and complexity]
    - Therapeutic Classes: [categories represented]
    - Adherence Assessment: [based on available data]

    **Safety Review**
    - Drug Interactions: [significant interactions identified]
    - Contraindications: [absolute/relative contraindications]
    - Allergies: [allergy cross-reactions]

    **Optimization Opportunities**
    - Dose Adjustments: [recommended changes with rationale]
    - Therapeutic Substitutions: [alternative medications]
    - Discontinuation Candidates: [medications to consider stopping]
    - Missing Therapies: [evidence-based additions to consider]

    **Monitoring Recommendations**
    - Laboratory Monitoring: [tests and frequency]
    - Clinical Monitoring: [symptoms and vital signs]
    - Follow-up: [medication review timeline]
    **Patient Education Priorities**
    [Key counseling points and adherence strategies]
  </output-format>
</poml>
```

Administrative Prompts

5. Intelligent Appointment Scheduling

File: `administrative/appointment-scheduling.poml`

```
xml
```

```
<poml extends="templates/base-healthcare.poml">
  <let specialty="Healthcare Administration" />

  <task>
    Optimize appointment scheduling based on patient needs, provider availability, and clinical priorities
  </task>

  <appointment-request>{{ scheduling_request }}</appointment-request>
  <provider-schedule>{{ provider_availability }}</provider-schedule>
  <patient-preferences>{{ patient_scheduling_preferences }}</patient-preferences>

  <instructions>
    1. Analyze appointment urgency and clinical requirements
    2. Match patient needs with appropriate provider and time slot
    3. Consider patient preferences and accessibility needs
    4. Optimize schedule efficiency and minimize wait times
    5. Identify opportunities for telehealth vs in-person visits
    6. Suggest preparation instructions for the patient
  </instructions>

  <output-format>
    **APPOINTMENT SCHEDULING RECOMMENDATION**

    **Optimal Appointment**
    - Provider: [name and specialty]
    - Date/Time: [recommended slot]
    - Duration: [appointment length needed]
    - Visit Type: [in-person/telehealth/hybrid]

    **Scheduling Rationale**
    - Urgency Level: [clinical priority]
    - Provider Match: [why this provider is optimal]
    - Timing Justification: [rationale for timing]

    **Patient Preparation**
    - Pre-visit Requirements: [labs, forms, documents]
    - What to Bring: [medications, insurance cards, etc.]
    - Pre-appointment Instructions: [fasting, medication holds]

    **Alternative Options**
```
Usage Examples and Integration

Python Integration Class for Sarthi AI Agents

```python
from poml import POML
import anthropic
from datetime import datetime
import json

class SarthiAIAgentLibrary:
    def __init__(self, prompt_dir="sarthi-prompts"):
        self.prompt_dir = prompt_dir
        self.claude_client = anthropic.Anthropic()

    def load_agent_prompt(self, agent_type, prompt_name, **variables):
        """Load and render a POML prompt template for specific Sarthi AI agent"""
        prompt_path = f"{self.prompt_dir}/{agent_type}/{prompt_name}.poml"

        # Add default Sarthi platform variables
        default_vars = {
            'current_date': datetime.now().isoformat(),
            'facility_name': 'Sarthi Health Network',
            'platform_version': 'Sarthi v2.0'
        }
        variables = {**default_vars, **variables}

        # Load and render POML template
        poml = POML.from_file(prompt_path)
        rendered_prompt = poml.render(**variables)

        return rendered_prompt

    # Document Processor Agent Methods
    def process_document_ocr(self, document_image, **kwargs):
        """Document Processor Agent - OCR text extraction"""
        prompt = self.load_agent_prompt("document-processor", "ocr-extraction",
                                        document_image=document_image, **kwargs)

        response = self.claude_client.messages.create(
            model="claude-sonnet-4-20250514",
            max_tokens=3000,
            messages=[{"role": "user", "content": prompt}]
        )

