https://orcid.org/0000-0001-6137-0599
https://orcid.org/0000-0002-5523-6120

УДК 004.94

С.В. ГОЛУБ[*], В.В. ОСТАПЮК[*]

DAG-ОРІЄНТОВАНЕ ПОДАННЯ ПРОЦЕСІВ КОНСТРУЮВАННЯ АЛГОРИТМІВ СИНТЕЗУ МОДЕЛЕЙ МОНІТОРИНГОВИМИ ПРОГРАМНИМИ АГЕНТАМИ

[*]Черкаський державний технологічний університет, м. Черкаси, Україна

Анотація. *У статті розглядається фундаментальна задача надійного та відтворюваного конструювання багатошарових алгоритмів синтезу моделей (АСМ), призначених для використання в моніторингових програмних агентах у складі мультиагентних систем (МАС). Після успішного навчання багатошарових модельних ансамблів та інших складних АСМ, що демонструють високу точність у дослідницькому середовищі, виникає виклик, пов'язаний з їхнім портативним та детермінованим розгортанням. Для вирішення цієї задачі запропоновано використати DAG-орієнтований підхід, де вся логіка конструювання АСМ подається у формі єдиного, самодостатнього та легко інтерпретованого спрямованого ациклічного графа (DAG). Розроблено набір типізованих вузлів графа, які дозволяють прозоро представляти такі складні операції, як-от підвищення однорідності даних та багатошарова рециркуляція, розбиваючи їх на логічні, зрозумілі кроки. Досліджено процес перетворення багатошарового навчання ансамблів моделей на портативний DAG, що містить всі залежності між компонентами, шляхи до артефактів та параметри агрегації. Продемонстровано, що запропонований підхід, завдяки чітко визначеним правилам обміну даними між вузлами та фіксованому порядку виконання, що гарантується топологічним сортуванням, забезпечує повну відповідність результатів виконання тим, що були отримані під час навчання. Встановлено, що такий підхід створює надійний міст між середовищем дослідження та практичним застосуванням. Він забезпечує портативність, прозорість (завдяки інструментам трасування) та верифікованість (через можливість оцінки проміжних вузлів) складних методів конструювання АСМ. У висновку, розроблений підхід є важливим для процесів конструювання АСМ програмних агентів, здатних стабільно та передбачувано функціонувати в реальних умовах.*

Ключові слова: *моніторинг, програмний агент, моделі, спрямований ациклічний граф (DAG), відтворюваність, декомпозиція, алгоритми синтезу моделей, ансамблі моделей.*

Abstract. *The article considers a fundamental task of ensuring reliable and reproducible construction of multilayer model synthesis algorithms (MSA) intended for use in monitoring software agents as part of multi-agent systems (MAS). After successful training of multilayer model ensembles and other complex MSAs that demonstrate high accuracy in a research environment, a challenge arises related to their portable and deterministic deployment. To solve this problem, a DAG-oriented architecture is proposed, where the entire logic of the MSA execution is represented as a single, self-sufficient, and easily interpreted directed acyclic graph (DAG). A set of typed graph nodes has been developed that allows for transparent representation of complex operations such as increasing data homogeneity and multilayer recirculation, breaking them into logical and understandable steps. The process of transforming the trained architecture into a portable DAG containing all dependencies between components, paths to artifacts, and aggregation parameters is investigated. It is demonstrated that the proposed approach, due to clearly defined rules of data exchange between nodes and a fixed execution order guaranteed by topological sorting, ensures full compliance of the execution results with those obtained during training. It is established that such an architecture creates a reliable bridge between the research environment and practical application. It provides portability, transparency (thanks to tracing tools), and verifiability (due to the possibility of evaluating intermediate nodes) of complex methods of MSA construction. In conclusion, the developed approach is important for building reliable software agents capable of stable and predictable functioning in real-world conditions.*

ISSN 1028-9763. Математичні машини і системи. 2026. № 1

Keywords: *monitoring, software agent, models, directed acyclic graph (DAG), reproducibility, decomposition, model synthesis algorithms, model ensembles.*

DOI: 10.34121/1028-9763-2026-1-73-86

1. Вступ

Сучасні системи інтелектуального моніторингу все частіше реалізують на основі архітектури мультиагентних систем (МАС), де моніторингові програмні агенти виступають ключовими елементами, що забезпечують аналіз даних та підтримку прийняття рішень у складних динамічних середовищах [1–3]. Якість роботи таких агентів безпосередньо залежить від ефективності їхніх внутрішніх методів конструювання АСМ. У попередніх дослідженнях було показано, що для досягнення високої точності при роботі з різнорідними даними є доцільним використання складних багатокомпонентних архітектур. Зокрема, свою ефективність продемонстрували багатошарові ансамблі, побудовані за методом рециркуляції, що був вперше запропонований у [4] та удосконалений у [7]. Аналогічно, позитивні результати показали методи, спрямовані на підвищення однорідності даних шляхом їх попередньої кластеризації.

Але успішне створення та валідація таких складних методів конструювання АСМ на етапі дослідження є лише першою частиною завдання. Наступною, не менш важливою, є задача їхнього надійного та відтворюваного виконання (inference) у практичних умовах. Проте ефективність традиційних підходів до розгортання значно знижується зі зростанням складності методу конструювання АСМ. Сучасні архітектури часто містять не один, а багато взаємопов'язаних компонентів, численні кроки передобробки даних та специфічні правила об'єднання проміжних результатів. Проста серіалізація однієї фінальної моделі, що є поширеною практикою, не здатна охопити всю цю багатоетапну логіку [5]. Це призводить до значних труднощів при перенесенні моделі в інше середовище, помилок у налаштуваннях та, як наслідок, до непередбачуваних розбіжностей у результатах. Така ситуація не лише підриває довіру до розробленої моделі, але й створює значний розрив між теоретичними досягненнями дослідження та його реальним практичним застосуванням.

Для вирішення цієї задачі у даній роботі запропоновано DAG-орієнтований підхід. Її основний принцип — представити весь процес виконання не як набір окремих моделей, а як єдину, цілісну систему. Для її опису використовується спрямований ациклічний граф (DAG), у якому кожен крок обчислень — від початкової обробки даних до отримання прогнозу — представлений як окремий вузол. Кожен вузол має чітко визначені входи та виходи, він описує весь потік даних через АСМ, що створює надійний, верифікований та портативний механізм, здатний використовувати навіть найскладніші архітектури ансамблів. Отже, розроблений підхід стає ключовим для побудови надійних програмних агентів, здатних стабільно та передбачувано функціонувати в реальних умовах.

