ADDITIONAL CRITICAL RESOURCES FOR JARVIS ENHANCEMENT

Essential Tools, Templates, and Advanced Implementation Support

Author: Manus AI (Original Concept Creator)
Date: June 8, 2025
Version: 1.0
Purpose: Complete resource package for advanced JARVIS implementation success

---

Advanced Prompt Engineering Templates

DeepSeek R1 Advanced Prompt Library

System Architecture Analysis Template:

```
 JARVIS SYSTEM ARCHITECTURE ANALYSIS

Current Implementation Status:
- Hardware: i7-12700H + RTX 3050 Ti + 16GB RAM
- Phase: [Current implementation phase]
- Components Active: [List active components]
- Performance Metrics: [Current performance data]

Analysis Request:
[Specific analysis requirement]

Required Deliverables:
1. Technical feasibility assessment with resource impact
analysis
2. Performance optimization recommendations with expected
improvements
3. Risk assessment with mitigation strategies
4. Implementation priority ranking with rationale
5. Resource allocation optimization strategy

Context Constraints:
- 4GB VRAM limitation requiring intelligent model management
- 16GB RAM constraint requiring efficient memory allocation
- Thermal management for sustained AI processing
- Windows 11 + WSL2 integration requirements
- Multi-AI coordination complexity

Please provide detailed technical analysis with specific,
actionable recommendations and quantified expected outcomes.
```

Performance Optimization Analysis Template:

```
 JARVIS PERFORMANCE OPTIMIZATION REQUEST

Current Performance Profile:
- Language Processing: [Current response times]
- Computer Vision: [Current FPS and accuracy]
- Speech Processing: [Current latency and accuracy]
- Memory Usage: [Current RAM/VRAM utilization]
- CPU Utilization: [Current CPU usage patterns]

Optimization Target:
[Specific performance improvement goal]

Analysis Requirements:
1. Bottleneck identification with root cause analysis
2. Optimization strategy development with implementation steps
3. Resource reallocation recommendations
4. Performance trade-off analysis
5. Implementation timeline with milestones

Hardware Optimization Context:
- Hybrid CPU architecture (6 P-cores + 8 E-cores)
- GPU memory constraint management
- Thermal throttling prevention
- Power efficiency optimization

Please provide comprehensive optimization strategy with
specific implementation steps and expected performance
improvements.
```

ChatGPT Optimization Templates

User Experience Design Template:

```
 JARVIS USER EXPERIENCE ENHANCEMENT

Project Context: Creating fictional-grade AI assistant with
natural, intuitive interaction patterns that rival JARVIS from
Iron Man.
```

```
Current Capabilities:
- Advanced language processing with emotional intelligence
- Real-time computer vision and object recognition
- Natural speech recognition and synthesis
- Proactive assistance and autonomous task execution
- Multi-modal interaction coordination

UX Design Challenge:
[Specific user experience improvement area]

Design Requirements:
1. Natural conversation flow that feels human-like and
intelligent
2. Seamless multi-modal interaction (speech, vision, text,
gesture)
3. Proactive assistance that anticipates needs without being
intrusive
4. Emotional intelligence and empathy in all interactions
5. Personality consistency that builds user trust and engagement

Target User Scenarios:
- Daily productivity assistance and task management
- Creative collaboration and brainstorming
- Technical problem-solving and system management
- Learning and information discovery
- Entertainment and casual conversation

Please provide detailed UX design recommendations including
interaction patterns, conversation templates, and user
engagement strategies that create a magical, effortless
experience.
```

Creative Problem-Solving Template:

```
 JARVIS INNOVATION BREAKTHROUGH REQUEST

Challenge Context: Achieving fictional-grade AI capabilities
within hardware constraints while maintaining real-time
performance and user experience excellence.

Current Limitations:
- 4GB VRAM constraining model size and complexity
- 16GB RAM limiting simultaneous AI model operation
- Laptop thermal constraints affecting sustained performance
- Need for seamless multi-AI coordination
- Real-time processing requirements across multiple modalities

Innovation Areas:
[Specific innovation challenge]
```

```
Creative Requirements:
1. Novel approaches that transcend conventional limitations
2. Innovative resource sharing and optimization strategies
3. Creative integration patterns for multi-AI coordination
4. Breakthrough user interaction paradigms
5. Unique solutions that create competitive advantages

Inspiration Sources:
- Fictional AI assistants (JARVIS, HAL 9000, FRIDAY, Cortana)
- Cutting-edge AI research and emerging technologies
- Creative applications of existing technologies
- Novel combinations of different AI approaches

Please provide innovative, creative solutions that push
boundaries and create breakthrough capabilities within our
constraints.
```

