ДОДАТОК А

Програмний код

Лістинг Б.1 – Програмний код AgentWorkflow

```
from llama_index.core.agent.workflow import AgentWorkflow
from configs.llm_config import Settings
from workflows.file_agent import file_agent
from workflows.powershell_agent import powershell_agent
from workflows.agent_orchestrator import agent_orchestrator
from workflows.run_app_agent import run_app_agent
from workflows.web_search_agent import web_search_agent
from workflows.media_control_agent import media_control_agent

from datetime import datetime
import os
info = {
    "User name": os.getenv("USERNAME"),
    "Computer name": os.getenv("COMPUTERNAME"),
    "User domain": os.getenv("USERDOMAIN"),
    "User profile": os.getenv("USERPROFILE"),
    "Home drive": os.getenv("HOMEDRIVE"),
    "Home path": os.getenv("HOMEPATH"),
    "APPDATA": os.getenv("APPDATA"),
    "LOCALAPPDATA": os.getenv("LOCALAPPDATA"),
    "TEMP": os.getenv("TEMP"),
    "SYSTEMROOT": os.getenv("SystemRoot"),
    'System date': datetime.now().date(),
    'System time of workflow execution': datetime.now().time()

}

agent_workflow = AgentWorkflow(
    agents=[agent_orchestrator, file_agent, powershell_agent,
run_app_agent, web_search_agent, media_control_agent],
    root_agent="AgentOrchestrator",
```

```
    initial_state={
        'system_variables': info,
    }
)
```

Лістинг Б.2 – Програмний код AgentOrchestrator

```
from llama_index.core.agent.workflow import ReActAgent

agent_orchestrator = ReActAgent(
    name='AgentOrchestrator',
    description='Used to orchestrate the whole multi-agent
workflow. Can call another agents with Handoff function. DO
NOT CALL TOOLS OF AGENTS! You can only handoff.',
    can_handoff_to=['FileAgent', 'PowershellAgent',
'RunAppAgent', 'WebSearchAgent', 'MediaControlAgent'],
)
```

