НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ
ІНСТИТУТ ПРОГРАМНИХ СИСТЕМ

ПРОБЛЕМИ ПРОГРАМУВАННЯ

науковий журнал

Головний редактор

Сініцин Ігор Петрович

чл.-корр. НАН України, д.т.н., професор,
директор Інституту програмних систем
НАН України

✉ Інститут програмних систем НАН України
проспект Академіка Глушкова, 40, корп. 5
03187, Київ-187

☎ Тел.+380 (44) 526 5507

E-mail: ips2014@ukr.net
http://www.pp.isofts.kiev.ua

Журнал «Проблеми програмування» за **категорією «Б»** включений до переліку наукових фахових видань України, в яких можуть публікуватися результати дисертаційних робіт на здобуття наукових ступенів доктора наук, кандидата наук та ступеня доктора філософії **за кластером «Інформаційні технології та електроніка», галузі науки F2, F3, F4, F5, F6** (Наказ МОН України від 19.01.2026 №56).

Журнал має Свідоцтво про державну реєстрацію друкованого засобу масової інформації, серія КВ, № 7490, а також Свідоцтво про внесення суб'єкта видавничої справи до державного реєстру видавців, виготовлювачів і розповсюджувачів видавничої продукції, серія ДК, № 5778.

Всі матеріали, прийняті до публікації в журналі, підлягають обов'язковому зовнішньому і внутрішньому рецензуванню.

Затверджено до друку вченою радою Інституту програмних систем НАН України.
Протокол №3 від 12.03.2026.

Редактор *З.В. Єгорова*
Комп'ютерна верстка *С. Кравченко*

Підписано до друку 19.03.2026. Дата друку (розміщення онлайн) 16.04.2026. Формат 60x84/8. Папір офс.
Ум. друк. арк. 14,40. Обл.-вид. арк.13,25. Тираж 120 прим. Ціна договірна. Замовлення 107.

НАЦІОНАЛЬНА
АКАДЕМІЯ НАУК УКРАЇНИ
ІНСТИТУТ ПРОГРАМНИХ СИСТЕМ

ПРОБЛЕМИ ПРОГРАМУВАННЯ

науковий журнал

№ 1 січень - березень 2026

Заснований у березні 1999 р.

ЗМІСТ

Свідоцтво про державну реєстрацію КВ № 7490 від 01.07.2003

Науковий журнал “Проблеми програмування” занесений до переліку наукових фахових видань України, в яких можуть публікуватися основні результати дисертаційних робіт.

ISSN 1727-4907

NATIONAL ACADEMY
OF SCIENCES OF UKRAINE
INSTITUTE OF SOFTWARE SYSTEMS

PROBLEMS IN PROGRAMMING

scientific journal

№ 1 ***January – March*** ***2026***

Founded in March, 1999

CONTENT

УДК 004.052.32, 004.43 https://doi.org/10.15407/pp2026.01.004

Р.С. Шевченко, А.Ю. Дорошенко, О.А. Яценко

ANTHILL: МОВА ПРОГРЕСИВНОЇ ФОРМАЛІЗАЦІЇ ДЛЯ ПРОГРАМНИХ ПРОЄКТІВ З АГЕНТНОЮ ПІДТРИМКОЮ

У статті представлено мову Anthill — спеціалізовану мову прогресивної (поступової) формалізації знань, призначену для використання в реальних програмних проєктах з інтеграцією агентної підтримки, зокрема, великих мовних моделей. Anthill дозволяє описувати, накопичувати та підтримувати в єдиній структурованій базі знань доменні знання, функціональні та нефункціональні вимоги, проєктні рішення, інваріанти, обмеження цілісності та доведення властивостей. Відмінною особливістю є те, що база знань зберігається безпосередньо у файловій системі репозиторію проєкту, що забезпечує тісну версійовану синхронізацію специфікацій з кодом і полегшує їхню еволюцію протягом життєвого циклу розробки. Ядро мови зроблено мінімальним і складається з чотирьох базових конструктів: простір імен, сорт, правило та операція. Така архітектура аналогічна ядру сучасних систем автоматизованого доведення теорем (наприклад, Lean, Coq), де мале довірене ядро виконує лише перевірку коректності виведення, тоді як пошук доведень, генерацію правил, уточнення специфікацій та розв'язання зобов'язань делеговано зовнішнім агентам. Головною інновацією Anthill є вбудована підтримка часткової формалізації: одна й та сама мова дозволяє описувати знання на всьому спектрі — від природномовних коментарів у блоках через напівформальні машинно-перевірні правила (клаузи Хорна) до повністю формальних доведень та законів алгебри. Розглянуто три приклади застосування: структуроване управління задачами, специфікація комунікаційних протоколів для вбудованих систем та доведення властивостей на етапі компіляції.

