```
from unittest.mock import MagicMock
import unittest
from swarms.structs.agent import Agent
from swarms.tools.tool_parse_exec import parse_and_execute_json

# Mock parse_and_execute_json for testing
parse_and_execute_json = MagicMock()
parse_and_execute_json.return_value = {
    "tool_name": "calculator",
    "args": {"numbers": [2, 2]},
    "output": "4",
}



class TestAgentLogging(unittest.TestCase):
    def setUp(self):
        self.mock_tokenizer = MagicMock()
        self.mock_tokenizer.count_tokens.return_value = 100

        self.mock_short_memory = MagicMock()
        self.mock_short_memory.get_memory_stats.return_value = {
            "message_count": 2
        }

        self.mock_long_memory = MagicMock()
        self.mock_long_memory.get_memory_stats.return_value = {
```

```
            "item_count": 5
        }

        self.agent = Agent(
            tokenizer=self.mock_tokenizer,
            short_memory=self.mock_short_memory,
            long_term_memory=self.mock_long_memory,
        )

    def test_log_step_metadata_basic(self):
        log_result = self.agent.log_step_metadata(
            1, "Test prompt", "Test response"
        )

        self.assertIn("step_id", log_result)
        self.assertIn("timestamp", log_result)
        self.assertIn("tokens", log_result)
        self.assertIn("memory_usage", log_result)

        self.assertEqual(log_result["tokens"]["total"], 200)

    def test_log_step_metadata_no_long_term_memory(self):
        self.agent.long_term_memory = None
        log_result = self.agent.log_step_metadata(
            1, "prompt", "response"
        )
```

```
    self.assertEqual(log_result["memory_usage"]["long_term"], {})

def test_log_step_metadata_timestamp(self):
    log_result = self.agent.log_step_metadata(
        1, "prompt", "response"
    )
    self.assertIn("timestamp", log_result)

def test_token_counting_integration(self):
    self.mock_tokenizer.count_tokens.side_effect = [150, 250]
    log_result = self.agent.log_step_metadata(
        1, "prompt", "response"
    )

    self.assertEqual(log_result["tokens"]["total"], 400)

def test_agent_output_updating(self):
    initial_total_tokens = sum(
        step["tokens"]["total"]
        for step in self.agent.agent_output.steps
    )
    self.agent.log_step_metadata(1, "prompt", "response")

    final_total_tokens = sum(
        step["tokens"]["total"]
        for step in self.agent.agent_output.steps
```

```
        )
        self.assertEqual(
            final_total_tokens - initial_total_tokens, 200
        )
        self.assertEqual(len(self.agent.agent_output.steps), 1)


class TestAgentLoggingIntegration(unittest.TestCase):
    def setUp(self):
        self.agent = Agent(agent_name="test-agent")

    def test_full_logging_cycle(self):
        task = "Test task"
        max_loops = 1

        result = self.agent._run(task, max_loops=max_loops)

        self.assertIsInstance(result, dict)
        self.assertIn("steps", result)
        self.assertIsInstance(result["steps"], list)
        self.assertEqual(len(result["steps"]), max_loops)

        if result["steps"]:
            step = result["steps"][0]
            self.assertIn("step_id", step)
            self.assertIn("timestamp", step)
```

```
            self.assertIn("task", step)
            self.assertIn("response", step)
            self.assertEqual(step["task"], task)
            self.assertEqual(step["response"], "Response for loop 1")

        self.assertTrue(len(self.agent.agent_output.steps) > 0)


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