Лістинг Б.3 – Програмний код FileAgent

```
from llama_index.core.agent.workflow import FunctionAgent
from llama_index.core.workflow import Context

import os


import shutil
import os
import send2trash
import glob

async def copy_file(ctx: Context, origin: str, destination:
str) -> str:
    """Useful for copying file from origin location in PC to
destination location. Your input must contain full paths of
origin and destination of a copy operation. Both must contain
name of file with an extension at the end
```

```
    Example: origin: C:/file.txt, destination: D:/file.txt
    """
    # Copy file from origin to destination
    try:
        shutil.copy2(origin, destination)
        return f'File from {origin} copied to {destination}
successfully.'
    except FileNotFoundError:
        return "Source file not found."
    except PermissionError:
        return "Permission denied."
    except Exception as e:
        return f"An error occurred: {e}"



async def create_file(ctx: Context, file_path:str) -> str:
    """Useful for creating an empty file using the path. Your
input must contain and full path of a future file containing
name of a file with extension. For example, 'C:/Work/test.txt'
    Be sure you've included the full path with file name! """
    # Create file
    try:
        with open(file_path, 'x') as file:
            pass
        return f'File was created in {file_path}
successfully.'
    except FileExistsError:
        return "File already exists."
    except Exception as e:
        return f"An error occurred: {e}"


async def delete_file(ctx: Context, file_path:str) -> str:
    """Useful for deleting a file using the path. Your input
must contain a full path of a file."""
```

```
    # Delete file
    try:
        send2trash.send2trash(file_path.replace('/', '\\'))
        return f'File {file_path} was deleted successfully.'
    except FileNotFoundError:
        return "File not found."
    except PermissionError:
        return "Permission denied."
    except Exception as e:
        return f"An error occurred: {e}"


async def write_file(ctx: Context, content: str,
file_path:str) -> str:
    """Useful for writing to an existing file. Your input must
contain a full path of a file and a content as a string."""
    # Write to file
    try:
        with open(file_path, 'w') as file:
            file.write(content)
        return f'Content was written successfully to
{file_path}'
    except PermissionError:
        return "Permission denied."
    except Exception as e:
        return f"An error occurred: {e}"


async def read_file(ctx: Context, file_path:str) -> str:
    """Useful for reading content of a file using its path.
Your input must contain a full path of a file."""
    # Read file
    try:
        with open(file_path, 'r') as file:
            content = file.read()
            return f'Contents: \n{content}'
```

```
    except FileNotFoundError:
        return "File not found."
    except PermissionError:
        return "Permission denied."


async def dir_path(ctx: Context, path:str) -> str:
    """Useful for checking structure of a file system given
path. Your input must contain a path."""
    # Dir folders
    def print_tree(startpath, max_depth=3, indent="  "):
        content = ''
        for root, dirs, files in os.walk(startpath):
            level = root.replace(startpath, '').count(os.sep)
            if level >= max_depth:
                continue
            indent_str = indent * level
            content +=
f"{indent_str}{os.path.basename(root)}/\n"
            subindent = indent * (level + 1)
            for f in files:
                content += f"{subindent}{f}\n"
        return content
    return f'Structure of a path {path}:\n{print_tree(path)}'

async def search(ctx: Context, full_pattern: str,
recursive=False) -> str:
    """Useful for searching for files using python glob
module. Your input must contain full pattern for search with
glob.glob.
    If recursive search is requested, use the `**/` wildcard
and set `recursive=True`
    Pattern should contain the file name pattern (e.g.,
`*.txt`, `data-??.csv`, `**/*.py`). It should also contain
path where to look for file, like this: C:/Games/**/*.txt.
    Use recursive field whether to search subdirectories
```

```
(`true` or `false`)"""

    files = glob.glob(full_pattern, recursive=recursive)
    return f'Search results: {sorted(files)}' if files else
"No matching files found."

async def rename(ctx: Context, old_path: str, new_path: str) -
> str:
    """Useful for renaming a file. Your input must contain a
full original path and a full new renamed path.
    Example: old_path: C:/file.txt, new_path: C:/new_file.txt
    """
    try:
        os.rename(old_path, new_path)
        return f'File renamed from {old_path} to {new_path}
successfully.'
    except FileNotFoundError:
        return "Source file not found."
    except FileExistsError:
        return "Destination file already exists."
    except PermissionError:
        return "Permission denied."
    except IsADirectoryError:
        return "A directory was specified where a file was
expected."
    except NotADirectoryError:
        return "A component of the path is not a directory."
    except OSError as e:
        return f"OS error occurred: {e}"


file_agent = FunctionAgent(
    name='FileAgent',
    description='Useful for operating with PC\'s file system.
Use it for any file operations like creating and writing
files, reading files, moving them etc.',
```

```
    system_prompt=(
        "You are FileAgent that can perform various actions
with PC's file system."
        "You can create, read, write, copy, delete files and
directories, search for files using glob module, rename files
and directories."
        "You can also check structure of a file system given
path."
        "You must perform any file operation that is requested
by the AgentOrchestrator."
        "Given a request, you should fulfil it and provide
concise response as a report of your work."
        "If the system asks you to do a task, which requires
additional information, do NOT make it up! For example, if you
need to check some system parameters, handoff to
AgentOrchestrator for extra info."
        "After you have completed your actions, handoff to
AgentOrchestrator!"
        "Always return back to AgentOrchestrator. Be sure you
finish with a handoff."
    ),
    tools=[copy_file, create_file, write_file, delete_file,
read_file, dir_path, search, rename],
    can_handoff_to=['AgentOrchestrator']
)
```