Ключові слова: формальна специфікація, прогресивна формалізація, база знань, стигмергія, мультиагентні системи, великі мовні моделі.

R.S. Shevchenko, A.Yu. Doroshenko, O.A. Yatsenko

ANTHILL: A PROGRESSIVE FORMALIZATION LANGUAGE FOR AGENT-SUPPORTED SOFTWARE PROJECTS

The article presents the Anthill language — a specialized language for progressive (gradual) knowledge formalization, designed for use in real-world software projects with the integration of agent support, in particular large language models. Anthill allows you to describe, accumulate and maintain domain knowledge, functional and non-functional requirements, design solutions, invariants, integrity constraints and property proofs in a single structured knowledge base. A distinctive feature is that the knowledge base is stored directly in the project repository file system, which ensures close versioned synchronization of specifications with code and facilitates their evolution throughout the development life cycle. The core of the language is made minimal and consists of only four basic constructs: namespace, sort, rule and operation. This architecture is similar to the core of modern automated theorem proving systems (e. g. Lean, Coq), where a small trusted core performs only the correctness check of the derivation, while the search for proofs, rule generation, specification refinement, and resolution of commitments are delegated to external agents. The main innovation of Anthill is the built-in support for partial (progressive) formalization: one and the same language allows for the description of knowledge across the entire spectrum — from free-text natural language comments in blocks through semi-formal machine-verifying rules (Horn clauses) to fully formal proofs and laws of algebra. Three application examples are considered: structured task management, specification of communication protocols for embedded systems, and proof of properties at the compilation stage.

Keywords: formal specification, progressive formalization, knowledge base, stigmergy, multi-agent systems, large language models.

Вступ

Специфікації в сучасних програмних проєктах існують у двох крайностях. З одного боку — неформальні описи: задачі в системі управління проєктами, коментарі в коді, текстові документи з вимогами. Вони доступні людям, але не перевіряються машиною: неузгодженість між вимогою та реалізацією виявляється лише під час тестування. З іншого боку — повністю формальні специфікації в системах доведення теорем (Lean [1], Coq [2], Isabelle [3]) або алгебраїчних мовах специфікацій (OBJ [4], Maude [5]). Вони забезпечують математичну строгість, але потребують спеціалізованих знань, і поріг входу для більшості таких програмних проєктів надто високий.

Поява великих мовних моделей (Large Language Models — LLM) як учасників розробки програмного забезпечення загострює цю проблему. LLM-агенти здатні генерувати код, але працюють без стійкої структури знань: кожна сесія починається з порожнього контексту, а результати попередньої роботи зберігаються лише як текст у файлах. Відсутність формальних інваріантів означає, що агент може порушити неявні припущення проєкту, які ніде не зафіксовані машинно-перевірним чином.

Ми пропонуємо третю точку: мову *часткової формалізації з поступовим уточненням*. Anthill дозволяє починати з природномовних описів і поступово замінювати їх на формальні правила та обмеження — без необхідності формалізувати все одразу. База знань (БЗ) живе безпосередньо у файловій системі проєкту (каталог anthill/ поруч з src/ та tests/) і еволюціонує разом із кодом.

Архітектура Anthill спирається на поділ, що зарекомендував себе в системах доведення теорем: *мале довірене ядро* перевіряє правила та обмеження, а *великі недовірені агенти* (LLM, скрипти, люди) шукають рішення та докази. Ядру байдуже, як доведення було знайдене — формальним рушієм, SMT-солвером (Satisfiability Modulo Theories solver) [6], нейронною мережею чи людиною — важливо лише, що перевірка пройдена.

Стаття організована наступним чином. Розділ 1 описує принципи проєктування. У розділі 2 дається стислий огляд мови. Розділ 3 розглядає концепцію легкої аплікації в репозиторії та агентний цикл зворотного зв'язку. Розділ 4 представляє три приклади застосування, а розділ 5 обговорює зв'язок з існуючими роботами.

1. Принципи проєктування

1.1. Мінімальне ядро. Ядро мови містить рівно чотири конструкти:

- namespace — простір імен з імпортом, експортом та вкладенням;
- sort — тип (параметричний, абстрактний або визначений);
- rule — правило (клауза Хорна), єдиний примітив знань;
- operation — оголошення операції з типізованим контрактом та ефектами.

