MVP-Leadership Intelligence System - Customization Guide

Overview

This guide explains how to customize the MVP-Leadership Intelligence System to match your specific leadership style, organization structure, and workflow preferences.

Configuration Customization

1. Time Windows and Data Sources

Conversation Analysis Window

```python
# In LifelogProcessor.fetch_lifelogs()
# Default: 3 days (2 days ago through today)
if start_date is None:
    start_date = (datetime.now() - timedelta(days=5)).date().isoformat()  # 5 days
```

Email Date Range

```python
# In EmailManager.fetch_emails()
# Default: Yesterday through tomorrow
yesterday = (datetime.now(CST) - timedelta(days=3)).strftime('%Y/%m/%d')  # 3 days ago
tomorrow = (datetime.now(CST) + timedelta(days=1)).strftime('%Y/%m/%d')   # Tomorrow
```

Task Due Date Filter

```python
# In TaskManager._parse_task_data()
# Default: Tasks due within next week
next_week = now + timedelta(days=14)  # Show tasks due in next 2 weeks
```

2. Priority and Categorization

Task Priority Levels

python

```python
# In SummaryGenerator._prepare_summary_data()
# Default: P1 and P2 only
if priority in ['P1', 'P2', 'P3']:  # Add P3 for broader view
```

Task Areas/Categories

python

```python
# Update these in your Google Sheets and modify validation as needed
valid_areas = ['T&T', 'Wedding', 'Personal', 'Business', 'Family', 'Health']
```

Email Priority Labels

python

```python
# In EmailManager.fetch_emails()
queries = {
    "inbox": "in:inbox",
    "sent": f"label:sent after:{yesterday} before:{tomorrow}",
    "priority": "label:\"! - YourName\" is:unread OR label:\"Urgent\" is:unread",  # Multiple l
    "team": "label:\"Team\" is:unread",  # Add team-specific emails
}
```

AI Prompt Customization

1. Conversation Analysis Rules

Modify Commitment Detection

```python
# In PromptTemplates.DETAILED_LIFELOG_SUMMARY
# Add your specific commitment phrases
DETAILED_LIFELOG_SUMMARY = (
    "You are [YourName]'s executive assistant analyzing conversations...\n\n"
    "COMMITMENT PHRASES TO WATCH FOR:\n"
    "- I will, I'll, I should, I need to, I might\n"
    "- I promise, I commit to, I agree to\n"
    "- Let me, I can, I'll make sure\n"
    "- [Add your specific phrases]\n\n"
    # ... rest of prompt
)
```

Customize Time-Based Categorization

```python
# In PromptTemplates.LEADERSHIP_SUMMARY
# Modify the calendar vs todo categorization rules
CALENDAR_PHRASES = [
    "on this day", "at this time", "next week", "by tomorrow", "this Friday",
    "Monday morning", "end of week", "by [specific date]",
    "[Add your organization's time phrases]"
]

TODO_PHRASES = [
    "I will", "I should", "I might", "I need to", "I'll follow up",
    "I'll check", "I'll verify", "I'll confirm",
    "[Add your common action phrases]"
]
```

2. Leadership Insights

Customize Insight Focus Areas

```python
# In PromptTemplates.LEADERSHIP_SUMMARY
# Modify the leadership insight section
"🧠 LEADERSHIP INSIGHT\n"
"----------------------------------------\n"
"[Focus on specific areas like:\n"
" - Communication effectiveness\n"
" - Team development\n"
" - Strategic thinking\n"
" - Work-life balance\n"
" - [Your specific development areas]]\n\n"
```

Industry-Specific Analysis

```python
# Add industry context to prompts
INDUSTRY_CONTEXT = """
INDUSTRY: [Your Industry - e.g., Technology, Manufacturing, Healthcare]
FOCUS AREAS:
- [Industry-specific priorities]
- [Regulatory requirements]
- [Key performance indicators]
"""

# Then add this context to your prompts
```

