# Evaluation Agent User Guide

The Evaluation Agent is an autonomous system that runs comprehensive evaluations of the prompt library.

## Overview

The evaluation agent automatically:

1. Generates evaluation files for all prompt categories
2. Runs evaluations using multiple AI models
3. Analyzes results and calculates pass rates
4. Identifies failing prompts
5. Generates improvement recommendations
6. Creates comprehensive reports

## Quick Start

### Run Full Evaluation

```
# Full autonomous evaluation
python tools/evaluation_agent.py --full

# Full evaluation in dry-run mode (shows what would happen)
python tools/evaluation_agent.py --full --dry-run

# Full evaluation with verbose logging
python tools/evaluation_agent.py --full --verbose
```text
### Run Specific Phase

The evaluation is organized into 4 phases based on category priority:

```bash
# Run phase 1 only (analysis, business)
python tools/evaluation_agent.py --phase 1

# Run phase 2 only (m365, developers)
python tools/evaluation_agent.py --phase 2

# Run phase 3 only (system, advanced)
python tools/evaluation_agent.py --phase 3

# Run phase 4 only (creative, governance)
python tools/evaluation_agent.py --phase 4
```sql
### Resume from Checkpoint

If an evaluation is interrupted, you can resume from the last checkpoint:

```bash
python tools/evaluation_agent.py --resume
```text
### Clear Checkpoint

To start fresh and clear any existing checkpoint:

```bash
python tools/evaluation_agent.py --clear-checkpoint
```yaml
## Prerequisites

Before running the evaluation agent, ensure you have:

1. **Python 3.8+** installed
2. **GitHub CLI (`gh`)** installed: https://cli.github.com/
3. **gh-models extension** installed: `gh extension install github/gh-models`
4. **Authentication** configured: `gh auth login`

The agent will check these prerequisites automatically and provide guidance if anything is missing.

## Configuration

The agent is configured in `tools/evaluation_agent.py` via the `AgentConfig` class:

### Categories

Evaluated categories (in priority order):
- **Phase 1**: analysis, business
- **Phase 2**: m365, developers
- **Phase 3**: system, advanced
- **Phase 4**: creative, governance

### Models

Each category is evaluated with a specific AI model:
- `analysis`: openai/gpt-5-mini
- `business`: xai/grok-3-mini
- `m365`: openai/gpt-5-mini
- `developers`: openai/o4-mini
- `system`: openai/gpt-4.1-mini
- `advanced`: openai/o3-mini
- `creative`: xai/grok-3-mini
- `governance`: openai/gpt-5-mini

### Thresholds

- **Pass Threshold**: 7.0 (out of 10)
- **Target Pass Rate**: 90%
- **Max Parallel Evaluations**: 3

## Output

```
The agent generates several reports:

### Evaluation Report
Location: `docs/reports/EVALUATION_REPORT.md`
- Detailed evaluation results
- Category-by-category breakdown
- Individual prompt scores

### Agent Execution Summary
Location: `docs/reports/AGENT_EXECUTION_SUMMARY.md`
- Overall statistics
- Pass/fail rates by category
- List of failing prompts
- Execution timeline

### Improvement Plan
Location: `docs/reports/IMPROVEMENT_PLAN.md`
- Generated only if prompts fail
- Recommendations for improving failing prompts
- Priority-ordered action items

## Checkpoint System

The agent saves checkpoints to `.eval_checkpoint.json` allowing:
- Resume capability after interruption
- State preservation across runs
- Progress tracking

The checkpoint includes:
- Current phase and category
- Completed categories
- Category results
- Task execution history
- Overall metrics

## Testing

Run the test suites to verify the agent works correctly:

```bash
# Unit tests
python tools/test_evaluation_agent.py

# Integration tests
python tools/test_evaluation_agent_integration.py
```sql
## Troubleshooting

### "GitHub CLI (gh) not found"
Install GitHub CLI from https://cli.github.com/

### "gh-models extension not found"
Run: `gh extension install github/gh-models`
```

```
### "Command failed: authentication required"
Run: `gh auth login` and follow the prompts

### Evaluation times out
- Increase `EVAL_TIMEOUT_SECONDS` in `AgentConfig`
- Run with fewer parallel evaluations
- Run specific phases instead of full pipeline

### Rate limiting errors
- Increase `DELAY_BETWEEN_EVALS_SECONDS`
- Reduce `MAX_PARALLEL_EVALS`

## Advanced Usage

### Dry Run Mode

Test what would happen without actually running evaluations:

```bash
python tools/evaluation_agent.py --full --dry-run
```text
This mode:
- Validates prerequisites
- Shows what would be executed
- Doesn't make API calls
- Doesn't generate reports

### Verbose Mode

Get detailed debug information:

```bash
python tools/evaluation_agent.py --full --verbose
```text
Useful for:
- Debugging issues
- Understanding execution flow
- Monitoring API calls

### Custom Configuration

To customize the evaluation:

1. Edit `AgentConfig` in `tools/evaluation_agent.py`
2. Modify category configurations
3. Adjust thresholds and timeouts
4. Change model assignments

## Architecture

The evaluation agent consists of:

### Core Components
```

```
- **AgentConfig**: Central configuration
- **AgentState**: Persistent state management
- **EvaluationAgent**: Main orchestration class
- **TaskResult**: Task execution tracking
- **CategoryResult**: Per-category results

### Execution Flow

1. **Initialization**: Check prerequisites, load/create state
2. **Phase Execution**: Run categories by priority
3. **Evaluation**: Generate eval files, run model evaluations
4. **Analysis**: Parse results, calculate metrics
5. **Reporting**: Generate comprehensive reports

### Key Features

- **Parallel Execution**: Run multiple evaluations concurrently
- **Retry Logic**: Automatic retries for failed operations
- **Checkpoint Management**: Resume capability
- **Progress Tracking**: Detailed execution history
- **Error Handling**: Graceful failure recovery

## Best Practices

1. **Start with dry-run** to understand what will happen
2. **Run specific phases** during development
3. **Use verbose mode** when troubleshooting
4. **Monitor checkpoints** to track progress
5. **Review reports** after each run

## Support

For issues or questions:
1. Check the troubleshooting section above
2. Review test files for usage examples
3. Examine the source code documentation
4. Open an issue in the repository

## Version History

- **v1.0** (2025-12-03): Initial release
  - Full autonomous evaluation pipeline
  - Multi-phase execution
  - Checkpoint system
  - Comprehensive reporting
```