Мета статті — розробити та обґрунтувати метод представлення процесів конструювання складних алгоритмів синтезу моделей (АСМ) у вигляді спрямованого ациклічного графа (DAG), що є необхідною умовою для забезпечення надійного функціонування моніторингових програмних агентів у складі мультиагентних систем при роботі з потоками різнорідних даних.

2. Методи та засоби для представлення та виконання обчислювальних графів

Підхід, який пропонується у статті, заснований на концепції спрямованого ациклічного графа (DAG) для представлення та виконання складних обчислювальних послідовностей. Щоб зрозуміти переваги та історію цього підходу, корисно розглянути, як DAG використовуються в сучасних системах обробки даних. Також варто розглянути сучасні підходи до серіалізації моделей та труднощі, які вони створюють.

2.1. Концепція спрямованих ациклічних графів (DAG) в обробці даних

Спрямований ациклічний граф, фундаментальна структура даних у комп'ютерних науках, є одним із найкращих способів представлення процесів з односпрямованими залежностями та операціями, що виконуються в певному порядку. Операція або крок обробки представлений набором вершин (вузлів) у такому графі, тоді як потік даних або керування між операціями представлений спрямованими ребрами (стрілками). Основною характеристикою є ациклічність, яка вказує на неможливість повернутися до вузла, який вже був відвіданий, рухаючись вздовж стрілок. Ця особливість гарантує, що процес має чіткий початок і кінець, позбавлений нескінченних циклів, і дозволяє дотримуватися суворого набору кроків у кожному виконанні.

У галузі обробки даних та машинного навчання DAG стали де-факто стандартом для опису складних послідовностей обробки даних (data pipelines). Їхня популярність зумовлена здатністю декларативно описувати складні процеси, що складаються з багатьох взаємозалежних кроків. Провідні інструменти для розподілених обчислень, як-от Apache Spark [11] та Dask [9], активно використовують DAG «під капотом». У них кожна трансформація даних — чи то фільтрація таблиці, чи агрегація, чи об'єднання — представляється як вузол графа. Система генерує повний DAG перед виконанням набору дій, визначених розробником. Це дозволяє механізму виконання ефективно налаштувати розподілене виконання, дослідити весь процес і визначити області, які потребують оптимізації (наприклад, об'єднання кількох операцій в одну або одночасне виконання різних гілок).

В області автоматизованого управління робочих процесів (workflow orchestration) системи, як-от Apache Airflow [6], виводять концепцію DAG на рівень користувача. Розробник явно описує DAG, де кожен вузол — це окреме завдання (наприклад, «завантажити дані з бази», «навчити модель», «зберегти результат»). Ребра, що з'єднують вузли, визначають залежності; завдання B не розпочнеться, доки не буде завершено завдання A. Помилки та повторні спроби обробляються самою системою управління, яка також гарантує, що завдання виконуються у правильному порядку.

Отже, використання DAG для опису послідовності дій є перевіреним та надійним методом, що забезпечує прозорість, керованість та оптимізацію складних процесів. Саме тому цей метод був обраний як основа для подання складних методів конструювання АСМ у нашому дослідженні. Він дозволяє представити багатокомпонентний ансамбль не як набір розрізнених моделей, а як єдиний, цілісний та верифікований граф обчислень [13].

2.2. Формати серіалізації моделей та обмеження їх застосування

Щоб використовувати навчену модель машинного навчання в майбутньому, її необхідно зберегти. Цей процес, відомий як серіалізація, перетворює об'єкт моделі з усіма її параметрами у формат, який можна записати на диск. Згодом таку модель можна відновити у пам'яті без необхідності повторного навчання. Існують декілька поширених підходів до серіалізації, однак їхні можливості стають обмеженими при переході від роботи з окремими моделями до складних багатокомпонентних методів конструювання АСМ.

Найбільш прямим та поширеним методом у середовищі Python є використання бібліотек, як-от pickle або його більш ефективний для великих масивів NumPy аналог joblib. Ці інструменти дозволяють надійно серіалізувати практично будь-який об'єкт Python, включаючи моделі, навчені за допомогою *scikit-learn* та подібних. Саме цей механізм є ефективним для збереження окремих компонентів майбутньої архітектури. Для забезпечення відтворюваності інформація про версії бібліотек, що використовувалися, зазвичай зберігається окремо (наприклад, у файлі *provenance.json*, як це реалізовано в нашому підході), що дозволяє відновити необхідне програмне оточення [1].

Однак навіть за умови вирішення задачі сумісності версій, фундаментальний виклик полягає в тому, що серіалізація окремих моделей не вирішує задачу відтворення всього процесу конструювання АСМ. Якщо архітектура складається з багатьох взаємопов'язаних компонентів, простого набору збережених файлів недостатньо. Відсутня формалізована інформація про те, в якому порядку ці моделі мають виконуватися, як поєднуються їхні результати та які дані подаються на вхід кожній із них. Ця логіка оркестрації залишається поза межами самих серіалізованих файлів.

Для вирішення задачі сумісності та портативності моделей між різними програмними середовищами було розроблено стандартизовані формати. Одним із найвідоміших є ONNX (Open Neural Network Exchange) [8]. Його головна мета — забезпечити інтероперабельність, дозволяючи експортувати моделі, навчені в одному фреймворку (наприклад, PyTorch), і виконувати їх в іншому (наприклад, TensorFlow або на спеціалізованому апаратному забезпеченні). ONNX описує обчислювальний граф самої моделі на рівні низькорівневих математичних операторів, що робить її надзвичайно портативною. Іншим, схожим за ідеєю стандартом, є PMML (Predictive Model Markup Language) [2], який використовує XML для опису моделей, забезпечуючи їхню незалежність від мови програмування та платформи виконання.

Незважаючи на їхню потужність, ці стандартизовані формати мають спільне обмеження в контексті нашого дослідження: вони зосереджені на описі однієї монолітної моделі. Вони чудово справляються із завданням представити архітектуру нейронної мережі або параметри логістичної регресії. Проте вони не надають вбудованих засобів для опису всього багатоетапного процесу виконання, що є характерним для складних ансамблів. Наприклад, стандартний ONNX не може описати логіку, де спочатку виконується кластеризація, потім на основі її результату обирається одна з кількох спеціалізованих моделей, а наприкінці їхні прогнози усереднюються.

Отже, виклик полягає не стільки в серіалізації окремих моделей, скільки у відсутності «головного креслення» (master blueprint), яке б декларативно описувало, як усі ці окремі серіалізовані компоненти мають взаємодіяти між собою. Це підкреслює необхідність у підході вищого рівня, як-от DAG, який би описував не окремі моделі, а весь процес та логіку взаємодії між ними, забезпечуючи цілісність та відтворюваність усього процесу виконання.

