МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

УТВЕРЖДАЮ
Директор Ассоциации
«Искусственный интеллект в промышленности»

_____ Т. М.Супатаев
_____2024

УТВЕРЖДАЮ
Научный руководитель ИЦ СИИП
Университета ИТМО

_____ А. В. Бухановский
_____2024

ИНСТРУМЕНТАРИЙ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ КОГНИТИВНЫХ КАРТ

ТЕКСТ ПРОГРАММЫ

ЛИСТ УТВЕРЖДЕНИЯ

RU.СНАБ.00853-02 12 ББ-ЛУ

Представители
Организации-разработчика

Руководитель разработки

_____ Я.С. Коровин
_____ 2024

Нормоконтролер

_____ Е. В. Игнатова
_____2024

2024

Подп. и дата	
Инв. № дубл.	
Взам. Инв №	
Подп. и дата	
Инв. № подл.	

УТВЕРЖДЕН

RU.СНАБ.00853-02 12 ББ-ЛУ

ИНСТРУМЕНТАРИЙ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ КОГНИТИВНЫХ КАРТ

ТЕКСТ ПРОГРАММЫ

RU.СНАБ.00853-02 12 ББ

ЛИСТОВ 8

Подп. и дата	
Инв. № дубл.	
Взам. Инв №	
Подп. и дата	
Инв. № подл.	

2024

АННОТАЦИЯ

Документ содержит исходный код компонента “Инструментарий параллельной обработки когнитивных карт”. Этот компонент предназначен для применения совместно с компонентом адаптивной оптимизации выполнения производственных процессов с использованием вероятностных моделей и динамически изменяемой среды. Компонент входит в состав ПО, разрабатываемого в рамках мероприятия М1 плана Исследовательского центра в сфере искусственного интеллекта «Сильный ИИ в промышленности» (ИЦ ИИ) в рамках соглашения с АНО «Аналитический центр при Правительстве Российской Федерации» (ИГК 000000D730321P5Q0002), № 70-2021-00141.

Компонент предназначен для организации параллельной работы компонента адаптивной оптимизации выполнения производственных процессов с использованием вероятностных моделей и динамически изменяемой среды.

1 ОБЩИЕ СВЕДЕНИЯ

Текст программы размещен в репозитории https://gitlab.actcognitive.org/itmo-sai-code/cogmapoptimizer.

