TALONS
LABORATORIES

User Acceptance Testing (UAT) Document

1. Overview

1.1 Purpose

This UAT document outlines the testing procedures for three Python scripts that form part of a desktop automation system:

- `run_agent_loop.py`: Orchestrates execution of multiple instructions from a txt file
- `run_with_arguments.py`: Executes individual atomic instructions
- `prompt_optimiser.py`: Prompt optimisation module for user instructions

1.2 Testing Objectives

The primary goal of this UAT is to **test the functionality of the scripts themselves**, not the underlying AI model performance.

The scripts should:

- Handle file operations correctly
- Manage session directories properly
- Process command-line arguments appropriately
- Handle errors gracefully
- Execute subprocess calls successfully

Note: Model performance variations are expected and should not be considered test failures.

1.3 Important Note on Model Behavior

CRITICAL: The underlying AI model behavior is **non-deterministic**, meaning:

- Each run will produce different outputs
- Model performance cannot be controlled or predicted
- UAT focuses on script functionality, not model accuracy
- Test success is measured by script execution, not model results

2. Test Environment Setup

2.1 Prerequisites

- Use/activate the python environment created in the VSCode.

2.2 Test Data Preparation

Create test instruction files with various scenarios:

- Single instruction file
- Multiple instruction file (separated by empty lines)
- Empty instruction file
- Invalid file path
- File with special characters

3. Test Cases for `run_agent_loop.py`

3.1 TC-001: Valid Single Instruction File

Objective: Test script with a valid instruction file containing one instruction **Steps**:

1. Create a test instruction file with one instruction
2. Run: `python run_agent_loop.py test_instructions.txt`
3. Verify script executes without errors
4. Check that session directories are created
5. Verify subprocess calls are made correctly

Expected Result: Script completes successfully, session directories created, subprocess executed

3.2 TC-002: Multiple Instructions File

Objective: Test script with multiple instructions separated by empty lines **Steps**:

1. Create instruction file with 3-5 instructions separated by empty lines
2. Run: `python run_agent_loop.py multi_instructions.txt`
3. Verify each instruction is processed sequentially
4. Check unique session IDs are generated for each instruction
5. Verify proper numbering in output

Expected Result: All instructions processed sequentially with unique session IDs

3.3 TC-003: Missing Arguments

Objective: Test script behavior when no arguments provided **Steps**:

1. Run: `python run_agent_loop.py`
2. Verify usage message is displayed
3. Check exit code is 2

Expected Result: Usage message displayed, exit code 2

3.4 TC-004: Non-existent File

Objective: Test script behavior with invalid file path **Steps**:

1. Run: `python run_agent_loop.py nonexistent.txt`

2. Verify error message is displayed
3. Check exit code is 2

Expected Result: Error message displayed, exit code 2

3.5 TC-005: Empty Instruction File

Objective: Test script behavior with empty instruction file **Steps**:

1. Create empty instruction file
2. Run: `python run_agent_loop.py empty.txt`
3. Verify "No instructions found" message
4. Check exit code is 2

Expected Result: Appropriate error message, exit code 2

3.6 TC-006: Interrupt Handling (Ctrl+C)

Objective: Test graceful handling of user interruption **Steps**:

1. Start script with long-running instructions
2. Press Ctrl+C during execution
3. Verify graceful exit message
4. Check exit code is 130

Expected Result: Graceful interruption handling, exit code 130

4. Test Cases for `run_with_arguments.py`

4.1 TC-007: Valid Instruction Execution

Objective: Test script with valid instruction and session ID **Steps**:

1. Run: `python run_with_arguments.py "test instruction" --session-id test123`
2. Verify session directory structure is created
3. Check log file is created and populated
4. Verify DesktopAgent is instantiated correctly

Expected Result: Session directory created, log file populated, agent initialized

4.2 TC-008: Instruction Without Session ID

Objective: Test script with instruction but no session ID **Steps**:

1. Run: `python run_with_arguments.py "test instruction"`
2. Verify flat session directory structure is created
3. Check timestamp-based directory naming
4. Verify log file creation

Expected Result: Flat session structure created with timestamp

4.3 TC-009: Missing Instruction Argument

Objective: Test script behavior when instruction argument is missing **Steps**:

1. Run: `python run_with_arguments.py`
2. Verify argument parser error message
3. Check script exits with error

Expected Result: Argument parser error, script exits

4.4 TC-010: Session Directory Creation

Objective: Test nested session directory structure **Steps**:

1. Run with session ID: `python run_with_arguments.py "test" --session-id abc123`

2. Verify directory structure: `session/session_abc123/session_YYYYMMDD_HHMMSS/`
3. Check log file is in correct location
4. Verify directory permissions

Expected Result: Correct nested directory structure created

4.5 TC-011: Log File Functionality

Objective: Test StreamLogger class functionality **Steps**:

1. Run script with instruction
2. Verify output appears in both console and log file
3. Check log file contains all output
4. Verify log file is properly closed

Expected Result: Dual output to console and file, proper file handling

4.6 TC-012: Interrupt Handling

Objective: Test graceful handling of Ctrl+C **Steps**:

1. Start script with long-running instruction
2. Press Ctrl+C during execution
3. Verify graceful exit message
4. Check log file is properly closed
5. Verify exit code is 130

Expected Result: Graceful interruption, proper cleanup, exit code 130

5. Integration Test Cases

5.1 TC-013: End-to-End Execution

Objective: Test complete workflow from instruction file to execution **Steps**:

1. Create instruction file with multiple instructions
2. Run: `python run_agent_loop.py instructions.txt`
3. Verify each instruction is passed to `run_with_arguments.py`
4. Check session directories are created for each instruction
5. Verify log files contain appropriate output