Blackbox AI Autonomous Implementation Templates

Autonomous Capability Development Template:

```
 BLACKBOX AI AUTONOMOUS IMPLEMENTATION TASK

Objective: Implement autonomous AI capabilities that enable
JARVIS to operate independently while maintaining safety and
user alignment.

Hardware Configuration:
- CPU: Intel i7-12700H (6 P-cores @ 4.6GHz, 8 E-cores @ 3.5GHz)
- GPU: NVIDIA RTX 3050 Ti (4GB VRAM, 2560 CUDA cores)
- RAM: 16GB DDR4
- OS: Windows 11 + WSL2 Ubuntu 22.04

Implementation Requirements:
1. Autonomous decision-making framework with safety constraints
2. Proactive task execution with user preference learning
3. Multi-modal input processing and response coordination
4. Resource-aware processing with dynamic optimization
5. Error handling and recovery mechanisms

Code Generation Focus:
[Specific autonomous capability to implement]

Technical Constraints:
- VRAM management for 4GB limitation
- CPU workload distribution across hybrid architecture
- Memory allocation optimization for 16GB constraint
- Thermal monitoring and performance scaling
- Integration with existing JARVIS components
```

```
Safety Requirements:
- User override capabilities for all autonomous actions
- Confidence thresholds for autonomous vs. assisted decisions
- Logging and explanation for all autonomous decisions
- Fail-safe mechanisms for error conditions

Please generate complete, production-ready code with
comprehensive error handling, performance optimization, and
safety mechanisms.

MANUAL_INTERVENTION_PROTOCOL:
If you encounter implementation obstacles or need clarification:
**MANUAL_INTERVENTION_REQUIRED**
Situation: [Describe current implementation status]
Obstacle: [Specific technical or design challenge]
Required Assistance: [What help is needed to proceed]
**END_INTERVENTION_REQUEST**
```

Advanced Configuration Templates

Hardware Optimization Configuration

NVIDIA GPU Optimization Script:

```
#!/usr/bin/env python3
"""
JARVIS GPU Optimization Configuration
Optimizes RTX 3050 Ti for AI workloads within 4GB VRAM
constraint
"""

import torch
import nvidia_ml_py3 as nvml
import psutil
import time
from typing import Dict, List, Optional

class JARVISGPUOptimizer:
    def __init__(self):
        self.device = torch.device("cuda" if
torch.cuda.is_available() else "cpu")
        self.max_vram_gb = 4.0
        self.safety_margin_gb = 0.5
        self.available_vram_gb = self.max_vram_gb -
self.safety_margin_gb

        # Initialize NVIDIA ML for monitoring
        nvml.nvmlInit()
        self.gpu_handle = nvml.nvmlDeviceGetHandleByIndex(0)

    def optimize_gpu_settings(self):
        """Configure optimal GPU settings for JARVIS
workloads"""
        # Set memory fraction for PyTorch
        torch.cuda.set_per_process_memory_fraction(
            self.available_vram_gb / self.max_vram_gb
        )

        # Enable memory mapping for large models
        torch.backends.cuda.matmul.allow_tf32 = True
        torch.backends.cudnn.allow_tf32 = True

        # Optimize CUDA cache
        torch.cuda.empty_cache()

        return self.get_gpu_status()

    def get_gpu_status(self) -> Dict:
        """Get current GPU status and utilization"""
        memory_info =
nvml.nvmlDeviceGetMemoryInfo(self.gpu_handle)
        utilization =
nvml.nvmlDeviceGetUtilizationRates(self.gpu_handle)
        temperature =
nvml.nvmlDeviceGetTemperature(self.gpu_handle,
nvml.NVML_TEMPERATURE_GPU)

        return {
            'memory_used_gb': memory_info.used / (1024**3),
            'memory_free_gb': memory_info.free / (1024**3),
            'memory_total_gb': memory_info.total / (1024**3),
            'gpu_utilization': utilization.gpu,
            'memory_utilization': utilization.memory,
            'temperature_c': temperature
        }

    def monitor_thermal_throttling(self) -> bool:
        """Monitor for thermal throttling conditions"""
        temp = nvml.nvmlDeviceGetTemperature(self.gpu_handle,
nvml.NVML_TEMPERATURE_GPU)
        # RTX 3050 Ti throttles around 83°C
        return temp > 80

    def intelligent_model_swapping(self, models: List[str],
current_task: str) -> str:
        """Implement intelligent model swapping based on task
requirements"""
        # Priority-based model loading
        task_priorities = {
            'conversation': ['language_model'],
            'vision': ['computer_vision_model'],
            'speech': ['speech_recognition_model',
'speech_synthesis_model'],
            'multi_modal': ['language_model',
'computer_vision_model']
        }

        required_models = task_priorities.get(current_task,
['language_model'])

        # Check available VRAM
        status = self.get_gpu_status()
        available_vram = status['memory_free_gb']

        # Implement swapping logic based on available memory
        if available_vram < 1.0:  # Less than 1GB free
            return "swap_required"
        else:
            return "sufficient_memory"

# Usage example for JARVIS implementation
if __name__ == "__main__":
    optimizer = JARVISGPUOptimizer()
    status = optimizer.optimize_gpu_settings()
    print(f"GPU Optimization Complete: {status}")
```