Лістинг Б.4 – Програмний код MediaControlAgent

```
 import keyboard
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from llama_index.core.agent.workflow import FunctionAgent

devices = AudioUtilities.GetSpeakers()
interface = devices.Activate(IAudioEndpointVolume._iid_,
```

```
CLSCTX_ALL, None)
volume = cast(interface, POINTER(IAudioEndpointVolume))



async def play() -> str:
    """Useful for sending start playback signal to system."""
    keyboard.send("play/pause media")
    return 'System received play command.'

async def pause() -> str:
    """Useful for sending playback pause signal to system."""
    keyboard.send("play/pause media")
    return 'System received pause command.'

async def next_track() -> str:
    """Useful for sending next track signal to system."""
    keyboard.send("next track")
    return 'System received next track command.'

async def previous_track() -> str:
    """Useful for sending previous track signal to system."""
    keyboard.send("previous track")
    return 'System received previous track command.'
async def set_volume_level(volume_level: int) -> str:
    """Useful for setting system volume level. Your input must
contain int value of volume level from 0 to 100."""
    volume.SetMasterVolumeLevelScalar(volume_level/100, None)
    return f'Volume level set to {volume_level}.'
async def get_volume_level() -> str:
    """Useful for getting system volume level."""
    current = volume.GetMasterVolumeLevelScalar()
    return f"Current Volume: {current*100}%."

media_control_agent = FunctionAgent(
    name='MediaControlAgent',
```

```
    description='Useful for controlling media in Windows. Can
switch tracks, pause and unpause them and set and get system
volume level.',
    system_prompt=(
        "You are MediaControlAgent that can control media in
Windows 11."
        "Given a request, you should fulfil it and provide
concise response as a report of your work."
        "If the system asks you to do a task, which requires
additional information, do NOT make it up! For example, if you
need to check some system parameters, handoff to
AgentOrchestrator for extra info."
        "After you have completed your actions, handoff to
AgentOrchestrator!"
        "Always return back to AgentOrchestrator. Be sure you
finish with a handoff."
    ),
    tools=[play, pause, next_track, previous_track,
set_volume_level, get_volume_level],
    can_handoff_to=['AgentOrchestrator']
)
```

Лістинг Б.5 – Програмний код PowershellAgent

```
 from llama_index.core.agent.workflow import FunctionAgent
from llama_index.core.workflow import Context
from tools.command_executor import execute_cmd_command
from workflows.powershell_command_generator import
CommandGenerator

command_generator = CommandGenerator()

async def generate_command(ctx: Context, query: str):
    """Create a list of Powershell commands to fulfil the
query. Your input should contain all information needed to
generate a Powershell command using LLM."""
```

```
    generated_commands =
command_generator.generate(query=query)
    return f'Generated Powershell command:
{str(generated_commands)}'

async def execute_command(ctx: Context, command: str):
    """Useful for executing generated Powershell command in
Windows 11 PC. Your input should contain a SINGLE properly
syntaxed command."""
    return execute_cmd_command(command)

powershell_agent = FunctionAgent(
    name='PowershellAgent',
    description='Useful for creating and executing various
Powershell commands in Windows 11. Can execute API calls, open
Websites and other. It is very powerfull and can execute
almost everything.',
    system_prompt=(
        "You are PowershellAgent that can create and execute
Windows 11 Powershell Commands"
        "Given a request, you should fulfil it and provide
concise response as a report of your work."
        "After you have completed your actions, handoff to
AgentOrchestrator!"
        "Always return back to AgentOrchestrator. Be sure you
finish with a handoff."
        "Always give a proper reason for a handoff.
AgentOrchestrator must know exactly why."
    ),
    tools=[generate_command, execute_command],
    can_handoff_to=['AgentOrchestrator']
)
```

Лістинг Б.6 – Програмний код RunAppAgent

```
from llama_index.core.agent.workflow import FunctionAgent
from llama_index.core.workflow import Context
import pandas as pd
import subprocess

df = pd.read_csv("workflows/InstalledApps.csv")

def get_app_path(app_name):
    row = df[df['Name'] == app_name]
    if not row.empty:
        return row.iloc[0]['Path']

    partial = df[df['Name'].str.contains(app_name, case=False,
na=False)]
    if not partial.empty:
        return partial.iloc[0]['Path']
    return None

async def run_app(ctx: Context, app_name: str):
    """Useful for running applications in Windows 11 PC. Your
input should contain a name of an app."""

    app_path = get_app_path(app_name)
    if app_path:
        subprocess.Popen([app_path])
    else:
        return f"App {app_name} not found."
    return f'App {app_name} was successfully launched.'

run_app_agent = FunctionAgent(
    name='RunAppAgent',
    description='Useful for running installed apps in Windows
like browsers, text editors, games etc. CANNOT RUN SCRIPTS!',
    system_prompt=(
```

```
        "You are RunAppAgent that can run different installed
apps in Windows."
        "Given a request, you should fulfil it and provide
concise response as a report of your work."
        "After you have completed your actions, handoff to
AgentOrchestrator!"
        "Always return back to AgentOrchestrator. Be sure you
finish with a handoff."
        "You can't run scripts directly. Handoff back to
orchestrator and give this as a reason. Advice it to use
PowershellAgent."
    ),
    tools=[run_app],
    can_handoff_to=['AgentOrchestrator']
)
```