Expected Result: Complete workflow executes successfully

5.2 TC-014: Error Propagation

Objective: Test error handling between scripts **Steps**:

1. Create instruction that causes `run_with_arguments.py` to fail
2. Run: `python run_agent_loop.py failing_instructions.txt`
3. Verify error is properly propagated
4. Check execution stops at failed instruction
5. Verify appropriate exit codes

Expected Result: Error propagation works correctly, execution stops appropriately

6. Prompt Optimiser Testing

6.1 TC-015: Basic Instruction Transformation

Objective: Test basic instruction transformation into atomic steps **Steps**:

1. Run: `python prompt_optimiser.py "Open Forensic Email Collector. Enter email address test@example.com and click Next."`
2. Verify output has each action on its own line with empty line separators
3. Check exact email address is preserved
4. Verify quotation marks are removed

Expected Result: Instruction broken into atomic steps with proper formatting

6.2 TC-016: Complex Multi-step Instruction

Objective: Test transformation of complex multi-step instructions **Steps**:

1. Run: `python prompt_optimiser.py "Launch the application, navigate to Settings menu and enable dark mode. Then restart the application and verify the theme has changed."`
2. Verify complex instruction is broken into atomic steps
3. Check logical sequence is preserved
4. Verify each step is on its own line

Expected Result: Complex instruction properly decomposed into logical atomic steps

6.3 TC-017: Special Character Handling

Objective: Test handling of special characters and quotation marks **Steps**:

1. Run: `python prompt_optimiser.py "Enter password 'P@$$w0rd!' in the field labeled 'Password' and click 'Sign In'."`
2. Verify quotation marks are removed
3. Check special characters in password are preserved
4. Verify output is clean and readable

Expected Result: Special characters preserved, quotation marks removed, clean output

6.4 TC-019: Error Handling

Objective: Test error handling with invalid inputs **Steps**:

1. Run: `python prompt_optimiser.py ""`
2. Run: `python prompt_optimiser.py --file nonexistent.txt`
3. Verify appropriate error messages are displayed
4. Check non-zero exit codes are returned

Expected Result: Errors handled gracefully with informative messages

6.5 TC-020: File Input Processing

Objective: Test processing instructions from file input **Steps**:

1. Create test file with multi-line instructions
2. Run: `python prompt_optimiser.py --file test_instructions.txt`
3. Verify file content is processed correctly
4. Check output quality matches direct input processing

Expected Result: File input processed with same quality as direct input

6.6 TC-021: Custom Output Path

Objective: Test custom output file path functionality **Steps**:

1. Run: `python prompt_optimiser.py "Test instruction" --output custom_path.txt`
2. Verify output is saved to specified file path
3. Check file exists at custom path with correct content
4. Verify content matches expected format

Expected Result: Output saved to custom path with correct content

7. Power Automate (PA) Specific Testing

7.1 TC-023: PA Environment Performance Comparison

Objective: Test and document performance differences between direct execution and PA execution **Steps**:

1. Execute script directly: `python run_with_arguments.py instructions.txt"`
2. Record execution time and model response quality
3. Execute same POC flow via Power Automate
4. Compare execution times and response quality
5. Document performance degradation metrics

Expected Result: Performance differences documented, PA execution may show reduced performance

Expected Result: Ctrl+C limitations confirmed, alternative methods identified

8. Test Execution Guidelines

8.1 Test Execution Notes

- **Model Behavior**: Note that AI model outputs are non-deterministic
- **Success Criteria**: Focus on script functionality, not model accuracy
- **Error Tolerance**: Some model-related errors are expected and acceptable
- **Logging**: All test outputs should be logged for analysis
- **PA Environment**: When testing via Power Automate, expect potential performance degradation and interrupt handling limitations
- **Execution Context**: Test both direct execution and PA execution to identify environment-specific issues

8.2 Pass/Fail Criteria

PASS: Script executes without Python errors, handles arguments correctly, creates appropriate directories and files **FAIL**: Script crashes with unhandled exceptions, incorrect argument processing, file system errors

9. Test Results Documentation

9.1 Test Results Documentation Guidelines

For each test case, document:

- Test case ID and name
- Tester name and date
- Pass/Fail status
- Any deviations or issues noted

9.2 Defect Reporting Procedures

If test failures occur:

- Document the specific failure
- Note whether it's script-related or model-related
- Provide steps to reproduce

Annex A: UAT Test Results Checklist

Test Case ID	Test Case Name	Date	Tester Name	Pass/Fail	Issues/Deviations/Remarks
TC-001	Valid Single Instruction File				
TC-002	Multiple Instructions File				
TC-003	Missing Arguments				
TC-004	Non-existent File				
TC-005	Empty Instruction File				
TC-006	Interrupt Handling (Ctrl+C)				
TC-007	Valid Instruction Execution				
TC-008	Instruction Without Session ID				
TC-009	Missing Instruction Argument				
TC-010	Session Directory Creation				

Test Case ID	Test Case Name	Date	Tester Name	Pass/Fail	Issues/Deviations/Remarks
TC-011	Log File Functionality				
TC-012	Interrupt Handling				
TC-013	End-to-End Execution				
TC-014	Error Propagation				
TC-015	Basic Instruction Transformation				
TC-016	Complex Multi-step Instruction				
TC-017	Special Character Handling				
TC-018	Stop Cue Recognition				
TC-019	Error Handling				
TC-020	File Input Processing				
TC-021	Custom Output Path				
TC-022	Power Automate Integration				

Test Case ID	Test Case Name	Date	Tester Name	Pass/Fail	Issues/Deviations/Remarks
TC-023	PA Environment Performance Comparison				
TC-024	PA Interrupt Handling Limitations				
TC-025	PA Resource Allocation Testing				