CPU Optimization Configuration:

```
#!/usr/bin/env python3
"""
JARVIS CPU Optimization for i7-12700H Hybrid Architecture
Optimizes workload distribution between P-cores and E-cores
"""

import os
import psutil
import threading
from typing import List, Dict
import subprocess

class JARVISCPUOptimizer:
    def __init__(self):
        self.cpu_count = psutil.cpu_count()
        self.p_cores = list(range(0, 12))  # P-cores with
hyperthreading
        self.e_cores = list(range(12, 20))  # E-cores

    def set_ai_process_affinity(self, pid: int, task_type: str):
        """Set CPU affinity for AI processes based on task
type"""
```

```
        try:
            process = psutil.Process(pid)

            if task_type in ['inference', 'training',
'critical']:
                # Use P-cores for AI inference and critical
tasks
                process.cpu_affinity(self.p_cores)
                process.nice(-10)  # Higher priority
            elif task_type in ['background', 'monitoring',
'logging']:
                # Use E-cores for background tasks
                process.cpu_affinity(self.e_cores)
                process.nice(10)  # Lower priority
            else:
                # Default to all cores

process.cpu_affinity(list(range(self.cpu_count)))

        except psutil.NoSuchProcess:
            print(f"Process {pid} not found")
        except psutil.AccessDenied:
            print(f"Access denied for process {pid}")

    def optimize_system_processes(self):
        """Optimize system processes for AI workload
performance"""
        # Move non-essential processes to E-cores
        for proc in psutil.process_iter(['pid', 'name',
'cpu_percent']):
            try:
                if proc.info['name'] in ['dwm.exe',
'explorer.exe', 'winlogon.exe']:

self.set_ai_process_affinity(proc.info['pid'], 'background')
            except (psutil.NoSuchProcess, psutil.AccessDenied):
                continue

    def monitor_cpu_performance(self) -> Dict:
        """Monitor CPU performance and thermal status"""
        cpu_percent = psutil.cpu_percent(interval=1,
percpu=True)
        cpu_freq = psutil.cpu_freq(percpu=True)

        # Separate P-core and E-core utilization
        p_core_usage = [cpu_percent[i] for i in self.p_cores]
        e_core_usage = [cpu_percent[i] for i in self.e_cores]

        return {
            'p_core_avg_usage': sum(p_core_usage) /
len(p_core_usage),
            'e_core_avg_usage': sum(e_core_usage) /
len(e_core_usage),
            'total_cpu_usage': psutil.cpu_percent(),
            'cpu_frequencies': cpu_freq,
            'thermal_throttling':
self.check_thermal_throttling()
        }

    def check_thermal_throttling(self) -> bool:
        """Check for CPU thermal throttling"""
        try:
            # Check CPU temperature using Windows WMI
            result = subprocess.run([
                'powershell',
                'Get-WmiObject -Namespace "root/
OpenHardwareMonitor" -Class Sensor | Where-Object {$_.SensorType
-eq "Temperature" -and $_.Name -like "*CPU*"} | Select-Object
Value'
            ], capture_output=True, text=True)

            if result.returncode == 0 and result.stdout:
                # Parse temperature (simplified)
                return "throttling" in result.stdout.lower()
        except:
            pass

        return False

# Integration with JARVIS main process
def optimize_jarvis_cpu():
    optimizer = JARVISCPUOptimizer()
    optimizer.optimize_system_processes()
    return optimizer.monitor_cpu_performance()
```