2.3. Узагальнення задачі та вимоги до процесу

Аналіз існуючих підходів до опису та виконання методів конструювання АСМ дозволяє сформулювати ключові виклики, що стоять на шляху надійного розгортання складних багатокомпонентних систем машинного навчання.

По-перше, існує задача повноти опису. Традиційні формати, як-от *joblib* або *ONNX*, ефективно зберігають стан окремих моделей, але не містять інформації про те, як ці моделі мають взаємодіяти між собою. Складна логіка, що містить послідовність кроків передобробки, розгалуження потоків даних та правила агрегації результатів, залишається неформалізованою, зазвичай у вигляді імперативного коду, що використовувався для навчання. Це робить процес розгортання нетривіальним і схильним до помилок. Тому ефективний підхід до виконання повинен дозволяти декларативно описати послідовність обчислень та залежності між усіма компонентами в єдиному самодостатньому форматі. Такий формальний опис, що є єдиним «джерелом істини», також має забезпечувати портативність, тобто можливість легкого перенесення всього процесу конструювання АСМ (наприклад, із дослідницького ноутбука на сервер) без необхідності модифікації коду.

По-друге, виникає задача верифікації та прозорості. Коли складний метод конструювання АСМ працює як «чорна скринька», важко діагностувати проблеми, аналізувати її поведінку або верифікувати коректність роботи окремих компонентів. Відсутність стандарти-

зованого доступу до проміжних результатів (наприклад, прогнозів окремих моделей в ансамблі) значно ускладнює налагодження та розуміння, чому система прийняла те чи інше рішення. Це є серйозною перешкодою для застосування таких систем у відповідальних галузях, що вимагають аудитоздатності. Отже, важливою вимогою є прозорість та верифікованість підходу. Вона має надавати стандартизований інтерфейс для доступу до проміжних результатів на будь-якому етапі обчислень для аналізу, налагодження та верифікації.

Нарешті, для забезпечення надійності необхідно гарантувати детермінізм виконання. При роботі зі складними архітектурами, де результат залежить від багатьох проміжних кроків, навіть невеликі відмінності в порядку операцій або числових обчисленнях можуть накопичуватися і призводити до значних розбіжностей у кінцевому результаті. Тому підхід має забезпечувати отримання ідентичних результатів для однакових вхідних даних при кожному запуску, усуваючи будь-які неоднозначності в порядку виконання операцій.

Саме для відповіді на ці вимоги у даній роботі пропонується DAG-орієнтований підхід, детальний опис якого наведено в наступному розділі.

3. Дослідження та результати

3.1. Мета дослідження

Метою даного дослідження є розробка та опис DAG-орієнтованого підходу, що дозволяє представити складний метод конструювання АСМ у вигляді єдиного, самодостатнього та портативного спрямованого ациклічного графа (DAG), забезпечуючи його детерміноване та верифіковане виконання.

Для досягнення цієї мети було розроблено архітектуру DAG та описано ролі і логіку роботи його компонентів. Зокрема, було деталізовано такі процеси, як-от багатошарова рециркуляція та підвищення однорідності даних, вони розбиваються на послідовність логічних етапів у графі.

Також було створено програмний рушій для інтерпретації та детермінованого виконання цього графа, що забезпечує стабільність результатів. Окрему увагу було приділено розробці інструментів для перевірки (валідації) результатів виконання як на кінцевому, так і на проміжних етапах графа. Це підвищує прозорість та аудитоздатність усього процесу.

Результати дослідження будуть використані для демонстрації практичної реалізації запропонованого підходу. Буде показано, як складні методи конструювання АСМ представляються у вигляді DAG, а також проаналізовано результати його виконання для підтвердження детермінізму та верифікованості підходу.

3.2. Методи та технології

Запропонований підхід базується на представленні всього процесу реалізації методу конструювання АСМ у вигляді спрямованого ациклічного графа, описаного у форматі JSON. Такий підхід перетворює набір окремих, незв’язаних артефактів на єдину, цілісну та самодостатню систему. Цей розділ детально описує загальну структуру графа, ролі та логіку роботи його основних компонентів (вузлів), а також формат обміну даними між ними.

3.2.1. Загальна структура графа

Граф виконання (DAG) є декларативним описом, що складається з двох ключових елементів: *nodes* (список усіх вузлів-операцій) та *entry* (ідентифікатор фінального вузла, результат якого є кінцевим результатом роботи всього графа). Кожен вузол у списку *nodes* має унікальний ідентифікатор (*id*), тип (*type*), що визначає його функцію, та список вхідних залежностей (*inputs*), що вказує, результати роботи яких вузлів є для нього вхідними даними.

Така структура дозволяє рушію виконання автоматично визначити правильний, детермінований порядок обчислень шляхом топологічного сортування. Важливою особливістю є те, що вся необхідна для роботи вузла конфігурація (наприклад, шляхи до моделей, параметри агрегації) зберігається безпосередньо в його описі в DAG, що робить граф повністю самодостатнім та незалежним від зовнішнього контексту.

3.2.2. Опис основних вузлів графа: декомпозиційний підхід

Для забезпечення максимальної прозорості та гнучкості процесу було розроблено набір типізованих вузлів. Замість створення великих монолітних вузлів, що виконують багато операцій, було обрано декомпозиційний підхід. Складні процеси, як-от підвищення однорідності, розбиваються на послідовність простих, логічно зрозумілих кроків, кожен з яких реалізується окремим типом вузла. Це дозволяє детально аналізувати та верифікувати кожен етап обчислень. Загальну структуру та зв'язки між основними типами вузлів продемонстровано на рис. 1.

• **Вузол input.** Це початковий вузол будь-якого графа. Його єдине завдання — прийняти на вхід вихідний набір даних (наприклад, у вигляді Pandas DataFrame) та передати його далі по графу. Він не виконує жодних трансформацій, а слугує єдиною точкою входу, що забезпечує узгодженість та чіткий початок обчислювального процесу.

• **Вузол single_model.** Цей вузол є фундаментальним «будівельним блоком» структури графа. Його функція — застосувати одну, попередньо навчену та серіалізовану модель (разом з її об'єктом передобробки) до вхідного набору даних. На вхід він отримує дані від попереднього вузла, застосовує до них модель та генерує прогнози. Цей тип вузла використовується універсально для представлення будь-якої окремої моделі в ансамблях. Важливою особливістю його конфігурації є чітке посилання на конкретні файли: збережену модель (*model_path*) та, за потреби, об'єкт передобробки (*input_pipeline_path*). Це гарантує, що вузол завжди використовує правильні, заздалегідь підготовлені компоненти. Таке об'єднання моделі та її передобробки в одному вузлі робить його самодостатнім та спрощує як аналіз графа, так і його можливі модифікації.