2 ИСХОДНЫЙ КОД

Файл deploy.py.

```
import os
import shutil
import json
import paramiko
import time


def zip_extract(file_name_, extract_dir=""):
    shutil.unpack_archive(file_name_, extract_dir, "zip")


def execute_remote_command(host_, username_, password_, command_):
    client = None
    try:
        client = paramiko.SSHClient()
        client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        client.connect(hostname=host_, username=username_, password=password_)
        _, _, _ = client.exec_command(command_)
        client.close()
        return
    finally:
        client.close()


"""
@brief Копирует файл из src в dst (поверх), поддерживает локальные пути и расша-
ренные папки по сети.
@param src Исходный файл (путь + имя файла).
@param dst Путь к папке назначения (только путь).
"""
def copy_file(src, dst):
    try:
        if not os.path.exists(src):
            print(f"Файл {src} не существует.")
            return
        if not os.path.exists(dst):
            os.makedirs(dst)
        if os.path.exists(f"{dst}{os.path.basename(src)}"):
            os.remove(f"{dst}{os.path.basename(src)}")
        shutil.copy(src, dst)
        return 0
    except (FileNotFoundError, PermissionError, OSError) as e:
        print(f"Error: {e}")
        return -1


def get_host_name(net_path):
    parts = net_path.lstrip("\\").split("\\")
    host_name = parts[0] if parts else None
    return host_name


def change_figure(command, new_figure):
    parts = command.split()
    for i in range(len(parts) - 1):
```

```
            if parts[i + 1] == "-remote":
                parts[i] = str(new_figure)
                break
        return ' '.join(parts)


def get_new_net_path_by_exe(exe_string, nodes_):
    exe_string = exe_string.split("&&")[0].strip()
    exe_string = exe_string.split("cd ")[1].strip()
    for item in nodes_:
        if item["local_path"] == exe_string:
            return item["net_path"]
    return None


def get_data_from_config_by_host(host_, nodes_):
    for item in nodes_:
        index = item["net_path"].find(host_)
        if index != -1:
            if 0 <= index <= 2:
                return item["user"], item["pass"]
    return None, None


DEPLOY_FILES_LIST = ["cogmap.py", "optimizer.py", "report.py",
"impact_generator.py",
                     "proba.py", "zip.py", "uid.py", "__main__.py", "model.h5"]


if __name__ == "__main__":
    start_time = time.time()
    print(f"start_time = {start_time}")

    # Чтение конфига
    with open("deploy.json", "r", encoding="cp1251") as file:
        j_data = json.load(file)

    # Копирование исполняемых файлов и файлов когнитивной карты
    local_storage_folder = j_data["local_storage_folder"]
    cog_map = j_data["cog_map"]
    cog_map_xyz = j_data["cog_map_xyz"]
    pulse_model_steps = j_data["pulse_model_steps"]
    nodes = j_data["nodes"]
    exe_files = []
    print(f"Files deployment...")
    figure = 0  # Устойчивая фигура для обработки
    for node in nodes:
        for file in DEPLOY_FILES_LIST:
            print(f"Deploy {file} to {node['net_path']}{file} ...")
            copy_file(file, node['net_path'])
        print(f"Deploy {cog_map} to {node['net_path']}{cog_map} ...")
        copy_file(cog_map, node['net_path'])
        print(f"Deploy {cog_map_xyz} to {node['net_path']}{cog_map_xyz} ...")
        copy_file(cog_map_xyz, node['net_path'])
        # Добавить скрипт для выполнения
        exe_files.append(f"cd {node['local_path']}&&python __main__.py {cog_map}
{cog_map_xyz} "
                         f"{pulse_model_steps} {figure} -remote  > NUL 2>&1 &")
        figure += 1
    print(f"Deployment completed\n")
```

```
    # Запуск
    max_running = len(nodes)  # Число запускаемых скриптов = число указанных в
конфиге папок
    running_set = {}  # Словарь для отслеживания запущенных процессов
    exe_queue = exe_files[:]  # Очередь файлов для запуска
    path_queue = [item['net_path'] for item in nodes]  # Очередь сетевых путей к
файлам для запуска
    done = []

    print("Starting remote tasking...")
    while exe_queue or running_set or figure <= 6:
        # Запуск новых процессов, если это возможно
        while (len(running_set) < max_running and exe_queue) or done and figure
<= 6:
            if done:
                # Добавить новый скрипт для запуска
                new_exe = change_figure(done.pop(0), figure)
                exe_queue.append(new_exe)
                figure += 1
                new_path = get_new_net_path_by_exe(new_exe, nodes)
                path_queue.append(new_path)
            exe_file = exe_queue.pop(0)
            path = path_queue.pop(0)
            txt_file = f"{path}done.txt"
            host = get_host_name(txt_file)
            print(f"Host: {host} - Starting {exe_file}")
            username, password = get_data_from_config_by_host(host, nodes)
            execute_remote_command(host, username, password, exe_file)
            running_set[exe_file] = {"script": exe_file, "txt_file": txt_file}

        # Проверка завершенных процессов
        completed_exes = []
        for exe_file, data in running_set.items():
            if os.path.exists(data["txt_file"]):  # Проверяем наличие txt-файла
                print(f"Completed {exe_file}")
                completed_exes.append(exe_file)
                time.sleep(1)
                # Копируем архив назад
                with open(data["txt_file"], "r", encoding="cp1251") as file:
                    arch_name = file.read()
                file_name = data["txt_file"]
                file_name =
file_name.replace(os.path.basename(data["txt_file"]), arch_name, 1)
                if file_name.endswith("\n"):
                    file_name = file_name[:-1]
                copy_file(file_name, local_storage_folder)
                os.remove(data["txt_file"])  # Удаляем done.txt
                os.remove(file_name)  # Удаляем архив с результатами
                done.append(exe_file)

        # Удаляем завершённые процессы из списка
        for exe_file in completed_exes:
            del running_set[exe_file]

        # Задержка перед следующей проверкой
        time.sleep(1)
    print("All tasks are finished\n")

    print("Unpacking results...")
    # Распаковка архивов с результатами
    archives = [arch for arch in os.listdir(local_storage_folder) if
```

```
arch.endswith(".zip")]
    for arch in archives:
        print(f"Unpacking {arch}")
        zip_extract(f"{local_storage_folder}{arch}", f"{local_storage_folder}")
    print("Results unpacked\n")

    print(f"All done in {(time.time() - start_time):.1f} second(s)")
    print(f"stop_time = {time.time()}")
```

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

	Номера листов (страниц)				*Всего листов (страниц) в докум.*	*№ документа*	*Входящий номер доп. Документа и дата*	*Подп.*	*Дата*
Изм.	*изменен-ных*	*заменен-ных*	*новых*	*аннулиро-ванных*					