Emergency Response Procedures

Critical Failure Recovery Scripts

System Recovery Automation:

```
#!/bin/bash
# JARVIS Emergency Recovery Script
# Automatically diagnoses and recovers from common system
failures

JARVIS_HOME="/home/ubuntu/jarvis"
LOG_FILE="/var/log/jarvis_recovery.log"
BACKUP_DIR="/home/ubuntu/jarvis_backup"
```

```
log_message() {
    echo "$(date '+%Y-%m-%d %H:%M:%S') - $1" | tee -a
"$LOG_FILE"
}

check_gpu_status() {
    log_message "Checking GPU status..."
    if nvidia-smi > /dev/null 2>&1; then
        log_message "GPU accessible"
        return 0
    else
        log_message "GPU not accessible - attempting driver
recovery"
        sudo systemctl restart nvidia-persistenced
        sleep 5
        if nvidia-smi > /dev/null 2>&1; then
            log_message "GPU recovery successful"
            return 0
        else
            log_message "GPU recovery failed - manual
intervention required"
            return 1
        fi
    fi
}

check_memory_status() {
    log_message "Checking memory status..."
    MEMORY_USAGE=$(free | grep Mem | awk '{print ($3/$2) *
100.0}')
    if (( $(echo "$MEMORY_USAGE > 90" | bc -l) )); then
        log_message "High memory usage detected: ${MEMORY_USAGE}
%"
        log_message "Attempting memory cleanup..."

        # Clear system caches
        sudo sync && sudo sysctl vm.drop_caches=3

        # Restart memory-intensive services
        sudo systemctl restart jarvis-language-model
        sudo systemctl restart jarvis-computer-vision

        log_message "Memory cleanup completed"
    else
        log_message "Memory usage normal: ${MEMORY_USAGE}%"
    fi
}

check_ai_services() {
    log_message "Checking AI service status..."

    SERVICES=("jarvis-main" "jarvis-language-model" "jarvis-
computer-vision" "jarvis-speech-processing")

    for service in "${SERVICES[@]}"; do
        if systemctl is-active --quiet "$service"; then
            log_message "$service is running"
        else
            log_message "$service is not running - attempting
restart"
            sudo systemctl restart "$service"
            sleep 10
            if systemctl is-active --quiet "$service"; then
                log_message "$service restart successful"
            else
                log_message "$service restart failed - manual
intervention required"
            fi
        fi
    done
}

backup_critical_data() {
    log_message "Creating emergency backup..."

    # Create backup directory with timestamp
    BACKUP_TIMESTAMP=$(date '+%Y%m%d_%H%M%S')
    CURRENT_BACKUP_DIR="${BACKUP_DIR}/emergency_$
{BACKUP_TIMESTAMP}"
    mkdir -p "$CURRENT_BACKUP_DIR"

    # Backup configuration files
    cp -r "$JARVIS_HOME/config" "$CURRENT_BACKUP_DIR/"
    cp -r "$JARVIS_HOME/models" "$CURRENT_BACKUP_DIR/"
    cp -r "$JARVIS_HOME/logs" "$CURRENT_BACKUP_DIR/"

    log_message "Emergency backup completed:
$CURRENT_BACKUP_DIR"
}

main() {
    log_message "JARVIS Emergency Recovery Started"

    # Create backup before attempting recovery
    backup_critical_data

    # Check and recover system components
    check_gpu_status
    check_memory_status
    check_ai_services

    log_message "JARVIS Emergency Recovery Completed"
}
```

```
# Run recovery if called directly
if [[ "${BASH_SOURCE[0]}" == "${0}" ]]; then
    main "$@"
fi
```