Ці конструкти дозволяють описувати багатосортні алгебри в традиції алгебраїчних мов специфікацій OBJ [4] та Maude [5]: простори імен визначають сигнатури, сорти задають носії, операції — це функціональні символи, а правила — аксіоми (тотожності та клаузи Хорна). Решта конструкцій мови зводяться до цих чотирьох. Зокрема, entity — це сорт з єдиним конструктором (алгебра з одним породжувальним елементом); fact — це правило без тіла, тобто безумовна аксіома; constraint — це правило без голови, тобто обмеження цілісності (denial у термінології логічного програмування).

Такий підхід аналогічний до архітектури систем типу proof assistant (Lean [1], Coq [2]): мале довірене ядро перевіряє виведення, тоді як тактики (великі, недовірені) шукають докази.

Ядро є мовонезалежним: сорти, правила та операції описують доменну логіку без прив'язки до конкретної мови реалізації. Ядро може бути підключене до мови-хоста через механізм підключення зовнішніх функцій (Foreign Function Interface — FFI) або реімплементоване для цієї мови; наразі підтримуються Rust та Scala. Вбудовування відбувається через механізм *реалізацій*, який відображає абстрактні

сорти на типи хост-мови, операції — на функції, а ефекти — на відповідні конструкції хост- мови.

1.2. Подання знань через правила. Всі знання в БЗ представляються у вигляді клауз Хорна:

- факт (правило без тіла): безумовна істина:

```
fact Eq{Int}
```

- правило виведення: length визначається рекурсивно:

```
rule length(nil) = 0

rule length(cons(?x, ?xs)) =
add(1, length(?xs))
```

- обмеження (правило без голови): дільник не може бути нулем:

```
constraint div_nonzero: neq(?b, 0) :-
divExact(?_, ?b)
```

Рушій виведення поєднує два механізми: SLD-резолюцію (як у Prolog) для логічного виведення над клаузами Хорна та екваціональне переписування для спрощення термів згідно з аксіомами алгебри. Хоча це різні обчислювальні моделі (резолюція працює з цілями та підстановками, а переписування — з напрямленими тотожностями), на рівні реалізації обидва зводяться до єдиної базової операції: співставлення терму з патерном та застосування підстановки. Цей підхід відтворює архітектуру системи TermWare [7], де правила переписування та логічне виведення об'єднані через контекстні терми з ефективною диспетчеризацією.

Кожний факт несе метадані: хто його створив, коли, з яким рівнем довіри та в якій ітерації. Кожний факт несе рівень довіри, що може поступово підвищуватися: від початкового припущення через підтвердження тестами, верифікацію автоматичним інструментом (наприклад, SMT-солвером) до повного формального доведення, перевіреного довіреним ядром.

1.3. Часткова формалізація. Будь-яке оголошення може мати один або декілька блоків опису ({< >}) — вільнотекстові описи, що зберігаються як факти в БЗ. На відміну від коментарів, блоки опису є структурними: вони доступні для запитів та аналізу. В поєднанні з анонімними змінними (?) це дає спектр від повністю неформального до повністю формального:

- повністю неформальне:

```
sort Account
  {< Банківський рахунок.
     Має баланс, власника,
     та підтримує операції зняття та
     поповнення. >}
  sort T = ?
end
```

- частково формальне:

```
sort Account
  {< Банківський рахунок >}

  entity account(id: AccountId,
  owner: String, balance: Money)

  operation withdraw(amount: Money)
  -> Account

  constraint positive_balance:
  gte(?bal, 0) :- account(?_, ?_, ?bal)
end
```

Всюди, де очікується ім'я типу або вираз, можна використовувати логічні змінні: ?x — іменована змінна (спільна в межах області видимості); ? — анонімна (кожне входження позначає окрему змінну). Зокрема, sort T = ? визначає абстрактний параметр типу — сорт, що буде уточнений під час інстанціації.

2. Огляд мови

Цей розділ дає стислий огляд мови, достатній для розуміння прикладів. Повна специфікація доступна як частина репозиторію проєкту [8].

2.1. Простори імен та модульність. Простори імен організують код ієрархічно, з імпортом та експортом символів:

```
namespace anthill.prelude.Int
  import anthill.prelude.{Eq, Ordered,
                          Numeric}
  export abs, neg, mod, rem, sign,
         divExact
  fact Eq{Int}
```

```
  fact Ordered{Int}
  fact Numeric{Int}
end
```

2.2. Сорти. Сорти описують типи. Параметричний сорт має абстрактні параметри (sort T = ?):

```
sort anthill.prelude.List

  export List, nil, cons, length,
         member, append

  sort T = ?  -- параметр типу
              -- (абстрактний)

  entity nil   -- порожній список

  -- клітинка
   entity cons(head: T, tail: List)

  operation length(l: List) -> Int

  rule length(nil) = 0

  rule length(cons(?x, ?xs)) =
  add(1, length(?xs))

end
```

Задоволення специфікації декларується через факти. Запис fact Eq{Int} означає, що тип Int задовольняє специфікацію Eq — операції eq та neq доступні для значень Int. На відміну від ключового слова instance у мові Haskell, тут задоволення специфікації — це звичайний факт в БЗ, який може мати метадані та рівень довіри.