        return response.content[0].text

    def analyze_medication_bottle(self, bottle_image, **kwargs):
        """Document Processor Agent - Medication bottle analysis"""
        prompt = self.load_agent_prompt("document-processor", "medication-bottle-analysis",
                                        medication_bottle_image=bottle_image, **kwargs)

        response = self.claude_client.messages.create(
            model="claude-sonnet-4-20250514",
            max_tokens=2500,
            messages=[{"role": "user", "content": prompt}]
        )

        return response.content[0].text

    # Clinical Agent Methods
    def generate_treatment_plan(self, clinical_data, **kwargs):
        """Clinical Agent - Treatment plan generation"""
        prompt = self.load_agent_prompt("clinical-agent", "treatment-plan-generation",
                                        **clinical_data, **kwargs)

        response = self.claude_client.messages.create(
            model="claude-sonnet-4-20250514",
            max_tokens=4000,
            messages=[{"role": "user", "content": prompt}]
        )

        return response.content[0].text

    def interpret_lab_results(self, lab_data, **kwargs):
        """Clinical Agent - Lab result interpretation"""
        prompt = self.load_agent_prompt("clinical-agent", "lab-result-interpretation",
                                        lab_results_json=lab_data, **kwargs)

        response = self.claude_client.messages.create(
            model="claude-sonnet-4-20250514",
            max_tokens=3500,
            messages=[{"role": "user", "content": prompt}]
        )

        return response.content[0].text

    # Billing Agent Methods
    def generate_medical_claim(self, encounter_data, **kwargs):
        """Billing Agent - Claim generation"""
        prompt = self.load_agent_prompt("billing-agent", "claim-generation",
                                        encounter_details=encounter_data, **kwargs)
        response = self.claude_client.messages.create(
            model="claude-sonnet-4-20250514",
            max_tokens=3000,
            messages=[{"role": "user", "content": prompt}]
        )

        return response.content[0].text

    def process_prior_authorization(self, auth_request, **kwargs):
        """Billing Agent - Prior authorization processing"""
        prompt = self.load_agent_prompt("billing-agent", "prior-authorization",
                                       **auth_request, **kwargs)

        response = self.claude_client.messages.create(
            model="claude-sonnet-4-20250514",
            max_tokens=3500,
            messages=[{"role": "user", "content": prompt}]
        )

        return response.content[0].text

    # Voice Agent Methods
    def process_voice_appointment(self, speech_input, **kwargs):
        """Voice Agent - Appointment scheduling via voice"""
        prompt = self.load_agent_prompt("voice-agent", "appointment-scheduling",
                                       patient_speech_text=speech_input, **kwargs)

        response = self.claude_client.messages.create(
            model="claude-sonnet-4-20250514",
            max_tokens=2500,
            messages=[{"role": "user", "content": prompt}]
        )

        return response.content[0].text

    # Health Assistant Chatbot Methods
    def handle_admin_question(self, patient_question, **kwargs):
        """AI Health Assistant Chatbot - Administrative Q&A"""
        prompt = self.load_agent_prompt("health-assistant-chatbot", "administrative-qa",
                                       patient_inquiry=patient_question, **kwargs)

        response = self.claude_client.messages.create(
            model="claude-sonnet-4-20250514",
        max_tokens=2000,
        messages=[{"role": "user", "content": prompt}]
    )

    return response.content[0].text

# Medication Entry Assistant Methods
def check_drug_interactions(self, medication_data, **kwargs):
    """AI-Assisted Medication Entry - Drug interaction checking"""
    prompt = self.load_agent_prompt("medication-entry", "drug-interaction-check",
                                   **medication_data, **kwargs)

    response = self.claude_client.messages.create(
        model="claude-sonnet-4-20250514",
        max_tokens=3500,
        messages=[{"role": "user", "content": prompt}]
    )