Performance Monitoring Dashboard

Real-Time Monitoring Script:

```
#!/usr/bin/env python3
"""
JARVIS Real-Time Performance Monitoring Dashboard
Provides comprehensive system monitoring for JARVIS enhancement
project
"""

import time
import psutil
import nvidia_ml_py3 as nvml
import json
import threading
from datetime import datetime
from typing import Dict, List
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from collections import deque

class JARVISMonitor:
    def __init__(self, history_length: int = 100):
        self.history_length = history_length
        self.metrics_history = {
            'timestamp': deque(maxlen=history_length),
            'cpu_usage': deque(maxlen=history_length),
            'memory_usage': deque(maxlen=history_length),
            'gpu_usage': deque(maxlen=history_length),
            'gpu_memory': deque(maxlen=history_length),
            'gpu_temperature': deque(maxlen=history_length)
        }

        # Initialize NVIDIA ML
        try:
            nvml.nvmlInit()
            self.gpu_handle = nvml.nvmlDeviceGetHandleByIndex(0)
            self.gpu_available = True
        except:
            self.gpu_available = False

        self.monitoring = False
        self.monitor_thread = None
    def collect_metrics(self) -> Dict:
        """Collect current system metrics"""
        timestamp = datetime.now()

        # CPU metrics
        cpu_usage = psutil.cpu_percent(interval=0.1)

        # Memory metrics
        memory = psutil.virtual_memory()
        memory_usage = memory.percent

        # GPU metrics
        gpu_usage = 0
        gpu_memory = 0
        gpu_temperature = 0

        if self.gpu_available:
            try:
                utilization =
nvml.nvmlDeviceGetUtilizationRates(self.gpu_handle)
                memory_info =
nvml.nvmlDeviceGetMemoryInfo(self.gpu_handle)
                temperature =
nvml.nvmlDeviceGetTemperature(self.gpu_handle,
nvml.NVML_TEMPERATURE_GPU)

                gpu_usage = utilization.gpu
                gpu_memory = (memory_info.used /
memory_info.total) * 100
                gpu_temperature = temperature
            except:
                pass

        metrics = {
            'timestamp': timestamp,
            'cpu_usage': cpu_usage,
            'memory_usage': memory_usage,
            'gpu_usage': gpu_usage,
            'gpu_memory': gpu_memory,
            'gpu_temperature': gpu_temperature
        }

        return metrics

    def update_history(self, metrics: Dict):
        """Update metrics history"""
        for key, value in metrics.items():
            if key in self.metrics_history:
                self.metrics_history[key].append(value)

    def check_alerts(self, metrics: Dict) -> List[str]:
        """Check for alert conditions"""
        alerts = []

        if metrics['cpu_usage'] > 90:
            alerts.append(f"HIGH CPU USAGE:
{metrics['cpu_usage']:.1f}%")

        if metrics['memory_usage'] > 85:
            alerts.append(f"HIGH MEMORY USAGE:
{metrics['memory_usage']:.1f}%")

        if metrics['gpu_temperature'] > 80:
            alerts.append(f"HIGH GPU TEMPERATURE:
{metrics['gpu_temperature']}°C")

        if metrics['gpu_memory'] > 90:
            alerts.append(f"HIGH GPU MEMORY USAGE:
{metrics['gpu_memory']:.1f}%")

        return alerts

    def start_monitoring(self):
        """Start continuous monitoring"""
        self.monitoring = True
        self.monitor_thread =
threading.Thread(target=self._monitor_loop)
        self.monitor_thread.daemon = True
        self.monitor_thread.start()

    def stop_monitoring(self):
        """Stop monitoring"""
        self.monitoring = False
        if self.monitor_thread:
            self.monitor_thread.join()

    def _monitor_loop(self):
        """Main monitoring loop"""
        while self.monitoring:
            metrics = self.collect_metrics()
            self.update_history(metrics)

            # Check for alerts
            alerts = self.check_alerts(metrics)
            if alerts:
                print(f"\n⚠️  ALERTS at
{metrics['timestamp'].strftime('%H:%M:%S')}:")
                for alert in alerts:
                    print(f"   {alert}")

            # Print current status
            print(f"\r{metrics['timestamp'].strftime('%H:%M:
%S')} | "
                  f"CPU: {metrics['cpu_usage']:5.1f}% | "
                  f"RAM: {metrics['memory_usage']:5.1f}% | "
                  f"GPU: {metrics['gpu_usage']:5.1f}% | "
                  f"VRAM: {metrics['gpu_memory']:5.1f}% | "
                  f"Temp: {metrics['gpu_temperature']:3.0f}°C",
end='')

            time.sleep(1)

    def generate_report(self) -> Dict:
        """Generate performance report"""
        if not self.metrics_history['timestamp']:
            return {"error": "No data available"}

        # Calculate averages
        avg_cpu = sum(self.metrics_history['cpu_usage']) /
len(self.metrics_history['cpu_usage'])
        avg_memory = sum(self.metrics_history['memory_usage']) /
len(self.metrics_history['memory_usage'])
        avg_gpu = sum(self.metrics_history['gpu_usage']) /
len(self.metrics_history['gpu_usage'])
        avg_gpu_memory =
sum(self.metrics_history['gpu_memory']) /
len(self.metrics_history['gpu_memory'])
        avg_temperature =
sum(self.metrics_history['gpu_temperature']) /
len(self.metrics_history['gpu_temperature'])

        # Calculate peaks
        max_cpu = max(self.metrics_history['cpu_usage'])
        max_memory = max(self.metrics_history['memory_usage'])
        max_gpu = max(self.metrics_history['gpu_usage'])
        max_gpu_memory = max(self.metrics_history['gpu_memory'])
        max_temperature =
max(self.metrics_history['gpu_temperature'])

        report = {
            'monitoring_duration_minutes':
len(self.metrics_history['timestamp']),
            'averages': {
                'cpu_usage': avg_cpu,
                'memory_usage': avg_memory,
                'gpu_usage': avg_gpu,
                'gpu_memory_usage': avg_gpu_memory,
                'gpu_temperature': avg_temperature
            },
            'peaks': {
                'max_cpu_usage': max_cpu,
                'max_memory_usage': max_memory,
                'max_gpu_usage': max_gpu,
                'max_gpu_memory_usage': max_gpu_memory,
                'max_gpu_temperature': max_temperature
            },
            'recommendations':
self._generate_recommendations(avg_cpu, avg_memory, avg_gpu,
max_temperature)
        }

        return report

    def _generate_recommendations(self, avg_cpu: float,
avg_memory: float, avg_gpu: float, max_temp: float) ->
List[str]:
        """Generate optimization recommendations"""
        recommendations = []

        if avg_cpu > 70:
            recommendations.append("Consider optimizing CPU-
intensive tasks or upgrading CPU cooling")

        if avg_memory > 75:
            recommendations.append("Memory usage is high -
consider model optimization or memory cleanup")

        if avg_gpu < 30:
            recommendations.append("GPU utilization is low -
consider increasing batch sizes or model complexity")

        if max_temp > 75:
            recommendations.append("GPU temperature is high -
improve cooling or reduce workload intensity")

        if not recommendations:
            recommendations.append("System performance is
optimal")

        return recommendations

# Usage example
if __name__ == "__main__":
    monitor = JARVISMonitor()

    print("JARVIS Performance Monitor Started")
    print("Press Ctrl+C to stop monitoring and generate report")

    try:
        monitor.start_monitoring()

        # Keep monitoring until interrupted
        while True:
            time.sleep(1)

    except KeyboardInterrupt:
        print("\n\nStopping monitor...")
        monitor.stop_monitoring()
        # Generate and display report
        report = monitor.generate_report()
        print("\n" + "="*50)
        print("PERFORMANCE REPORT")
        print("="*50)
        print(json.dumps(report, indent=2))
```