2.3. Специфікації та вимоги. Специфікації — це сорти з абстрактними операціями та правилами-законами:

```
sort anthill.prelude.Ordered
  sort T = ?
  requires Eq{T}
  operation compare(a: T, b: T) -> Int

  -- Похідні операції
  operation gt(a: T, b: T) -> Bool
  rule gt(?a, ?b) =
  gt(compare(?a, ?b), 0)

  -- Закони
  rule compare_antisym:
  compare(?a, ?b) =
  neg(compare(?b, ?a))
  rule compare_refl:
  compare(?a, ?a) = 0

end
```

Зв'язок requires Eq{T} означає: для будь-якого типу, що реалізує Ordered, має існувати факт Eq{T}.

2.4. Операції та ефекти. Операції мають типізований контракт та оголошення ефектів:

```
operation withdraw(account: Account,
          amount: Money) -> Account
  requires gte(amount, 0)
  ensures gte(balance(result), 0)
  effects (Modify(ledger))
```

Ефекти явно декларують, які ресурси операція модифікує, читає або може перервати. Це забезпечує можливість статичного аналізу залежностей між - операціями.

3. Аплікація в репозиторії та агентний цикл

Anthill-застосунок — це набір .anthill файлів, що живуть безпосередньо в репозиторії проєкту поруч з src/ та tests/, еволюціонують разом з кодом і версіонуються в тому ж репозиторії.

Центральне поняття агентної взаємодії — *зобов'язання* (obligation): логічне твердження, яке має бути виконане. Це може бути твердження, яке потрібно довести; операція без реалізації; обмеження, для якого потрібно побудувати свідчення; або декомпозиція складної задачі на підзадачі. Зобов'язання формулюються як звичайні терми та правила мови Anthill. Агент (LLM, скрипт, людина або формальний рушій) бере зобов'язання та надає відповідь — доведення, реалізацію, факт з відповідним рівнем довіри. Ядро перевіряє, чи відповідь задовольняє зобов'язання.

Координація між агентами відбувається через спільне середовище — базу знань — без центрального оркестратора (принцип *стигмергії*). Стан БЗ визначає поточну роботу: агент спостерігає відкриті зобов'язання, порушені обмеження, операції без реалізації — і обирає, що виконувати. Результат записується як факт з метаданими походження (провенансу). Якщо факт визнано некоректним, виникає *кон-*

тамінація: залежні факти автоматично втрачають довіру, блокуючи подальшу роботу, поки неузгодженість не буде усунена.

Формалізація не відбувається одномоментно — різні частини проєкту можуть перебувати на різних рівнях: від вільнотекстових описів (рівень 0), через доменну модель як багатосортну алгебру (рівень 1), формальні обмеження та специфікації з законами (рівень 2), до формальних доведень з верифікацією зовнішнім рушієм (рівень 3). Зобов'язання та агентний цикл працюють однаково на всіх рівнях.

4. Приклади застосування

4.1. Декомпозиція задач як зобов'язання. Зобов'язання в Anthill можуть описувати декомпозицію задач — розбиття складної цілі на підзадачі з формальними залежностями та критеріями прийому. Кожна задача — це факт в БЗ:

```
workitem WI-AUTH-001 {
  description: "Define User entity and
                authentication traits"
  acceptance: Compiles({ path: "src",
                         scope: Main })
  depends_on: []
  status: Open
  [trust: proposed, agent: "architect"]
}

workitem WI-AUTH-002 {
  description: "Implement JWT token
                generation and validation"
  acceptance: Compiles({path:"src"}),
  ToolPasses(cargo-test)
  depends_on: [WI-AUTH-001]
  status: Open
  [trust: proposed, agent: "architect"]
}
```

Правила виведення визначають логіку робочого процесу — наприклад, які задачі готові до виконання:

```
rule claimable(?id, ?desc)
  :- WorkItem(id: ?id, status: Open,
              description: ?desc),
     all_deps_verified(?id)

rule blocked(?id, ?desc)
  :- WorkItem(id: ?id, status: Open,
              description: ?desc),
     not(all_deps_verified(?id))
```

Запит claimable(?id, ?desc) — це логічний висновок над станом БЗ через SLD-резолюцію. Та ж механіка працює і для більш формальних зобов'язань: «які обмеження порушені», «які операції не мають реалізації».