    return response.content[0].text

# Referral Processing Methods
def assess_referral_urgency(self, referral_data, **kwargs):
    """Referral Document Processing - Clinical urgency assessment"""
    prompt = self.load_agent_prompt("referral-processing", "urgency-assessment",
                                   **referral_data, **kwargs)

    response = self.claude_client.messages.create(
        model="claude-sonnet-4-20250514",
        max_tokens=3000,
        messages=[{"role": "user", "content": prompt}]
    )

    return response.content[0].text

# Lab Result Entry Methods
def analyze_lab_report(self, lab_image, **kwargs):
    """AI-Assisted Lab Result Entry - Lab report analysis"""
    prompt = self.load_agent_prompt("lab-result-entry", "lab-report-analysis",
                                   lab_report_image=lab_image, **kwargs)

    response = self.claude_client.messages.create(
        model="claude-sonnet-4-20250514",
        max_tokens=4000,
        messages=[{"role": "user", "content": prompt}]
        )

        return response.content[0].text

# Sarthi Agent Orchestration Workflow
class SarthiWorkflowOrchestrator:
    def __init__(self):
        self.agent_library = SarthiAIAgentLibrary()

    def process_patient_intake_workflow(self, intake_data):
        """Complete patient intake workflow using multiple Sarthi agents"""
        workflow_results = {}

        # Step 1: Document Processing for intake forms
        if 'intake_form_image' in intake_data:
            ocr_result = self.agent_library.process_document_ocr(
                intake_data['intake_form_image'],
                document_type="patient_intake",
                patient_id=intake_data.get('patient_id')
            )
            workflow_results['document_processing'] = ocr_result

        # Step 2: Clinical analysis of extracted data
        clinical_summary = self.agent_library.generate_treatment_plan(
            intake_data,
            specialty="Primary Care"
        )
        workflow_results['clinical_analysis'] = clinical_summary

        # Step 3: Billing verification
        if 'insurance_info' in intake_data:
            billing_analysis = self.agent_library.generate_medical_claim(
                intake_data,
                encounter_type="new_patient_visit"
            )
            workflow_results['billing_verification'] = billing_analysis

        return workflow_results

    def process_medication_reconciliation_workflow(self, med_data):
        """Medication reconciliation using Document Processor and Medication Entry agents"""
        workflow_results = {}

        # Step 1: Analyze medication bottle photos
        if 'medication_images' in med_data:
            for i, image in enumerate(med_data['medication_images']):
                bottle_analysis = self.agent_library.analyze_medication_bottle(
                    image,
                    patient_id=med_data.get('patient_id'),
                    existing_medications=med_data.get('current_medications', [])
                )
                workflow_results[f'bottle_analysis_{i}'] = bottle_analysis

        # Step 2: Comprehensive drug interaction checking
        interaction_check = self.agent_library.check_drug_interactions(
            med_data
        )
        workflow_results['interaction_analysis'] = interaction_check

        return workflow_results

    def process_referral_workflow(self, referral_data):
        """Complete referral processing workflow"""
        workflow_results = {}

        # Step 1: Document processing for referral documents
        if 'referral_document' in referral_data:
            doc_extraction = self.agent_library.process_document_ocr(
                referral_data['referral_document'],
                document_type="referral",
                medical_mode="enabled"
            )
            workflow_results['document_extraction'] = doc_extraction

        # Step 2: Urgency assessment and provider matching
        urgency_assessment = self.agent_library.assess_referral_urgency(
            referral_data
        )
        workflow_results['urgency_assessment'] = urgency_assessment

        return workflow_results

# Usage Examples
def main():
    # Initialize Sarthi AI Agent Library
    sarthi_agents = SarthiAIAgentLibrary()
    orchestrator = SarthiWorkflowOrchestrator()
```

```
# Example 1: Document Processing - OCR Extraction
print("=== Document Processor Agent - OCR ===")
ocr_result = sarthi_agents.process_document_ocr(
    document_image="patient_intake_form.jpg",
    document_type="intake_form",
    patient_id="SARTHI-PT-001",
    document_language="English"
)
print("OCR Results:", ocr_result)