Output Customization

1. File Naming and Location

Custom Output Directory Structure

python

```python
class Config:
    def __init__(self):
        # Organize by date
        today = datetime.now().strftime("%Y-%m")
        self.output_dir = rf"G:\Leadership Intelligence\{today}"

        # Or organize by type
        self.detailed_output_dir = r"G:\Leadership Intelligence\Detailed Analysis"
        self.summary_output_dir = r"G:\Leadership Intelligence\Daily Summaries"
```

Custom File Naming

python

```python
# In save_detailed_lifelog_summary() and SummaryGenerator._save_and_open_summary()
# Add your naming convention
date_str = datetime.now().strftime("%Y-%m-%d")
time_str = datetime.now().strftime("%H%M")
filename = f"{date_str}_JD_Leadership_Intelligence_{time_str}.txt"  # Add your initials
```

2. Summary Format Customization

Add Custom Sections

python

```python
# In PromptTemplates.LEADERSHIP_SUMMARY
# Add sections relevant to your role
"📊 FINANCIAL REVIEW\n"
"----------------------------------------\n"
"[Budget items, expenses, revenue discussions from conversations]\n\n"

"🌟 TEAM DEVELOPMENT\n"
"----------------------------------------\n"
"[Mentions of team growth, training, performance discussions]\n\n"

"🎯 STRATEGIC INITIATIVES\n"
"----------------------------------------\n"
"[Long-term projects, strategic decisions, planning discussions]\n\n"
```

Modify Section Priorities

python

```python
# Reorder sections based on your priorities
LEADERSHIP_SUMMARY = (
    "🎯 [YOUR NAME]'S LEADERSHIP INTELLIGENCE SUMMARY\n"
    # Put your most important sections first
    "📅 TODAY'S CALENDAR\n"           # Always keep calendar first
    "🔥 URGENT PRIORITIES\n"          # Your custom urgent section
    "📋 HIGH-PRIORITY TASKS\n"        # Standard tasks
    # ... continue with your preferred order
)
```

Advanced Customization

1. Multiple Calendar Integration

Different Calendar Types

python

```python
class Config:
    def __init__(self):
        self.calendar_configs = {
            'work': ['primary', 'team-calendar@company.com'],
            'personal': ['personal@gmail.com'],
            'shared': ['shared-project@group.calendar.google.com']
        }
```

Calendar-Specific Analysis

python

```python
# In CalendarManager.fetch_events()
# Add calendar source tracking
for cal_id in calendar_ids:
    # Add calendar type to events
    for event in events:
        event['calendar_source'] = self._get_calendar_type(cal_id)
```

2. Team-Specific Customization

Multi-Person Analysis

python

```python
# For team leaders managing multiple people
TEAM_ANALYSIS_PROMPT = """
Analyze conversations for commitments involving team members:
- Direct reports: [List names]
- Peer managers: [List names]
- Executive team: [List names]

Categorize commitments by:
- Upward commitments (to superiors)
- Downward commitments (to direct reports)
- Lateral commitments (to peers)
"""
```

Department-Specific Rules

python

```python
# Add department context
DEPARTMENT_CONTEXT = {
    'sales': {
        'focus_areas': ['pipeline', 'quotas', 'customer meetings'],
        'key_phrases': ['deal', 'proposal', 'customer', 'revenue']
    },
    'engineering': {
        'focus_areas': ['sprints', 'releases', 'technical debt'],
        'key_phrases': ['deployment', 'bug', 'feature', 'architecture']
    }
}
```

3. Integration Customization

Additional Data Sources

```python
# Template for adding new data sources
class SlackManager:
    def fetch_messages(self):
        # Add Slack integration
        pass

class TeamsManager:
    def fetch_messages(self):
        # Add Microsoft Teams integration
        pass
```

Custom Task Management Systems

```python
# For non-Google Sheets task management
class AsanaManager:
    def fetch_tasks(self):
        # Add Asana integration
        pass

class TrelloManager:
    def fetch_cards(self):
        # Add Trello integration
        pass
```

