aunching pytest with arguments
/Users/abhardwaj/Desktop/pythonProject/markitdown/tests/test_markitdown.py --no-
header --no-summary -q in /Users/abhardwaj/Desktop/pythonProject/markitdown/tests

============================= test session starts
==============================
collecting ... collected 5 items

test_markitdown.py::test_markitdown_remote
test_markitdown.py::test_markitdown_local
test_markitdown.py::test_markitdown_exiftool
test_markitdown.py::test_markitdown_deprecation
test_markitdown.py::test_markitdown_llm

============= 2 failed, 1 passed, 2 skipped, 3 warnings in 13.55s ==============
PASSED   [20%]FAILED   [40%]
tests/test_markitdown.py:160 (test_markitdown_local)
'This is a test comment. 12df-321a' != ('\n'
 '\n'
 'AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation\n'
 '\n'
 'Qingyun Wu , Gagan Bansal , Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li '
 'Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Awadallah, Ryen W. '
 'White, Doug Burger, Chi Wang\n'
 '\n'
 '# Abstract\n'
 '\n'
 'AutoGen is an open-source framework that allows developers to build LLM '
 'applications via multiple agents that can converse with each other to '
 'accomplish tasks. AutoGen agents are customizable, conversable, and can '
 'operate in various modes that employ combinations of LLMs, human inputs, and '
 'tools. Using AutoGen, developers can also flexibly define agent interaction '
 'behaviors. Both natural language and computer code can be used to program '
 'flexible conversation patterns for different applications. AutoGen serves as '
 'a generic framework for building diverse applications of various '
 'complexities and LLM capacities. Empirical studies demonstrate the '
 'effectiveness of the framework in many example applications, with domains '
 'ranging from mathematics, coding, question answering, operations research, '
 'online decision-making, entertainment, etc.\n'
 '\n'
 '# Introduction\n'
 '\n'
 'Large language models (LLMs) are becoming a crucial building block in '
 'developing powerful agents that utilize LLMs for reasoning, tool usage, and '
 'adapting to new observations (Yao et al., 2022; Xi et al., 2023; Wang et '
 'al., 2023b) in many real-world tasks. Given the expanding tasks that could '
 'benefit from LLMs and the growing task complexity, an intuitive approach to '
 'scale up the power of agents is to use multiple agents that cooperate. Prior '
 'work suggests that multiple agents can help encourage divergent thinking '
 '(Liang et al., 2023), improve factuality and reasoning (Du et al., 2023), '
 'and provide validation (Wu et al., 2023).\n'
 '\n'
 '## d666f1f7-46cb-42bd-9a39-9a39cf2a509f\n'
 '\n'

```
'In light of the intuition and early evidence of promise, it is intriguing to '
'ask the following question: how can we facilitate the development of LLM '
'applications that could span a broad spectrum of domains and complexities '
'based on the multi-agent approach? Our insight is to use multi-agent '
'conversations to achieve it. There are at least three reasons confirming its '
'general feasibility and utility thanks to recent advances in LLMs: First, '
'because chat optimized LLMs (e.g., GPT-4) show the ability to incorporate '
'feedback, LLM agents can cooperate through conversations with each other or '
'human(s), e.g., a dialog where agents provide and seek reasoning, '
'observations, critiques, and validation. Second, because a single LLM can '
'exhibit a broad range of capabilities (especially when configured with the '
'correct prompt and inference settings), conversations between differently '
'configured agents can help combine these broad LLM capabilities in a modular '
'and complementary manner. Third, LLMs have demonstrated ability to solve '
'complex tasks when the tasks are broken into simpler subtasks. Here is a '
'random UUID in the middle of the paragraph! '
'314b0a30-5b04-470b-b9f7-eed2c2bec74a Multi-agent conversations can enable '
'this partitioning and integration in an intuitive manner. How can we '
'leverage the above insights and support different applications with the '
'common requirement of coordinating multiple agents, potentially backed by '
'LLMs, humans, or tools exhibiting different capacities? We desire a '
'multi-agent conversation framework with generic abstraction and effective '
'implementation that has the flexibility to satisfy different application '
'needs. Achieving this requires addressing two critical questions: (1) How '
'can we design individual agents that are capable, reusable, customizable, '
'and effective in multi-agent collaboration? (2) How can we develop a '
'straightforward, unified interface that can accommodate a wide range of '
'agent conversation patterns? In practice, applications of varying '
'complexities may need distinct sets of agents with specific capabilities, '
'and may require different conversation patterns, such as single- or '
'multi-turn dialogs, different human involvement modes, and static vs. '
'dynamic conversation. Moreover, developers may prefer the flexibility to '
'program agent interactions in natural language or code. Failing to '
'adequately address these two questions would limit the framework's scope of '
'applicability and generality.\n'
'\n'
'Here is a random table for .docx parsing test purposes:\n'
'\n'
'| 1 | 2 | 3 | 4 | 5 | 6 |\n'
'| --- | --- | --- | --- | --- | --- |\n'
'| 7 | 8 | 9 | 10 | 11 | 12 |\n'
'| 13 | 14 | 49e168b7-d2ae-407f-a055-2167576f39a1 | 15 | 16 | 17 |\n'
'| 18 | 19 | 20 | 21 | 22 | 23 |\n'
'| 24 | 25 | 26 | 27 | 28 | 29 |\n'
'\n')
```