• **Вузли для підвищення однорідності даних.** Процес підвищення однорідності в графі представлений у вигляді послідовності з кількох типів вузлів, що чітко розмежовує логіку.

1. **Segment_predictions.** Для кожного з *k* кластерів у графі створюється свій власний, окремий вузол segment_predictions. Його завдання — застосувати одну конкретну модель, навчену для цього сегмента, до вхідних даних і згенерувати одну нову ознаку — прогноз цієї моделі. Отже, якщо є три кластери, у графі буде три таких вузли (seg_pred_0, seg_pred_1, seg_pred_2).

2. **Cluster_assignment.** Цей вузол використовує навчений класифікатор кластерів (також визначений у маніфесті) для генерації однієї нової ознаки — прогнозованої мітки кластера для кожного екземпляра.

3. **Cluster_homogenization_assemble.** Фінальний вузол цього етапу. Він виконує функцію «збирача»: отримує на вхід вихідні ознаки, прогнози від вузла *segment_predictions* та прогнозовану мітку від *cluster_assignment*. Його завдання — об'єднати всі ці дані, сформувавши збагачений словник ознак у строго визначеному детермінованому порядку, що також задається маніфестом. Така детальна декомпозиція процесу підвищення однорідності, що включає паралельну роботу кількох вузлів segment_predictions та їхнє подальше об'єднання, наочно продемонстрована на рис. 1.

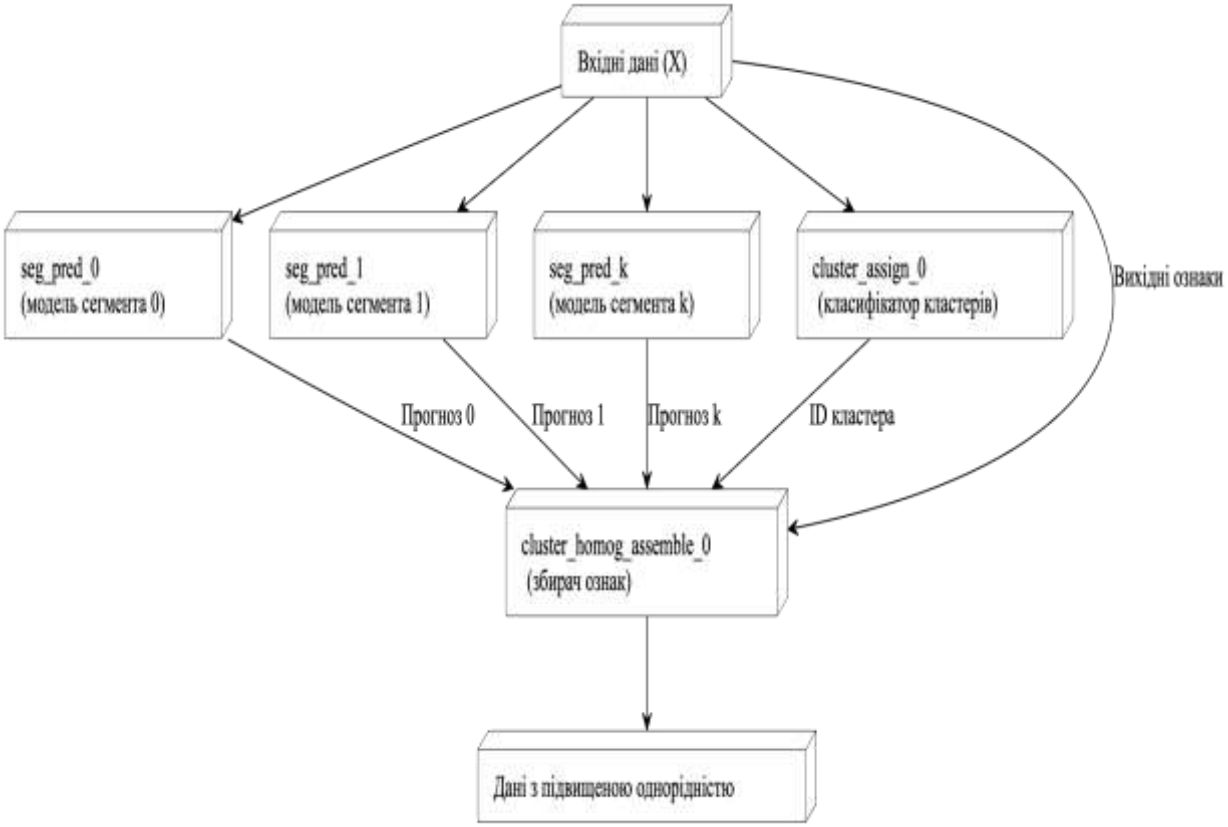

Рисунок 1 — Діаграма діяльності процесу підвищення однорідності даних

• **Вузол recirculation_layer.** Цей вузол є центральним для реалізації багатошарових ансамблів. Він виконує дві ключові функції: збирає прогнози від усіх окремих моделей (*single_model*) поточного шару та готує розширений набір ознак для наступного. Для цього він отримує на вхід дані від попереднього етапу (попереднього шару рециркуляції або вузла *cluster_homogenization_assemble*) і додає до них прогнози моделей поточного шару. Його конфігурація (*gating*) визначає, як саме ці прогнози будуть представлені у вихідному наборі даних: average для простого усереднення, *weighted* для зваженого або просто як окремі нові ознаки. Саме така послідовна передача розширених даних від шару до шару і є суттю методу рециркуляції. Детальний потік даних через два послідовні шари рециркуляції ілюструє рис. 2.

• **Вузол result_selector.** Це кінцевий вузол графа. Він отримує на вхід прогнози від однієї або кількох моделей кінцевого шару та, згідно з визначеною стратегією (наприклад, вибрати найкращий або усереднити), формує єдиний вектор фінальних прогнозів усього методу конструювання АСМ. Наявність цього окремого вузла є важливим архітектурним рішенням. Воно дозволяє чітко відокремити логіку побудови багатошарового ансамблю від логіки вибору кінцевого результату, що робить граф більш гнучким та легким для аналізу.