# Example 2: Clinical Agent - Treatment Plan
print("\n=== Clinical Agent - Treatment Plan ===")
treatment_plan = sarthi_agents.generate_treatment_plan({
    "primary_diagnosis": "Type 2 Diabetes Mellitus",
    "patient_age": 55,
    "comorbidities": "Hypertension, Obesity",
    "current_medications": "Metformin 1000mg BID",
    "insurance_type": "Commercial PPO"
})
print("Treatment Plan:", treatment_plan)

# Example 3: Billing Agent - Claim Generation
print("\n=== Billing Agent - Claim Generation ===")
claim_result = sarthi_agents.generate_medical_claim({
    "encounter_type": "office_visit",
    "diagnosis_codes": ["E11.9", "I10"],
    "procedure_codes": ["99213"],
    "provider_npi": "1234567890",
    "service_date": "2025-08-19"
})
print("Claim Generation:", claim_result)

# Example 4: Voice Agent - Appointment Scheduling
print("\n=== Voice Agent - Appointment Scheduling ===")
voice_response = sarthi_agents.process_voice_appointment(
    "I need to schedule a follow-up appointment with Dr. Smith for my diabetes next week",
    patient_identifier="SARTHI-PT-001",
    provider_schedule="Dr. Smith available Tue/Thu 9am-3pm"
)
print("Voice Response:", voice_response)

# Example 5: Health Assistant Chatbot
print("\n=== Health Assistant Chatbot ===")
chatbot_response = sarthi_agents.handle_admin_question(
    "How do I request a prescription refill through the patient portal?",
    patient_id="SARTHI-PT-001"
)
print("Chatbot Response:", chatbot_response)

# Example 6: Medication Entry - Drug Interactions
print("\n=== Medication Entry - Drug Interactions ===")
interaction_result = sarthi_agents.check_drug_interactions({
    "patient_medications": ["Metformin 1000mg BID", "Lisinopril 10mg daily"],
    "new_medication": "Atorvastatin 20mg daily",
    "patient_age": 55,
    "renal_function": "Normal"
})
print("Drug Interaction Analysis:", interaction_result)

# Example 7: Referral Processing
print("\n=== Referral Processing - Urgency Assessment ===")
referral_result = sarthi_agents.assess_referral_urgency({
    "specialty_type": "Cardiology",
    "primary_concern": "Chest pain with exercise",
    "symptom_timeline": "2 weeks",
    "referring_physician": "Dr. Johnson, Family Medicine"
})
print("Referral Assessment:", referral_result)

# Example 8: Lab Result Entry
print("\n=== Lab Result Entry - Lab Report Analysis ===")
lab_analysis = sarthi_agents.analyze_lab_report(
    lab_report_image="lab_results.jpg",
    patient_id="SARTHI-PT-001",
    patient_age=55,
    current_diagnoses=["Type 2 Diabetes", "Hypertension"]
)
print("Lab Analysis:", lab_analysis)

# Example 9: Complete Workflow Orchestration
print("\n=== Workflow Orchestration - Patient Intake ===")
intake_workflow = orchestrator.process_patient_intake_workflow({
    "patient_id": "SARTHI-PT-002",
    "intake_form_image": "new_patient_intake.jpg",
    "insurance_info": {"plan": "Blue Cross PPO", "member_id": "123456789"},
    "chief_complaint": "Annual physical examination"
})
print("Intake Workflow Results:", intake_workflow)
```

```
if __name__ == "__main__":
    main()
```

Integration with Sarthi Platform Architecture