Model and Performance Customization

1. AI Model Selection

Choose Models Based on Needs

python

```python
class OpenAIManager:
    def get_completion(self, messages, task_type="general"):
        # Different models for different tasks
        model_selection = {
            "detailed_analysis": "gpt-4",           # Most capable for complex analysis
            "task_extraction": "gpt-4o-mini",      # Faster for simple extraction
            "summarization": "gpt-4",               # High quality for summaries
            "quick_insights": "gpt-4o-mini"        # Fast for simple insights
        }
        model = model_selection.get(task_type, "gpt-4")
```

Adjust Token Limits

python

```python
# In analyze_lifelogs()
max_chars_per_chunk = 30000  # Increase for more context (costs more)
max_chars_per_chunk = 20000  # Decrease for faster processing (less context)
```

2. Processing Optimization

Chunking Strategy

python

```python
# In _analyze_conversations_in_chunks()
# Prioritize recent conversations
conversations.sort(key=lambda x: (x['date'], -x['length']), reverse=True)

# Or prioritize by importance
conversations.sort(key=lambda x: self._calculate_importance(x), reverse=True)
```

Parallel Processing

```python
# For large datasets, add parallel processing
import concurrent.futures

def process_chunks_parallel(self, chunks):
    with concurrent.futures.ThreadPoolExecutor(max_workers=3) as executor:
        futures = [executor.submit(self._analyze_single_chunk, chunk) for chunk in chunks]
        results = [future.result() for future in futures]
    return results
```

Organization-Specific Templates

1. Startup Environment

```python
STARTUP_CONFIG = {
    'focus_areas': ['product', 'funding', 'hiring', 'market_fit'],
    'meeting_types': ['standup', 'investor', 'customer', 'team'],
    'priorities': ['P0', 'P1', 'P2'],  # Different priority system
    'time_horizon': 'weeks'  # Faster-paced environment
}
```

2. Enterprise Environment

```python
ENTERPRISE_CONFIG = {
    'focus_areas': ['strategy', 'compliance', 'governance', 'stakeholder_mgmt'],
    'meeting_types': ['board', 'executive', 'department', 'project_review'],
    'priorities': ['Critical', 'High', 'Medium', 'Low'],
    'time_horizon': 'quarters',  # Longer planning cycles
    'approval_chains': True,  # Track approval requirements
    'budget_tracking': True   # Include budget discussions
}
```

3. Non-Profit Environment

```python
NONPROFIT_CONFIG = {
    'focus_areas': ['fundraising', 'programs', 'volunteers', 'community'],
    'meeting_types': ['donor', 'board', 'program', 'volunteer'],
    'priorities': ['Mission Critical', 'High Impact', 'Standard'],
    'grant_tracking': True,   # Track grant deadlines
    'volunteer_coordination': True
}
```

Role-Specific Customization

1. CEO/Executive Director

```python
CEO_FOCUS = {
    'strategic_commitments': ['board reporting', 'investor relations', 'strategic partnerships'
    'operational_oversight': ['department reviews', 'budget approvals', 'policy decisions'],
    'external_relations': ['media', 'industry events', 'customer relationships'],
    'time_blocking': ['deep work', 'strategic thinking', 'team development']
}
```

2. Department Manager

```python
MANAGER_FOCUS = {
    'team_development': ['1-on-1s', 'performance reviews', 'training'],
    'project_delivery': ['milestones', 'resource allocation', 'timeline management'],
    'cross_functional': ['stakeholder alignment', 'dependency management'],
    'reporting': ['status updates', 'metrics reporting', 'escalations']
}
```

3. Sales Leader

```python
SALES_FOCUS = {
    'pipeline_management': ['deals', 'forecasting', 'customer meetings'],
    'team_performance': ['quota tracking', 'coaching', 'territory management'],
    'customer_relations': ['executive relationships', 'contract negotiations'],
    'market_intelligence': ['competitive analysis', 'market trends']
}
```