Project Management and Coordination Tools

Multi-AI Task Coordination Template

Task Distribution Framework:

```
#!/usr/bin/env python3
"""
JARVIS Multi-AI Task Coordination Framework
Manages task distribution between DeepSeek R1, ChatGPT, and
Blackbox AI
"""

from enum import Enum
from dataclasses import dataclass
from typing import Dict, List, Optional, Any
import json
import time
from datetime import datetime

class AISystem(Enum):
    DEEPSEEK_R1 = "deepseek_r1"
    CHATGPT = "chatgpt"
    BLACKBOX_AI = "blackbox_ai"
    MANUS = "manus"

class TaskType(Enum):
    ANALYSIS = "analysis"
    DESIGN = "design"
    IMPLEMENTATION = "implementation"
    OPTIMIZATION = "optimization"
    TESTING = "testing"
    DOCUMENTATION = "documentation"

class Priority(Enum):
    CRITICAL = 1
    HIGH = 2
    MEDIUM = 3
    LOW = 4
```

```
@dataclass
class Task:
    id: str
    title: str
    description: str
    task_type: TaskType
    priority: Priority
    assigned_ai: Optional[AISystem]
    dependencies: List[str]
    estimated_duration: int  # minutes
    status: str = "pending"
    created_at: datetime = None
    completed_at: Optional[datetime] = None
    result: Optional[Any] = None

    def __post_init__(self):
        if self.created_at is None:
            self.created_at = datetime.now()

class JARVISTaskCoordinator:
    def __init__(self):
        self.tasks: Dict[str, Task] = {}
        self.ai_capabilities = {
            AISystem.DEEPSEEK_R1: {
                'strengths': ['analysis', 'optimization',
'reasoning', 'planning'],
                'optimal_for': [TaskType.ANALYSIS,
TaskType.OPTIMIZATION],
                'current_load': 0,
                'max_concurrent': 3
            },
            AISystem.CHATGPT: {
                'strengths': ['design', 'documentation',
'creativity', 'user_experience'],
                'optimal_for': [TaskType.DESIGN,
TaskType.DOCUMENTATION],
                'current_load': 0,
                'max_concurrent': 2  # Free tier limitation
            },
            AISystem.BLACKBOX_AI: {
                'strengths': ['implementation', 'coding',
'automation', 'testing'],
                'optimal_for': [TaskType.IMPLEMENTATION,
TaskType.TESTING],
                'current_load': 0,
                'max_concurrent': 5
            }
        }

    def create_task(self, title: str, description: str,
task_type: TaskType,
                   priority: Priority, dependencies: List[str] =
None,
                   estimated_duration: int = 60) -> str:
        """Create a new task"""
        task_id = f"task_{int(time.time())}_{len(self.tasks)}"

        task = Task(
            id=task_id,
            title=title,
            description=description,
            task_type=task_type,
            priority=priority,
            assigned_ai=None,
            dependencies=dependencies or [],
            estimated_duration=estimated_duration
        )

        self.tasks[task_id] = task
        return task_id

    def assign_optimal_ai(self, task_id: str) ->
Optional[AISystem]:
        """Assign task to optimal AI system"""
        task = self.tasks.get(task_id)
        if not task:
            return None

        # Check dependencies
        if not self._dependencies_completed(task):
            return None

        # Find optimal AI based on task type and current load
        best_ai = None
        best_score = -1

        for ai_system, capabilities in
self.ai_capabilities.items():
            if capabilities['current_load'] >=
capabilities['max_concurrent']:
                continue

            # Calculate suitability score
            score = 0
            if task.task_type in capabilities['optimal_for']:
                score += 10

            # Priority bonus
            score += (5 - task.priority.value)

            # Load penalty
            score -= capabilities['current_load'] * 2

            if score > best_score:
                best_score = score
                best_ai = ai_system

        if best_ai:
            task.assigned_ai = best_ai
            task.status = "assigned"
            self.ai_capabilities[best_ai]['current_load'] += 1

        return best_ai

    def _dependencies_completed(self, task: Task) -> bool:
        """Check if all task dependencies are completed"""
        for dep_id in task.dependencies:
            dep_task = self.tasks.get(dep_id)
            if not dep_task or dep_task.status != "completed":
                return False
        return True

    def complete_task(self, task_id: str, result: Any = None):
        """Mark task as completed"""
        task = self.tasks.get(task_id)
        if task and task.assigned_ai:
            task.status = "completed"
            task.completed_at = datetime.now()
            task.result = result
            self.ai_capabilities[task.assigned_ai]
['current_load'] -= 1

    def get_pending_tasks(self, ai_system: Optional[AISystem] =
None) -> List[Task]:
        """Get pending tasks for specific AI or all"""
        tasks = []
        for task in self.tasks.values():
            if task.status == "pending" and
self._dependencies_completed(task):
                if ai_system is None or task.assigned_ai ==
ai_system:
                    tasks.append(task)

        # Sort by priority and creation time
        tasks.sort(key=lambda t: (t.priority.value,
t.created_at))
        return tasks

    def generate_status_report(self) -> Dict:
        """Generate comprehensive status report"""