Лістинг Б.7 – Програмний код WebSearchAgent

```
 from llama_index.core.agent.workflow import FunctionAgent
from llama_index.core.workflow import Context
from tavily import AsyncTavilyClient
import os

async def search_web(query: str) -> str:
    """Useful for using the web to answer questions. Provide
concise query."""
    client =
AsyncTavilyClient(api_key=os.getenv('TAVILY_API_KEY'))
    return str(await client.search(query))

web_search_agent = FunctionAgent(
    name='WebSearchAgent',
    description='Useful for searching any information in Web.
Use it whenever you need.',
    system_prompt=(
        "You are WebSearchAgent that can perform search of
```

```
information in WEB."
        "Given a request, you should fulfil it and provide
concise response as a report of your work."
        "After you have completed your actions, handoff to
AgentOrchestrator!"
        "Always return back to AgentOrchestrator. Be sure you
finish with a handoff."

    ),
    tools=[search_web],
    can_handoff_to=['AgentOrchestrator']
)
```

ДОДАТОК Б

Запити до LLM

Лістинг В.1 – Запит до GPT-4o для генерації few-shot прикладів

```
You must generate pairs query - steps, that are required to perform a task as if you were an PC-Assistant multiagent-system that fulfils user requests.
You are an orchestrating agent, that delegates tasks to another functional agents.

Available agents:
**FileAgent** – Useful for operating with PC's file system. Handles file creation, reading, writing, copying, deletion, directory inspection, renaming, and file search.

**Tools:**

- `copy_file(origin: str, destination: str)` – Copies a file from origin to destination. Full file paths required.

- `create_file(file_path: str)` – Creates an empty file at the specified path. Requires full path including filename and extension.

- `write_file(content: str, file_path: str)` – Writes string content to a file at the specified path.

- `delete_file(file_path: str)` – Deletes a file at the specified full path.

- `read_file(file_path: str)` – Reads and returns the content of a file.

- `dir_path(path: str)` – Returns a tree view of the directory structure up to 3 levels deep for the given path.
```

- `search(full_pattern: str, recursive: bool)` - Searches for files using glob pattern matching. Supports recursive search.

- `rename(old_path: str, new_path: str)` - Renames or moves a file from old_path to new_path.

MediaControlAgent - Useful for controlling media playback and system volume in Windows. Can play/pause media, switch tracks, and set or get the system volume level.

Tools:

- `play()` - Sends a system-level command to start/resume media playback.

- `pause()` - Sends a system-level command to pause media playback.

- `next_track()` - Skips to the next media track.

- `previous_track()` - Goes back to the previous media track.

- `set_volume_level(volume_level: int)` - Sets system volume to a specified level (0-100).

- `get_volume_level()` - Retrieves and reports the current system volume level.

PowershellAgent - Useful for creating and executing various PowerShell commands in Windows 11. Can handle tasks like API calls, opening websites, and more. Very powerful for system-level automation.

Tools:

- `generate_command(query: str)` – Uses an LLM-based system to generate one or more PowerShell commands based on the provided natural language query.

- `execute_command(command: str)` – Executes a single, properly formatted PowerShell command on the system.

RunAppAgent – Useful for launching installed applications in Windows (e.g., browsers, editors, games). **Cannot run scripts directly.**

Tools:

- `run_app(app_name: str)` – Launches an installed application by its name. Matches exact or partial name from a predefined CSV of installed apps. Returns an error if not found.

WebSearchAgent – Useful for searching any information on the web. Use it whenever external up-to-date knowledge is required.

Tools:

- `search_web(query: str)` – Performs a web search using the Tavily API with a concise query. Returns search results as a string.

Example:
Query: Open the URL https://example.com in the default browser
Steps:
1) PowershellAgent -> generate_command('Open https://example.com in browser')
2) PowershellAgent -> execute_command('<generated command>')

You must create 30 pairs. Create such queries that require at least three agentic calls.

ДОДАТОК В

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	84 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Крижній Д.Д.			Інтелектуальний асистент для керування операційною системою на основі мультиагентної технології та великих мовних моделей	Шифр групи	Код напр./спец.
Перевірив		Шевченко О.Ю.				ІТШІ-21-5	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Золотухін О.В.					