Custom Analysis Rules

1. Industry-Specific Commitment Detection

Healthcare/Medical

```python
HEALTHCARE_COMMITMENTS = {
    'patient_care': ['patient follow-up', 'treatment plans', 'consults'],
    'compliance': ['regulatory reporting', 'quality measures', 'audits'],
    'research': ['study protocols', 'data analysis', 'publication deadlines'],
    'phrases': ['patient', 'treatment', 'protocol', 'compliance', 'study']
}
```

Technology/Software

```python
TECH_COMMITMENTS = {
    'development': ['feature delivery', 'bug fixes', 'technical debt'],
    'operations': ['deployments', 'monitoring', 'incident response'],
    'product': ['roadmap updates', 'user research', 'metrics review'],
    'phrases': ['deploy', 'release', 'feature', 'bug', 'user story', 'sprint']
}
```

Manufacturing

```python
MANUFACTURING_COMMITMENTS = {
    'production': ['line efficiency', 'quality control', 'maintenance'],
    'supply_chain': ['vendor management', 'inventory', 'logistics'],
    'safety': ['incident reports', 'training', 'compliance'],
    'phrases': ['production', 'quality', 'supplier', 'inventory', 'safety']
}
```

2. Communication Style Analysis

Direct Communication Style

```python
DIRECT_STYLE_PROMPTS = """
Focus on explicit commitments and clear deadlines.
Look for direct language: "I will do X by Y"
Pay attention to specific dates and times mentioned.
Identify clear accountability assignments.
"""
```

Collaborative Communication Style

```python
COLLABORATIVE_STYLE_PROMPTS = """
Look for consensus-building language and shared commitments.
Identify team decisions and group responsibilities.
Track follow-up items from collaborative discussions.
Note when decisions require team input or approval.
"""
```

Advanced Integration Examples

1. CRM Integration Template

python

```python
class CRMIntegration:
    def __init__(self, crm_type='salesforce'):
        self.crm_type = crm_type

    def extract_customer_commitments(self, conversations):
        """Extract customer-related commitments for CRM updates"""
        customer_mentions = []
        for conv in conversations:
            # Look for customer names, deals, proposals
            # Extract follow-up actions for CRM
            pass
        return customer_mentions

    def sync_to_crm(self, commitments):
        """Sync extracted commitments to CRM tasks"""
        # Implementation depends on your CRM
        pass
```

2. Project Management Integration

python

```python
class ProjectManagementIntegration:
    def __init__(self, pm_tool='asana'):
        self.pm_tool = pm_tool

    def extract_project_commitments(self, conversations):
        """Extract project-related commitments"""
        project_items = []
        # Look for project names, milestones, deliverables
        return project_items

    def create_pm_tasks(self, commitments):
        """Create tasks in project management tool"""
        # Implementation depends on your PM tool
        pass
```

3. Team Communication Integration

python

```python
class TeamCommIntegration:
    def __init__(self, platform='slack'):
        self.platform = platform

    def extract_team_commitments(self, conversations):
        """Extract commitments made to team members"""
        team_items = []
        # Look for team member names and commitments
        return team_items

    def send_team_reminders(self, commitments):
        """Send reminders to team members"""
        # Implementation for Slack/Teams notifications
        pass
```

Performance and Cost Optimization

1. Token Usage Optimization

python

```python
# Reduce token usage for cost savings
class TokenOptimizer:
    def optimize_conversation_length(self, conversations):
        """Reduce conversation length while preserving key information"""
        optimized = []
        for conv in conversations:
            # Remove filler words, redundant phrases
            # Keep key commitments and context
            optimized_content = self._extract_key_phrases(conv['content'])
            conv['content'] = optimized_content
            optimized.append(conv)
        return optimized

    def _extract_key_phrases(self, content):
        """Extract only commitment-related phrases"""
        key_phrases = []
        commitment_indicators = ['I will', 'I should', 'I need to', 'I promise']
        # Implementation to extract relevant sentences
        return "\n".join(key_phrases)
```