4.2. Специфікація комунікаційних протоколів. Розглянемо більш формальний приклад: специфікація правил та обмежень для системи моделювання безпілотних апаратів. У роботі [9] симуляційне середовище (на базі Bevy ECS) взаємодіє з агентами (Python, Scala) через gRPC, передаючи повідомлення за протоколом MAVLink. Поведінка агента — це функція, що на основі поточних показів сенсорів та історії видає команди актуаторам. Агенти реалізуються різними мовами, але мають відповідати спільним обмеженням безпеки та протоколу.

Anthill дозволяє формалізувати три категорії знань, які в поточній реалізації існують лише неявно, а саме: обмеження місії, автомат станів MAVLink і правила деградації сенсорів.

Обмеження місії. Місія (пошук, патрулювання, доставка) має властивості, які мають виконуватися протягом усього польоту:

```
namespace blefusku.mission
  import anthill.prelude.{Int, Float,
                          Bool, List}

  entity DroneState(id: String, position:
Vector3D, altitude: Float, battery: Float)

  -- Дрон має залишатися в межах
  -- геозони
  constraint geofence:
    in_zone(?pos, ?zone)

    :- DroneState(id: ?d, position: ?pos),
       Mission(drone: ?d, zone: ?zone)
  -- Безпечна відстань між дронами
```

```
  constraint safe_separation:
    distance(?p1, ?p2) > 100.0
    :- DroneState(id: ?d1, position:
                  ?p1),
       DroneState(id: ?d2, position: ?p2),
       ?d1 != ?d2

  -- Заряд батареї має бути достатнім
  -- для повернення

  constraint battery_reserve:
    ?battery > return_energy(?pos,
    ?home)
    :- DroneState(id: ?d, position: ?pos,
                  battery: ?battery),
       HomeBase(drone: ?d, position:
                  ?home)
end
```

Автомат станів MAVLink. Протокол MAVLink визначає автомат станів автопілота (Disarmed → Armed → Takeoff → Guided → Land), де певні команди допустимі лише в певних станах. Ці правила наразі розкидані по документації та реалізаціях:

```
namespace blefusku.mavlink_fsm

  -- Допустимі переходи між
  -- станами

  rule valid_transition(Disarmed,
     Armed) :- PrearmChecks(ok)

  rule valid_transition(Armed,
                        Takeoff)

  rule valid_transition(Takeoff,
                        Guided)
    :- DroneState(altitude: ?a), ?a >
       min_takeoff_alt

  rule valid_transition(Guided, Land)

  rule valid_transition(Land,
                        Disarmed)

    :- DroneState(altitude: ?a), ?a < 0.5

  -- Команда допустима лише у
  -- відповідному стані

  constraint valid_command:
    valid_in_state(?cmd, ?state)
      :- CommandSent(command: ?cmd),
         FlightState(state: ?state)
end
```

Правила деградації сенсорів. Поведінка у разі втраті GPS-сигналу або дрейфі IMU — це доменне знання, яке має бути явним:

```
namespace blefusku.sensors
  rule gps_reliable(?d)
    :- SensorData(drone: ?d, type: gps,
                  satellites: ?n, fix_type: ?f),
       ?n >= 6, ?f >= 3

  rule navigation_mode(?d,
                       gps_primary)
    :- gps_reliable(?d)

  rule navigation_mode(?d,
                  imu_dead_reckoning)
    :- not(gps_reliable(?d)),
       SensorData(drone: ?d, type: imu)

  -- в режимі Guided дрон має мати
  -- навігацію
  constraint must_have_navigation:
    navigation_mode(?d, ?)
    :- DroneState(id: ?d, state: Guided)
end
```

Сформульовані правила та обмеження можуть використовуватися двома способами. По-перше, *верифікація трас*: записані траси виконання агентів (послідовності станів, сенсорних даних та команд) завантажуються в БЗ як факти, після чого перевіряються обмеження — порушення виявляються перед- і пост- умови автоматично. По-друге, *аналіз коду агентів*: обмеження Anthill можуть бути відображені на перед- і пост- умови хост- мови (наприклад, requires/ensures в Scala або assert в Python), що дозволяє перевіряти їх статично або під час тестування.