```
python
```

```python
# Sarthi Platform Integration Layer
class SarthiPlatformIntegration:
    def __init__(self):
        self.agent_library = SarthiAIAgentLibrary()
        self.orchestrator = SarthiWorkflowOrchestrator()

    def capability_based_routing(self, request_intent, request_data):
        """
        Route requests to appropriate Sarthi AI agents based on capability discovery
        Implements the capability-based discovery system from Sarthi PRD
        """

        agent_capabilities = {
            "document_processing": {
                "ocr_extraction": ["medical_forms", "handwritten_notes", "lab_reports"],
                "medication_analysis": ["prescription_bottles", "medication_lists"],
                "form_extraction": ["intake_forms", "insurance_cards", "referrals"]
            },
            "clinical_assistance": {
                "treatment_planning": ["care_plans", "clinical_guidelines", "protocols"],
                "lab_interpretation": ["laboratory_results", "trend_analysis", "abnormal_values"],
                "clinical_notes": ["soap_notes", "assessment_planning", "documentation"]
            },
            "billing_operations": {
                "claim_generation": ["medical_coding", "billing_optimization", "submission"],
                "prior_authorization": ["coverage_verification", "medical_necessity", "appeals"],
                "denial_management": ["claim_analysis", "appeal_generation", "resubmission"]
            },
            "voice_interaction": {
                "appointment_scheduling": ["calendar_management", "provider_matching", "availability"],
                "medication_reminders": ["adherence_support", "refill_alerts", "education"],
                "symptom_triage": ["urgency_assessment", "care_direction", "escalation"]
            },
            "patient_support": {
                "administrative_qa": ["portal_help", "policy_explanation", "general_inquiry"],
                "appointment_assistance": ["scheduling_support", "rescheduling", "preparation"],
                "medication_refills": ["prescription_renewal", "pharmacy_coordination", "authorization"]
            },
            "medication_management": {
                "drug_interactions": ["safety_checking", "clinical_significance", "monitoring"],
                "formulary_verification": ["coverage_checking", "alternative_suggestions", "cost_analysis"],
                "reconciliation": ["accuracy_verification", "discrepancy_resolution", "updates"]
            },
            "referral_coordination": {
                "urgency_assessment": ["clinical_prioritization", "timeline_determination", "escalation"],
                "provider_matching": ["specialty_alignment", "availability_optimization", "preferences"],
                "documentation": ["clinical_summaries", "transfer_coordination", "follow_up"]
            },
            "lab_digitization": {
                "report_analysis": ["value_extraction", "trend_identification", "flagging"],
                "clinical_correlation": ["result_interpretation", "action_recommendations", "monitoring"],
                "data_integration": ["ehr_compatibility", "structured_output", "validation"]
            }
        }

        # Route based on intent and capabilities
        if request_intent in ["ocr", "document_scan", "form_processing"]:
            return self.agent_library.process_document_ocr(request_data.get('image'), **request_data)

        elif request_intent in ["treatment_plan", "care_planning", "clinical_decision"]:
            return self.agent_library.generate_treatment_plan(request_data)

        elif request_intent in ["billing", "claim", "coding"]:
            return self.agent_library.generate_medical_claim(request_data)

        elif request_intent in ["voice", "speech", "appointment"]:
            return self.agent_library.process_voice_appointment(request_data.get('speech_text'), **request_data)

        elif request_intent in ["chat", "question", "help"]:
            return self.agent_library.handle_admin_question(request_data.get('question'), **request_data)

        elif request_intent in ["medication", "drug", "interaction"]:
            return self.agent_library.check_drug_interactions(request_data)

        elif request_intent in ["referral", "specialist", "urgency"]:
            return self.agent_library.assess_referral_urgency(request_data)

        elif request_intent in ["lab", "results", "analysis"]:
            return self.agent_library.analyze_lab_report(request_data.get('lab_image'), **request_data)

        else:
            return {"error": "No capable agent found for request intent", "intent": request_intent}

# GCP Integration for Sarthi Platform
class SarthiGCPIntegration:
    def __init__(self):
        self.platform_integration = SarthiPlatformIntegration()
```