2. Intelligent Caching

```python
class AnalysisCache:
    def __init__(self):
        self.cache_dir = "cache"
        os.makedirs(self.cache_dir, exist_ok=True)

    def cache_conversation_analysis(self, conv_id, analysis):
        """Cache analysis results to avoid re-processing"""
        cache_file = os.path.join(self.cache_dir, f"{conv_id}.json")
        with open(cache_file, 'w') as f:
            json.dump(analysis, f)

    def get_cached_analysis(self, conv_id):
        """Retrieve cached analysis if available"""
        cache_file = os.path.join(self.cache_dir, f"{conv_id}.json")
        if os.path.exists(cache_file):
            with open(cache_file, 'r') as f:
                return json.load(f)
        return None
```

Testing and Validation

1. Custom Validation Rules

python

```python
class ValidationRules:
    def validate_commitments(self, extracted_commitments):
        """Validate that extracted commitments make sense"""
        valid_commitments = []
        for commitment in extracted_commitments:
            if self._has_clear_action(commitment):
                if self._has_identifiable_person(commitment):
                    valid_commitments.append(commitment)
        return valid_commitments

    def _has_clear_action(self, commitment):
        """Check if commitment has a clear action"""
        action_words = ['send', 'call', 'review', 'complete', 'follow up']
        return any(word in commitment.lower() for word in action_words)

    def _has_identifiable_person(self, commitment):
        """Check if commitment mentions a specific person"""
        # Implementation to identify person names
        return True  # Simplified
```

2. Quality Metrics

python

```python
class QualityMetrics:
    def calculate_analysis_quality(self, analysis_result):
        """Calculate quality metrics for analysis"""
        metrics = {
            'commitment_specificity': self._measure_specificity(analysis_result),
            'person_identification': self._measure_person_id(analysis_result),
            'deadline_clarity': self._measure_deadline_clarity(analysis_result),
            'actionability': self._measure_actionability(analysis_result)
        }
        return metrics

    def _measure_specificity(self, analysis):
        """Measure how specific the commitments are"""
        # Implementation for specificity scoring
        return 0.85  # Example score
```

Deployment Customization

1. Multi-Environment Setup

python

```python
class EnvironmentConfig:
    def __init__(self, environment='development'):
        self.environment = environment
        self.configs = {
            'development': {
                'chunk_size': 15000,
                'api_timeout': 30,
                'detailed_logging': True
            },
            'production': {
                'chunk_size': 25000,
                'api_timeout': 60,
                'detailed_logging': False
            }
        }
        self.config = self.configs[environment]
```

2. Automated Deployment

python

```python
# deployment_script.py
class DeploymentManager:
    def setup_for_new_user(self, user_config):
        """Automated setup for new users"""
        # Create directory structure
        # Copy template files
        # Generate user-specific config
        # Set up API credentials template
        pass

    def validate_setup(self, user_directory):
        """Validate that setup is correct"""
        required_files = ['.env', 'credentials.json', 'requirements.txt']
        missing_files = []
        for file in required_files:
            if not os.path.exists(os.path.join(user_directory, file)):
                missing_files.append(file)
        return missing_files
```

Best Practices for Customization

1. Configuration Management

- Keep customizations in separate config files
- Use environment variables for sensitive data
- Document all customizations for team sharing
- Version control your customization files

2. Testing Customizations

- Test with sample data before full deployment
- Validate AI prompts with different conversation styles
- Monitor token usage after customizations
- Compare output quality before and after changes

3. Maintenance

- Regularly review and update prompts based on output quality
- Monitor API costs and optimize as needed
- Update customizations as your role or organization evolves
- Backup your working configurations

This customization guide provides the foundation for adapting the MVP-Leadership Intelligence System to virtually any leadership role, industry, or organizational structure. The key is to start with the base system and incrementally add customizations that match your specific needs and workflow.