Зв’язок з інтерфейсно-орієнтованим підходом до моделювання мультиагентних систем [9]: Anthill надає формальний шар специфікації поверх gRPC-інтерфейсів [10], дозволяючи виражати інваріанти, які не можуть бути описані лише засобами protobuf [11], — обмеження

місії, автомат станів протоколу та правила деградації.

4.3. Доведення на етапі компіляції. Попередні приклади ілюструють використання Anthill для специфікації доменних знань. В рамках поточної роботи ми також досліджуємо використання БЗ Anthill як рушія доведень, вбудованого безпосередньо в компіляційний конвеєр мови-хоста.

Ідея полягає у створенні системи верифікації властивостей на етапі компіляції для Scala 3. Центральним є тип Checked[A, P], що позначає значення типу A, для якого доведено предикат P. Предикати записуються як типорівневий AST: @pre(_ > 0) на параметрі означає, що значення має тип Checked[A, GT[Self, Lit[0]]], а @post(_ >= 0) на результаті — що повернене значення несе доведення невід'ємності.

Арифметичні операції над Checked-значеннями оголошують *зобов'язання доведення* як using-параметри — наприклад, def +[Q](b: Checked[A, Q])(using proof: ProofAdd[P, Q]): Checked[A, proof.Out]. Компілятор шукає відповідний given-instance через механізм implicit resolution, який є рушієм зворотного виведення (backward chaining): given відповідає правилу виведення, fact — аксіомі, summon — зобов'язанню доведення.

Прості доведення вирішуються прямими правилами given-instances (наприклад, given mulNonNeg: ProofMul[NonNeg, NonNeg] with { type Out = NonNeg }). Коли прямих правил недостатньо, компілятор звертається до macro-given, який делегує до БЗ Anthill.

Предикати типорівневого AST в Scala відображаються на терми Anthill (наприклад, GT[Self, Lit[0]] стає gt(self, lit(0))), а правила доведення зберігаються в БЗ. Наведемо приклад: набір правил, що описує безпечний доступ до масиву, де потрібно довести, що індекс знаходиться в межах.

```
namespace ctproof.array_safety
  import anthill.prelude.{Int, Bool}
  -- Базові аксіоми послаблення та
транзитивності
  rule ?a >= ?b :- ?a > ?b
  rule ?a >= ?c :- ?a >= ?b, ?b >= ?c

  -- Індекс в межах масиву
  rule in_bounds(?i, ?n) :- ?i >= 0,
                            ?i < ?n
  -- Якщо індекс в межах
  -- i n == length(arr),

  -- то доступ arr(i) безпечний
  rule safe_access(?arr, ?i)
    :- in_bounds(?i, ?n), ?n =
       length(?arr)

  -- Індукційний крок: доступ до
  -- хвоста потребує нового
  -- доведення зі зменшеним
  -- індексом

  Rule requires_proof(
    safe_access(?arr2, ?i - 1))

    :- safe_access(?arr, ?i),
      ?arr2 = tail(?arr)

  -- Ціле ділення: якщо дільник
  -- додатний і ділене невід'ємне,
  -- результат невід'ємний

  rule (?a / ?b) >= 0 :- ?a >= 0, ?b > 0

  -- Композиція: якщо f зберігає P,
  -- і g зберігає Q, то compose(g, f)
  -- зберігає and(P, Q)

  rule preserves(compose(?g, ?f), ?p
                          and ?q)
    :- preserves(?f, ?p), preserves(?g,
                                    ?q)
end
```

Коли Scala-компілятор зустрічає операцію над Checked-значенням (наприклад, arr(i) де arr: Checked[Array[T], _]), макрос генерує зобов'язання доведення safe_access(arr, i). Спочатку доведення шукається в контентно-адресованому кеші: якщо для даного зобов'язання вже існує збережене доведення і його передумови не змінилися, воно повторно перевіряється ядром і використовується без повторного пошуку. Якщо кешоване доведення відсу-

тнє або стало невалідним (наприклад, через зміни в коді, що вплинули на передумови), зобов'язання передається до БЗ Anthill для пошуку нового доведення. Якщо і було отримано з конструкції if (i < arr.length), передумова (i >= 0) and (i < length(arr)) вже є в контексті, і доведення будується автоматично через SLD-резолюцію. Для арифметичних підцілей, що виходять за межі можливостей резолюції, викликається Z3 SMT-солвер як вбудована операція.

Таким чином, Anthill працює як *вбудований рушій доведень* — активна частина компіляційного конвеєра, а не лише формат зберігання знань. Трирівнева архітектура (пряма резолюція given → SLD-резолюція Anthill → Z3 SMT) дозволяє балансувати між швидкістю та потужністю: більшість доведень вирішуються на першому рівні без накладних витрат.