<Click to see difference>

```
def test_markitdown_local() -> None:
    markitdown = MarkItDown()

    # Test XLSX processing
    result = markitdown.convert(os.path.join(TEST_FILES_DIR, "test.xlsx"))
    for test_string in XLSX_TEST_STRINGS:
```

```
        text_content = result.text_content.replace("\\", "")
        assert test_string in text_content

    # Test DOCX processing
    result = markitdown.convert(os.path.join(TEST_FILES_DIR, "test.docx"))
    for test_string in DOCX_TEST_STRINGS:
        text_content = result.text_content.replace("\\", "")
        assert test_string in text_content

    # Test DOCX processing, with comments
    result = markitdown.convert(
        os.path.join(TEST_FILES_DIR, "test_with_comment.docx"),
        style_map="comment-reference => ",
    )
    for test_string in DOCX_COMMENT_TEST_STRINGS:
        text_content = result.text_content.replace("\\", "")
>       assert test_string in text_content
E       AssertionError: assert 'This is a test comment. 12df-321a' in '\n\nAutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation\n\nQingyun Wu , Gagan Bansal , Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Awadallah, Ryen W. White, Doug Burger, Chi Wang\n\n# Abstract\n\nAutoGen is an open-source framework that allows developers to build LLM applications via multiple agents that can converse with each other to accomplish tasks. AutoGen agents are customizable, conversable, and can operate in various modes that employ combinations of LLMs, human inputs, and tools. Using AutoGen, developers can also flexibly define agent interaction behaviors. Both natural language and computer code can be used to program flexible conversation patterns for different applications. AutoGen serves as a generic framework for building diverse applications of various complexities and LLM capacities. Empirical studies demonstrate the effectiveness of the framework in many example applications, with domains ranging from mathematics, coding, question answering, operations research, online decision-making, entertainment, etc.\n\n# Introduction\n\nLarge language models (LLMs) are becoming a crucial building block in developing powerful agents that utilize LLMs for reasoning, tool usage, and adapting to new observations (Yao et al., 2022; Xi et al., 2023; Wang et al., 2023b) in many real-world tasks. Given the expanding tasks that could benefit from LLMs and the growing task complexity, an intuitive approach to scale up the power of agents is to use multiple agents that cooperate. Prior work suggests that multiple agents can help encourage divergent thinking (Liang et al., 2023), improve factuality and reasoning (Du et al., 2023), and provide validation (Wu et al., 2023).\n\n## d666f1f7-46cb-42bd-9a39-9a39cf2a509f\n\nIn light of the intuition and early evidence of promise, it is intriguing to ask the following question: how can we facilitate the development of LLM applications that could span a broad spectrum of domains and complexities based on the multi-agent approach? Our insight is to use multi-agent conversations to achieve it. There are at least three reasons confirming its general feasibility and utility thanks to recent advances in LLMs: First, because chat optimized LLMs (e.g., GPT-4) show the ability to incorporate feedback, LLM agents can cooperate through conversations with each other or human(s), e.g., a dialog where agents provide and seek reasoning, observations, critiques, and validation. Second, because a single LLM can exhibit a broad range of capabilities (especially when configured with the correct prompt and inference settings), conversations between differently configured agents can help combine these broad LLM capabilities in a modular and complementary manner. Third, LLMs have demonstrated ability to solve complex tasks when the tasks are broken into simpler subtasks. Here is a random UUID in the middle of the paragraph! 314b0a30-5b04-470b-b9f7-eed2c2bec74a Multi-agent
```

conversations can enable this partitioning and integration in an intuitive manner. How can we leverage the above insights and support different applications with the common requirement of coordinating multiple agents, potentially backed by LLMs, humans, or tools exhibiting different capacities? We desire a multi-agent conversation framework with generic abstraction and effective implementation that has the flexibility to satisfy different application needs. Achieving this requires addressing two critical questions: (1) How can we design individual agents that are capable, reusable, customizable, and effective in multi-agent collaboration? (2) How can we develop a straightforward, unified interface that can accommodate a wide range of agent conversation patterns? In practice, applications of varying complexities may need distinct sets of agents with specific capabilities, and may require different conversation patterns, such as single- or multi-turn dialogs, different human involvement modes, and static vs. dynamic conversation. Moreover, developers may prefer the flexibility to program agent interactions in natural language or code. Failing to adequately address these two questions would limit the framework's scope of applicability and generality.\n\nHere is a random table for .docx parsing test purposes:\n\n| 1 | 2 | 3 | 4 | 5 | 6 |\n| --- | --- | --- | --- | --- | --- |\n| 7 | 8 | 9 | 10 | 11 | 12 |\n| 13 | 14 | 49e168b7-d2ae-407f-a055-2167576f39a1 | 15 | 16 | 17 |\n| 18 | 19 | 20 | 21 | 22 | 23 |\n| 24 | 25 | 26 | 27 | 28 | 29 |\n\n'

test_markitdown.py:183: AssertionError
SKIPPED (do not run if
exiftool is not installed) [60%]
Skipped: do not run if exiftool is not installed
FAILED [80%]
tests/test_markitdown.py:260 (test_markitdown_deprecation)
0 != 1

Expected :1
Actual :0
<Click to see difference>

```
def test_markitdown_deprecation() -> None:
    try:
        with catch_warnings(record=True) as w:
            test_client = object()
            markitdown = MarkItDown(mlm_client=test_client)
>           assert len(w) == 1
E           assert 0 == 1
E            +  where 0 = len([])
```

test_markitdown.py:266: AssertionError
SKIPPED (do not run llm tests
without a key) [100%]
Skipped: do not run llm tests without a key

Process finished with exit code 1