        total_tasks = len(self.tasks)
        completed_tasks = len([t for t in self.tasks.values() if
t.status == "completed"])
        pending_tasks = len([t for t in self.tasks.values() if
t.status == "pending"])
        assigned_tasks = len([t for t in self.tasks.values() if
t.status == "assigned"])

        ai_workload = {}
        for ai_system, capabilities in
self.ai_capabilities.items():
            ai_tasks = [t for t in self.tasks.values() if
t.assigned_ai == ai_system]
            ai_workload[ai_system.value] = {
                'current_load': capabilities['current_load'],
                'max_concurrent':
capabilities['max_concurrent'],
                'total_assigned': len(ai_tasks),
                'completed': len([t for t in ai_tasks if
t.status == "completed"]),
                'utilization': capabilities['current_load'] /
capabilities['max_concurrent'] * 100
            }

        return {
            'timestamp': datetime.now().isoformat(),
            'task_summary': {
                'total': total_tasks,
                'completed': completed_tasks,
                'pending': pending_tasks,
                'assigned': assigned_tasks,
                'completion_rate': completed_tasks / total_tasks
* 100 if total_tasks > 0 else 0
            },
            'ai_workload': ai_workload,
            'next_actions': self._get_next_actions()
        }

    def _get_next_actions(self) -> List[str]:
        """Get recommended next actions"""
        actions = []

        # Check for unassigned high-priority tasks
        high_priority_pending = [t for t in self.tasks.values()
                               if t.status == "pending" and
t.priority in [Priority.CRITICAL, Priority.HIGH]]

        if high_priority_pending:
            actions.append(f"Assign
{len(high_priority_pending)} high-priority pending tasks")

        # Check for overloaded AI systems
        for ai_system, capabilities in
self.ai_capabilities.items():
            if capabilities['current_load'] >=
capabilities['max_concurrent']:
                actions.append(f"{ai_system.value} is at
capacity - consider task redistribution")
```

```python
        # Check for blocked tasks
        blocked_tasks = [t for t in self.tasks.values()
                        if t.status == "pending" and not self._dependencies_completed(t)]

        if blocked_tasks:
            actions.append(f"{len(blocked_tasks)} tasks blocked by dependencies")

        return actions

# Example usage for JARVIS project coordination
def setup_jarvis_project_tasks():
    coordinator = JARVISTaskCoordinator()

    # Phase 1: Foundation Setup
    foundation_tasks = [
        ("Environment Setup Analysis", "Analyze optimal development environment configuration", TaskType.ANALYSIS, Priority.CRITICAL),
        ("Hardware Optimization Strategy", "Develop hardware optimization strategy for i7-12700H + RTX 3050 Ti", TaskType.OPTIMIZATION, Priority.CRITICAL),
        ("Development Environment Implementation", "Implement optimized development environment", TaskType.IMPLEMENTATION, Priority.HIGH),
        ("Foundation Testing", "Test and validate foundation setup", TaskType.TESTING, Priority.HIGH)
    ]

    # Phase 2: Core AI Models
    model_tasks = [
        ("Language Model Selection Analysis", "Analyze optimal language model for hardware constraints", TaskType.ANALYSIS, Priority.HIGH),
        ("Computer Vision Model Optimization", "Optimize computer vision models for real-time processing", TaskType.OPTIMIZATION, Priority.HIGH),
        ("Speech Processing Implementation", "Implement speech recognition and synthesis", TaskType.IMPLEMENTATION, Priority.MEDIUM),
        ("Model Integration Testing", "Test multi-model integration and performance", TaskType.TESTING, Priority.HIGH)
    ]

    # Create tasks with dependencies
    task_ids = {}

    # Foundation tasks
    for i, (title, desc, task_type, priority) in enumerate(foundation_tasks):
        deps = [task_ids[foundation_tasks[i-1][0]]] if i > 0 else []
        task_id = coordinator.create_task(title, desc, task_type, priority, deps)
        task_ids[title] = task_id

    # Model tasks (depend on foundation completion)
    foundation_completion = task_ids["Foundation Testing"]
    for title, desc, task_type, priority in model_tasks:
        deps = [foundation_completion] if "Analysis" not in title else [foundation_completion]
        task_id = coordinator.create_task(title, desc, task_type, priority, deps)
        task_ids[title] = task_id

    return coordinator, task_ids

if __name__ == "__main__":
    coordinator, task_ids = setup_jarvis_project_tasks()

    # Auto-assign tasks
    for task_id in task_ids.values():
        assigned_ai = coordinator.assign_optimal_ai(task_id)
        if assigned_ai:
            print(f"Task {task_id} assigned to {assigned_ai.value}")

    # Generate status report
    report = coordinator.generate_status_report()
    print("\nProject Status Report:")
    print(json.dumps(report, indent=2))
```

This comprehensive resource package provides everything needed for successful JARVIS enhancement implementation, including advanced prompt templates, hardware optimization scripts, emergency recovery procedures, performance monitoring tools, and project coordination frameworks. These resources ensure that you have all the tools necessary to achieve fictional-grade AI assistant capabilities while maintaining system reliability and performance.