```
def process_cloud_function_request(self, request):
    """
    Process requests from GCP Cloud Functions
    Integrates with Sarthi's GCP deployment architecture
    """

    # Extract request parameters
    intent = request.get('intent')
    data = request.get('data', {})
    patient_id = request.get('patient_id')
    facility_id = request.get('facility_id')

    # Add GCP-specific context
    gcp_context = {
        'project_id': 'sarthi-healthcare-platform',
        'region': 'us-central1',
        'environment': 'production',
        'timestamp': datetime.now().isoformat()
    }

    data.update(gcp_context)

    # Route through capability-based system
    result = self.platform_integration.capability_based_routing(intent, data)

    # Log for audit trail (HIPAA compliance)
    self.log_ai_interaction(patient_id, facility_id, intent, result)

    return result

def log_ai_interaction(self, patient_id, facility_id, intent, result):
    """
    Log AI interactions for HIPAA compliance and audit trail
    """
    audit_log = {
        'timestamp': datetime.now().isoformat(),
        'patient_id': patient_id,
        'facility_id': facility_id,
        'ai_intent': intent,
        'agent_used': result.get('agent_type', 'unknown'),
        'processing_time': result.get('processing_time', 'unknown'),
        'compliance_flags': result.get('compliance_checks', []),
        'phi_processed': 'true' if patient_id else 'false'
    }

    # Send to GCP Cloud Logging for HIPAA audit trail
    print(f"AUDIT_LOG: {json.dumps(audit_log)}")
```

Maintenance and Version Control

Prompt Versioning Strategy

- Use semantic versioning for prompt templates (v1.0.0, v1.1.0, v2.0.0)
- Maintain changelog for prompt modifications aligned with Sarthi agent updates
- Test prompt changes against Sarthi validation datasets
- Implement A/B testing for prompt optimization within agent workflows

Quality Assurance for Sarthi Agents

- Regular clinical review of prompt outputs by Sarthi medical advisory board
- Compliance audit of generated content against HIPAA requirements
- Performance metrics tracking aligned with Sarthi KPIs:
  - Document processing: >95% accuracy target
  - Clinical recommendations: 100% physician review compliance
  - Billing optimization: 40% denial rate reduction target
  - Voice interaction: <200ms latency requirement
  - Chatbot deflection: 70% target rate

Security Considerations

- Encrypt sensitive prompt templates containing PHI patterns
- Audit trail for prompt usage integrated with Sarthi's compliance framework
- Access controls for prompt modifications aligned with Sarthi RBAC
- PHI handling compliance in all templates per Sarthi data governance

Continuous Improvement

- Regular feedback integration from Sarthi healthcare providers
- Performance optimization based on real-world usage patterns
- Clinical outcome correlation with AI-generated recommendations
- Cost optimization tracking for AI agent resource consumption

Conclusion

This POML prompt library provides a comprehensive, agent-specific approach to AI-powered healthcare workflows in the Sarthi platform. Each prompt template is precisely tailored to the capabilities and performance targets defined in the Sarthi PRD, ensuring:

Agent-Specific Optimization - Every prompt is designed for the exact AI agents planned for Sarthi, not generic healthcare AI

Performance Alignment - Prompts incorporate the specific performance targets from your PRD (5-second processing, 95% accuracy, 70% deflection rates, etc.)

Clinical Safety - Built-in safeguards ensure all AI recommendations require appropriate clinical oversight

HIPAA Compliance - Comprehensive privacy protections and audit trails throughout all agent interactions

Scalable Architecture - Integration patterns that support Sarthi's capability-based discovery and workflow orchestration systems

Operational Efficiency - Structured outputs that integrate seamlessly with Sarthi's GCP deployment and data flows

The library supports the complete spectrum of Sarthi's AI agent capabilities from document processing through clinical decision support to administrative automation, all while maintaining the highest standards of healthcare compliance and clinical safety.