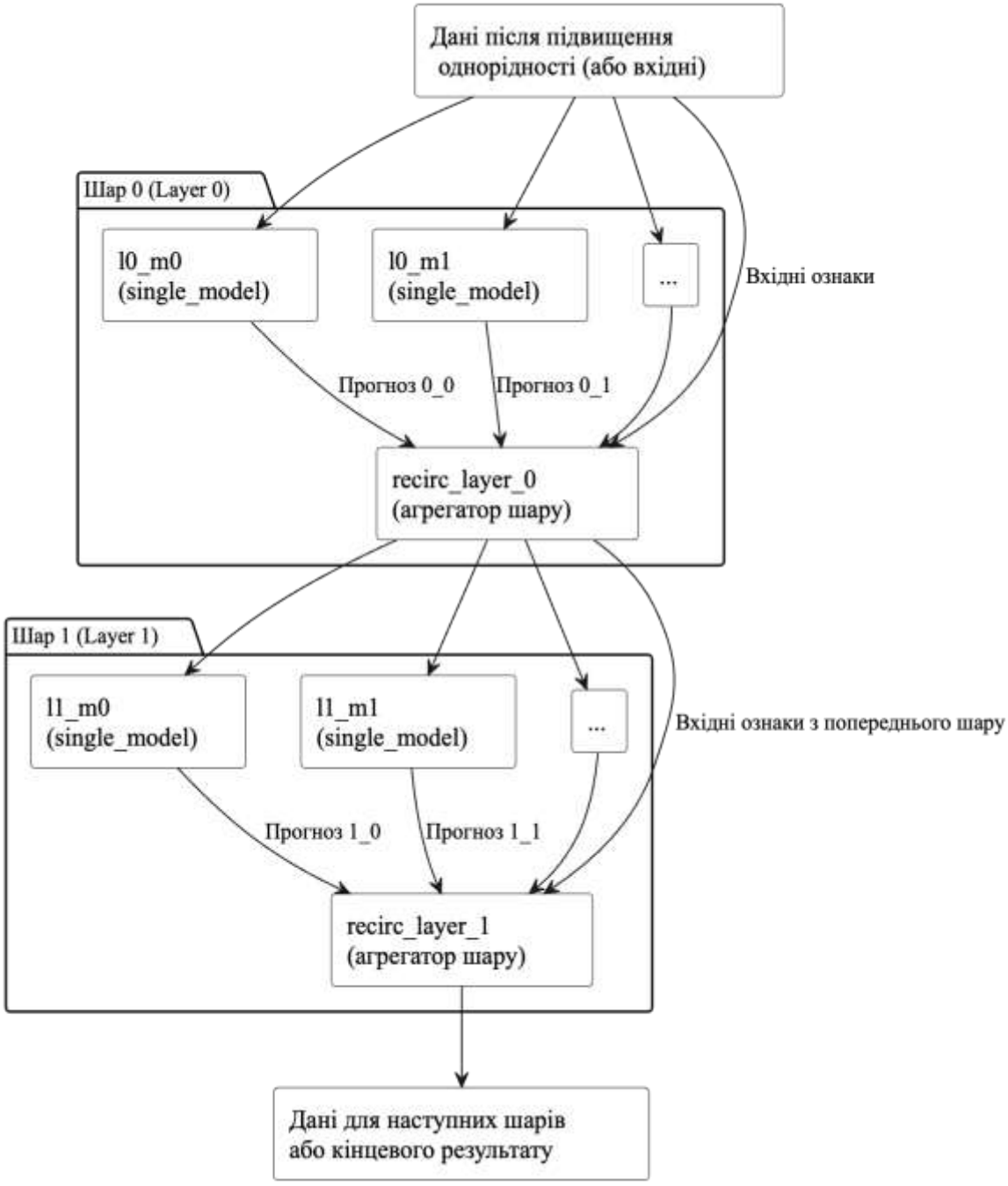

Рисунок 2 — Діаграма діяльності процесу рециркуляції

Отже, комбінуючи ці типи вузлів, можна декларативно описати надзвичайно складні багатоетапні методи конструювання АСМ. Для ілюстрації того, як ці компоненти поєднуються в єдину систему, на рис. 3 наведено загальну структуру DAG, що містить як фазу підвищення однорідності, так і кілька наступних шарів рециркуляції, завершуючись вузлом вибору кінцевого результату.