5. Огляд суміжних досліджень

Anthill знаходиться в традиції алгебраїчних мов специфікацій OBJ [4], CafeOBJ [12], Maude [5] та використовує архітектурний принцип proof assistants Lean 4 [1], Coq [2], Isabelle [3] — мале довірене ядро перевіряє докази, тоді як великі недовірені тактики їх шукають.

Питання формальних гарантій для ШІ-агентів набуває актуальності в кількох напрямках. Universalis [13] — це мова для програмного синтезу класу AI-first, побудована на Prolog, де перед- та пост-умови вбудовані в семантику як механізм безпеки ШІ. В роботі [14] пропонують capture checking в Scala 3 як оболонку "safety harness" для агентів — статичне відстеження спроможностей (capabilities) запобігає витоку інформації та несанкціонованим побічним ефектам. Обидва підходи підтверджують тенденцію до використання формальних методів замість суто статистичних методів вирівнювання (RLHF).

В роботі [15] надається огляд взаємодії формальних методів та LLM: автоформалізація, LLM-керований пошук доведень, нейронне доведення теорем (AlphaProof [16]). В екосистемі Lean 4 це реалізовано в проєктах Lean Copilot [17] та APOLLO [18] — агентних фреймворках, де LLM інтегровані як тактики пошуку доведень. Окремий напрямок — використання машинного навчання як евристики для прискорення формального пошуку: графові нейронні мережі (Graph neural networks — GNN) для ранжування рівнянь у солвері [19], scored logic monad для направлення логічного бектрекінгу [20].

Висновки

Представлено мову Anthill для прогресивної формалізації знань у програмних проєктах. Мова дозволяє описувати багатосортні алгебри з аксіомами у вигляді клауз Хорна. Ключовою є підтримка часткової формалізації: спектр від вільнотекстових описів до формальних доведень в одній мові з єдиним механізмом рівнів довіри та провенансу.

Наведені приклади демонструють застосування на різних рівнях формалізації: від декомпозиції задач із машинно-перевірними критеріями, через специфікацію обмежень протоколів та правил деградації для мультиагентних систем, до верифікації властивостей на етапі компіляції з використанням БЗ як рушія доведень.

Повна специфікація мови та реалізація доступні за адресою https://github.com/rssh/anthill.

Література

1. L. de Moura, S. Ullrich, The Lean 4 theorem prover and programming language, *28th International Conference on Automated Deduction (CADE 2021). Lecture Notes in Computer Science.* 2021. Vol. 12699. P. 625–635. https://doi.org/10.1007/978-3-030-79876-5_37
2. The Coq Development Team. The Coq Proof Assistant. Accessed: 18.03.2026. https://coq.inria.fr
3. T. Nipkow, L. Paulson, M. Wenzel, Isabelle/HOL — a proof assistant for higher-order logic, Springer, 2002.
4. J. Goguen, T. Winkler. Introducing OBJ. Technical Report, SRI International, 1988.
5. M. Clavel et al., Maude: specification and programming in rewriting logic, *Theoretical Computer Science*. 2002. Vol. 285, N 2. P. 187–243. https://doi.org/10.1016/S0304-3975(01)00359-0