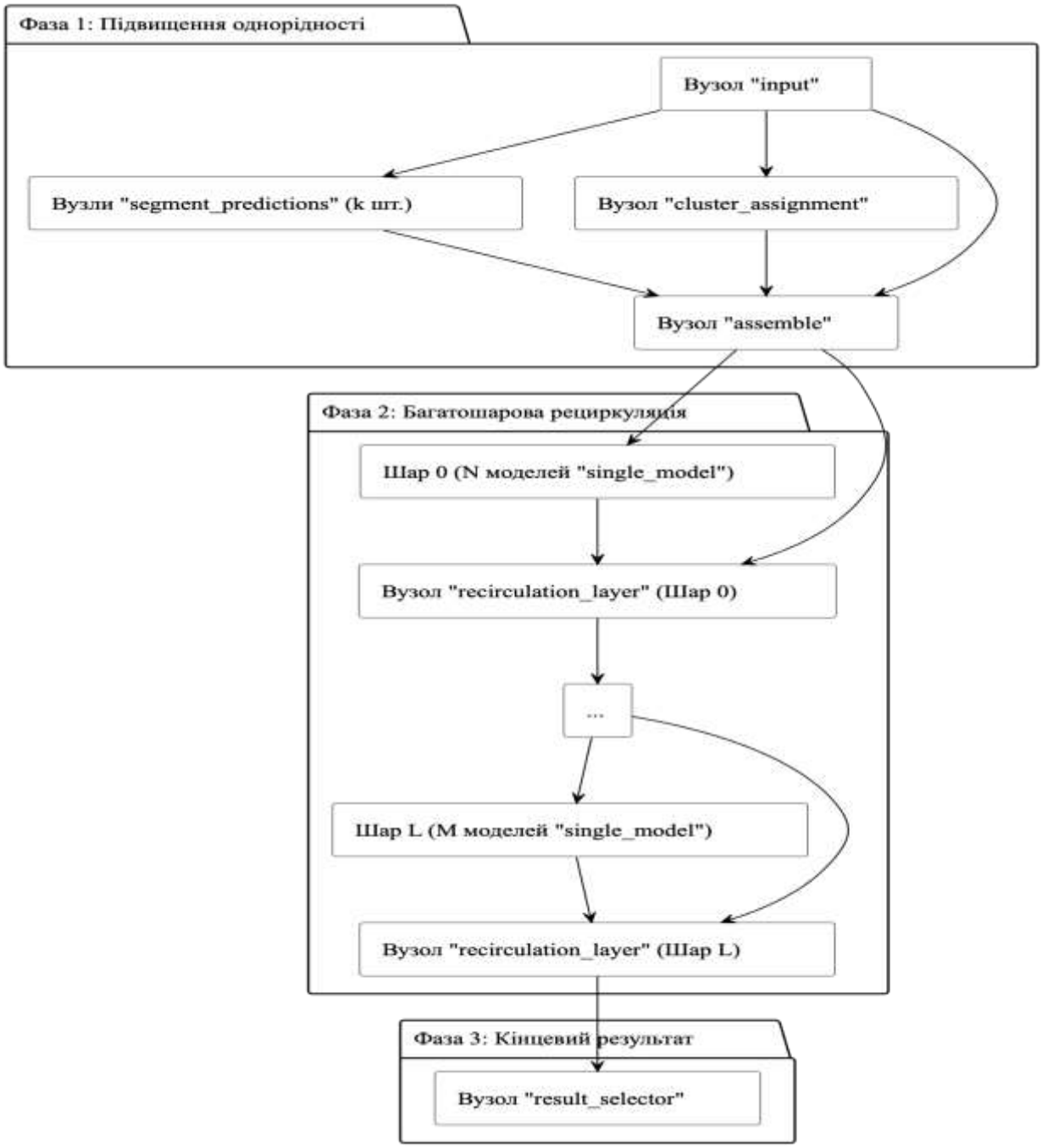

Рисунок 3 — Загальна діаграма діяльності обробки DAG

3.2.3. Формат обміну даними

Для стандартизації обміну даними між вузлами та забезпечення їхньої узгодженої роботи було розроблено єдиний формат — PredictionFrame. Це структура даних, яка інкапсулює результат роботи будь-якого вузла. Вона містить не лише сам вектор прогнозів (result), але й додатково згенеровані ознаки (features), що є ключовим для багатошарових архітектур, де вихід одного шару стає входом для іншого. Крім того, ця структура містить метадані про роботу вузла (його тип, шлях до моделі тощо), що підвищує прозорість та полегшує налагодження. Використання такого єдиного контракту даних значно спрощує логіку рушія виконання та забезпечує надійність всієї системи.

3.3. Деталі проведеного експерименту

У цьому розділі детально описано, як програмний рушій інтерпретує та виконує DAG, як забезпечуються стабільність та відтворюваність результатів, а також які інструменти існують для перевірки та аналізу роботи графа.

3.3.1. Алгоритм виконання графа

Процес виконання графа починається з його валідації. На цьому етапі рушій перевіряє цілісність та коректність структури DAG: контролює відсутність циклів, наявність усіх вузлів, на які є посилання, та існування єдиного кінцевого вузла (entry). Після успішної перевірки виконується топологічне сортування графа. Цей алгоритм аналізує залежності між вузлами та створює лінійну послідовність їх виконання, яка гарантує, що жоден вузол не буде виконаний раніше, ніж будуть готові результати всіх вузлів, від яких він залежить.

Після отримання відсортованого списку рушій ітеративно виконує кожен вузол. Для цього він збирає результати роботи його «батьківських» вузлів (вхідних залежностей) у вигляді PredictionFrame та передає їх на вхід поточному вузлу для обробки. Результат роботи поточного вузла також пакується у PredictionFrame і зберігається для подальшого використання «дочірніми» вузлами.

3.3.2. Механізми забезпечення детермінізму та валідації

Для забезпечення надійності та довіри до результатів підхід до виконання реалізує кілька ключових механізмів. Насамперед забезпечується детермінізм — здатність гарантувати отримання ідентичних результатів для однакових вхідних даних при кожному запуску. Це досягається завдяки строгому порядку виконання, що визначається топологічним сортуванням графа, та використанню фіксованих версій артефактів, де кожен вузол посилається на конкретні незмінні файли навчених моделей. Початкові значення (*seeds*) для моделей, що використовують елементи випадковості, фіксуються ще на етапі навчання, що забезпечує стабільність їхньої поведінки і на етапі виконання [10].

Для забезпечення прозорості та можливості глибокого аналізу розроблено інструменти валідації та аналізу. Система надає механізм верифікації (verify), який дозволяє автоматично обчислювати метрики якості (*RMSE, MAE* тощо) для виходу будь-якого вузла графа, а не лише кінцевого. Це дає можливість, наприклад, порівняти точність прогнозу на різних шарах рециркуляції. Також реалізовано інструменти трасування (trace), які при активації зберігають проміжні результати роботи кожного вузла у вигляді окремих файлів, забезпечуючи повну прозорість обчислень. Крім того, система підтримує можливість експорту та аналізу частини графа (субграфа**)**. Це є корисним інструментом для досліджень, де потрібно оцінити, як видалення певного блока або вузла впливає на кінцевий результат, що допомагає зрозуміти важливість кожного компонента.

3.4. Демонстрація роботи методу та результати валідації

У цьому розділі на конкретному прикладі продемонстровано, як описана архітектура представляє складні методи конструювання АСМ та як інструменти валідації працюють на практиці.

3.4.1. Представлення складної архітектури у вигляді DAG

Для демонстрації розглянемо складну архітектуру, що поєднує як підвищення однорідності даних (з використанням трьох кластерів), так і подальшу багатошарову рециркуляцію глибиною у п’ять шарів. На рис. 4 наведено діаграму, що візуалізує відповідний DAG для цієї структури.

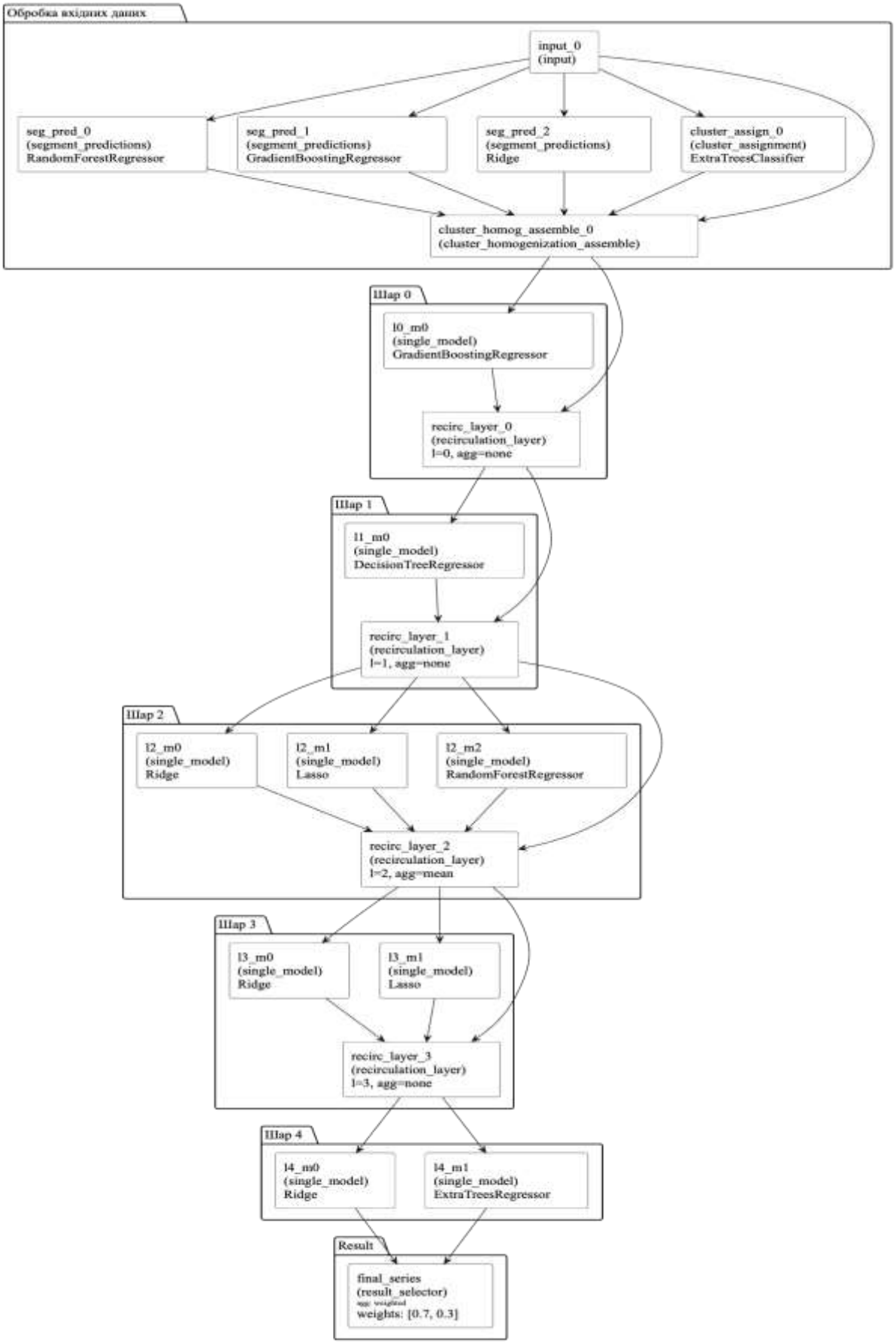

Рисунок 4 — Структура DAG, що був використаний для демонстрації архітектури

Як видно з діаграми, весь процес чітко декомпозовано. Спочатку виконується блок підвищення однорідності («Обробка вхідних даних»), що містить паралельне виконання трьох вузлів *segment_predictions* (по одному для кожної моделі сегмента) та вузла *cluster_assignment*. Їхні результати, разом із вихідними даними, об'єднуються у вузлі *cluster_homogenization_assemble*. Збагачений набір ознак із цього вузла слугує входом для першого шару рециркуляції («Шар 0»). Далі вихід кожного шару рециркуляції (*recirc_layer_k*) стає входом для моделей наступного шару. Фінальний результат формується вузлом *result_selector*, який у даному прикладі застосовує зважене усереднення до прогнозів моделей останнього шару.

Така структура повністю описується у декларативному форматі *dag.json*. Наприклад, вузол *cluster_homogenization_assemble* містить у своїй конфігурації поле *feature_order*, яке чітко визначає порядок та імена всіх ознак у збагаченому наборі даних. Вузол *recirc_layer_2* має параметр *gating_mode: «average»*, що вказує на використання простого усереднення, тоді як фінальний вузол *result_selector* має *aggregation: «weighted»* та поле *weights*, що визначає ваги для кожної з вхідних моделей.

3.4.2. Результати верифікації вузлів графа

Використання механізму верифікації (*verify*) дозволяє провести глибокий аналіз ефективності на рівні окремих компонентів графа. У табл. 1 наведено приклад результатів верифікації для описаної складної архітектури. Для кожного вузла, що генерує прогнози, розраховано метрики якості та визначено їхній ранг (1 — найкращий) серед усіх вузлів. Також розраховано сумарний ранг (сума рангів за всіма метриками) як узагальнюючий показник якості.

Таблиця 1 — Результати верифікації вузлів DAG на тестовій вибірці

Ідентифікатор вузла	Рециркуляція (шар)	Алгоритм	RMSE	MAE	R2	Сумарний ранг
l2_m0	2	Ridge	7,10	4,67	0,978	10
l1_m0	1	Decision Tree	7,09	4,71	0,978	16
l2_m1	2	Lasso	7,42	4,67	0,976	30
recirc_layer_2	2	Рециркуляція (середнє)	7,32	4,78	0,977	31
l4_m1	4	ExtraTree	7,27	4,78	0,977	35
recirc_layer_0	0	Рециркуляція	9,57	4,93	0,961	43

Аналіз таблиці дозволяє зробити кілька важливих спостережень. Наприклад, можна побачити, що точність прогнозу не завжди монотонно зростає з глибиною рециркуляції. У даному прикладі окрема модель *l2_m0* (Ridge на 2-му шарі) демонструє один із найкращих результатів, що підтверджується її низьким сумарним рангом. Це демонструє, як інструмент верифікації дозволяє знаходити ефективні проміжні конфігурації всередині складної архітектури, що може бути використано для її спрощення без значної втрати точності.

3.5. Обговорення результатів та подальші напрями досліджень

Проведене дослідження та описаний підхід демонструють практичні переваги DAG-орієнтованого подання методів конструювання АСМ. Аналіз результатів дозволяє виділити кілька ключових властивостей запропонованої системи, що стосуються її портативності, прозорості та гнучкості.

Запропонований підхід забезпечує портативність та відтворюваність. Представлення всього процесу виконання у вигляді єдиного *dag.json* файлу разом із необхідними моделями створює самодостатній пакет. Такий пакет можна легко переносити між різними середовищами, гарантуючи отримання однакових результатів. Це вирішує ключову задачу розриву

між дослідженням та розгортанням, де результати часто залежать від неявних налаштувань локального середовища, версій бібліотек чи навіть операційної системи.

Важливою перевагою є прозорість та можливість перевірки виконання. Підхід, за яким складні операції розбиваються на послідовність простих вузлів, дозволяє чітко відстежити потік даних та логіку обчислень. Інструменти трасування та верифікації, що дозволяють аналізувати проміжні результати роботи кожного вузла, надають потужні можливості для налагодження, аналізу ефективності окремих компонентів та аудиту всієї структури методу конструювання АСМ. Можливість експорту та аналізу частини графа (субграфа) є корисним інструментом для досліджень, де потрібно оцінити, як видалення певного блока або вузла впливає на кінцевий результат, що допомагає зрозуміти внесок кожного компонента в загальну ефективність. Це робить підхід більш гнучким та легким для модифікації.

Отже, розроблено метод подання складних ансамблевих технік (як-от підвищення однорідності та рециркуляція) у вигляді послідовності простих типізованих вузлів DAG. Було запропоновано підхід, де файл *dag.json* виступає як єдине джерело істини (single source of truth) для всього процесу виконання, інкапсулюючи всю необхідну конфігурацію безпосередньо в своїй структурі, що усуває потребу у зовнішніх маніфестах чи складних файлах налаштувань на етапі виконання. Крім того, реалізовано механізм для перевірки якості на рівні проміжних вузлів графа, що є корисним інструментом для глибокого аналізу та потенційного спрощення багатокомпонентних архітектур.

Практична значущість отриманих результатів полягає у створенні надійного інструментарію для роботи зі складними модельними архітектурами. Запропонований підхід підвищує довіру до складних моделей, спрощує їхній аудит [12] та створює надійний міст для їхнього використання в реальних системах. Для МоПрАг це означає можливість надійного оновлення та розгортання складних внутрішніх моделей, гарантуючи, що їхня поведінка буде стабільною та передбачуваною.

Незважаючи на продемонстровані переваги, запропонований підхід має певні обмеження та напрями для подальших досліджень. Поточна реалізація орієнтована на офлайн-виконання і не оптимізована для обробки даних у режимі реального часу (стрімінгу), що вимагало б розробки механізмів управління станом всередині графа. Перспективним напрямом є також розробка інтерактивних інструментів візуалізації, які б дозволяли в реальному часі аналізувати проміжні результати та потоки даних всередині графа. Майбутні дослідження також можуть бути спрямовані на розширення набору доступних вузлів для підтримки нових типів моделей та алгоритмів агрегації.

4. Висновки

У даній роботі було запропоновано та описано DAG-орієнтований підхід для вирішення задачі надійного та відтворюваного виконання складних методів конструювання АСМ, призначених для використання МоПрАг.

Розроблений підхід представляє весь процес виконання у вигляді єдиного самодостатнього графа, де складні операції представлені у вигляді послідовності простих, типізованих вузлів. Було показано, що такий підхід забезпечує портативність, дозволяючи легко переносити методи конструювання АСМ між різними середовищами у вигляді самодостатніх пакетів. Також було продемонстровано, що фіксований порядок обчислень у графі гарантує детермінізм результатів. Крім того, підхід забезпечує високий рівень прозорості та верифікованості завдяки інструментам для аналізу проміжних етапів обчислень.

Отже, поставлена у роботі мета була досягнута. Отримані результати доводять, що запропонований підхід є ефективним рішенням проблеми розриву між дослідженням та практичним розгортанням. Він створює надійний інструмент для використання складних, але точних моделей у складі програмних агентів, що є важливим для побудови надійних та

передбачуваних систем інтелектуального моніторингу. Подальші дослідження будуть спрямовані на адаптацію методу для роботи з потоками даних та розширення набору підтримуваних вузлів.

СПИСОК ДЖЕРЕЛ

1. Model persistence. *scikit-learn*. URL: https://scikit-learn/stable/model_persistence.html (дата звернення: 19.10.2025).
2. Guazzelli A., Zeller M., Lin W.C. та ін. PMML: An open standard for sharing models. *R Journal*. Вип. 1, N 1. С. 60–65. DOI: https://doi.org/10.32614/rj-2009-010.
3. Wooldridge M. An Introduction to MultiAgent Systems. John Wiley & Sons, 2009. 484 с. ISBN 978-0-470-51946-2.
4. Голуб С.В. Багаторівневе моделювання в технологіях моніторингу оточуючого середовища. Черкаси: Вид. від. ЧНУ імені Богдана Хмельницького, 2007. Р. 220.
5. Breck E., Cai S., Nielsen E. et al. The ML test score: A rubric for ML production readiness and technical debt reduction. *2017 IEEE International Conference on Big Data (Big Data 2017 IEEE International Conference on Big Data (Big Data)*. P. 1123–1132. DOI: https://doi.org/10.1109/BigData.2017.8258038.
6. Dags — Airflow 3.1.0 Documentation. URL: https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/dags.html (accessed 19/10/2025).
7. Holub S.V., Ostapiuk V.V. Machine learning of multilayer models of a monitoring software agent. *Mathematical machines and systems*. 2025. Vol. 2. P. 76–96. DOI: https://doi.org/10.34121/1028-9763-2025-2-76-95.
8. ONNX Concepts — ONNX 1.20.0 documentation. URL: https://onnx.ai/onnx/intro/concepts.html (accessed 19/10/2025).
9. Rocklin M. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. *Python in Science Conference*. Austin, Texas, 2015. P. 126–132. DOI: https://doi.org/10.25080/Majora-7b98e3ed-013.
10. Sculley D., Holt G., Golovin D. et al. Hidden Technical Debt in Machine Learning Systems. *Advances in Neural Information Processing Systems*Curran Associates, Inc., 2015. URL: https://papers.nips.cc/paper_files/paper/2015/hash/86df7dcfd896fcaf2674f757a2463eba-Abstract.html (accessed: 25/10/2025).
11. Zaharia M., Xin R. S., Wendell P. et al. Apache Spark: a unified engine for big data processing. *Commun. ACM*. 2016, Vol. 59, Issue 11. P. 56–65. DOI: https://doi.org/10.1145/2934664.
12. Doshi-Velez F., Kim B. Towards A Rigorous Science of Interpretable Machine Learning. arXiv, 2017. DOI: https://doi.org/10.48550/arXiv.1702.08608.
13. Low Y., Gonzalez J., Kyrola A. et al. GraphLab: A New Framework For Parallel Machine Learning. arXiv, 2014. DOI: https://doi.org/10.48550/arXiv.1408.2041.

Стаття надійшла до редакції 22.12.2025 / прийнята до друку 12.02.2026