6. L. de Moura, N. Bjørner, Z3: An Efficient SMT Solver, in: Ramakrishnan, C.R., Rehof, J. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2008. Lecture Notes in Computer Science. 2008. Vol. 4963. P. 337–340. https://doi.org/10.1007/978-3-540-78800-3_24
7. Р. С. Шевченко, А. Ю. Дорошенко, TermWare-3 — система переписування термів, заснована на контекстному численні, *Проблеми програмування*. 2019. № 1. https://doi.org/10.15407/pp2019.01.048
8. Anthill kernel language specification. Accessed: 18.03.2026. https://github.com/rssh/anthill
9. Р. С. Шевченко, А. Ю. Дорошенко, В. О. Лесик, О. В. Савчук, О. А. Яценко, Інтерфейсно-орієнтований підхід до засобів моделювання мультиагентних систем, *Проблеми програмування*. 2025. № 1. С. 110–117. https://doi.org/10.15407/pp2025.01.110
10. gRPC. Accessed: 18.03.2026. URL: https://grpc.io/
11. Protocol Buffers. Accessed: 18.03.2026. https://protobuf.dev
12. R. Diaconescu, K. Futatsugi, CafeOBJ Report. World Scientific, 1998.
13. E. Meijer, Unleashing the power of end-user programmable AI, *ACM Queue*. 2025. Vol. 23, N 3. Accessed: 18.03.2026. https://spawn-queue.acm.org/doi/10.1145/3746223
14. M. Odersky, Y. Zhao, Y. Xu, O. Bračevac, C.N. Pham, tracking capabilities for safer agents, *arXiv:2603.00991*. 2026. P. 1–21. https://doi.org/10.48550/arXiv.2603.00991
15. K. Yang, G. Poesia, J. He, W. Li, K. Lauter, S. Chaudhuri, D. Song, Formal mathematical reasoning: a new frontier in AI, *arXiv:2412.16075*. 2025. P. 1–42. https://doi.org/10.48550/arXiv.2412.16075
16. Google DeepMind. AI achieves silver-medal standard solving International Mathematical Olympiad problems (AlphaProof announcement). 2024. Accessed: 18.03.2026. https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/
17. P. Song et al., Lean Copilot: large language models as copilots for theorem proving in Lean, *arXiv:2404.12534*, 2024. P. 1–26. https://doi.org/10.48550/arXiv.2404.12534
18. X. Hu et al., APOLLO: automated LLM and Lean collaboration for advanced formal reasoning, *arXiv:2505.05758*. 2025. P. 1–28. https://doi.org/10.48550/arXiv.2505.05758
19. P. A. Abdulla, M. F. Atig, J. Cailler, C. Liang, P. Rümmer, When GNNs met a Word equations solver: learning to rank equations, *arXiv:2506.23784*, 2025. https://doi.org/10.48550/arXiv.2506.23784
20. R. Shevchenko, A. Doroshenko, O. Yatsenko, A. Nemish, Merging logic and the coinductive selection monad: mixing machine learning into logical search. *ICTERI 2025. Communications in Computer and Information Science*. Vol. 2763. 2025. P. 33–46. https://doi.org/10.1007/978-3-032-10477-9_3

References

1. L. de Moura, S. Ullrich, The Lean 4 theorem prover and programming language, in: *28th International Conference on Automated Deduction (CADE 2021), Lecture Notes in Computer Science* 12699 (2021) 625–635. doi: 10.1007/978-3-030-79876-5_37
2. The Coq Development Team. The Coq Proof Assistant. Accessed: 18.03.2026. https://coq.inria.fr
3. T. Nipkow, L. Paulson, M. Wenzel, Isabelle/HOL — a proof assistant for higher-order logic, Springer, 2002.
4. J. Goguen, T. Winkler. Introducing OBJ. Technical Report, SRI International, 1988.
5. M. Clavel et al., Maude: specification and programming in rewriting logic, in: *Theoretical Computer Science* 285 (2002) 187–243. doi: 10.1016/S0304-3975(01)00359-0
6. L. de Moura, N. Bjørner, Z3: An Efficient SMT Solver, in: Ramakrishnan, C.R., Rehof, J. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2008. Lecture Notes in Computer Science 4963 (2008) 337–340. doi: 10.1007/978-3-540-78800-3_24
7. R. S. Shevchenko, A.Yu. Doroshenko, TermWare-3 – term rewriting system, based on context-term calculus, *Problems in programming* 1 (2019) 48–58. https://doi.org/10.15407/ pp2019.01.048 [in Ukrainian]
8. Anthill kernel language specification. Accessed: 18.03.2026. https://github.com/rssh/anthill
9. R.S. Shevchenko, A.Yu. Doroshenko, V.O. Lesyk, O.V. Savchuk, O.A. Yatsenko, Interface-oriented approach to modelling tools for multi-agent systems, *Problems in programming* 1 (2025) 110–117.

https://doi.org/10.15407/pp2025.01.110 [in Ukrainian]
10. gRPC. Accessed: 18.03.2026. URL: https://grpc.io/
11. Protocol Buffers. Accessed: 18.03.2026. https://protobuf.dev
12. R. Diaconescu, K. Futatsugi, CafeOBJ Report. World Scientific, 1998.
13. E. Meijer, Unleashing the power of end-user programmable AI, in: *ACM Queue* 23 (2025) Accessed: 18.03.2026. https://spawn-queue.acm.org/doi/10.1145/3746223
14. M. Odersky, Y. Zhao, Y. Xu, O. Bračevac, C.N. Pham, tracking capabilities for safer agents, in: *arXiv:2603.00991* (2026) 1–21. doi: 10.48550/arXiv.2603.00991
15. K. Yang, G. Poesia, J. He, W. Li, K. Lauter, S. Chaudhuri, D. Song, Formal mathematical reasoning: a new frontier in AI, in: *arXiv:2412.16075* (2025) 1–42. doi: 10.48550/arXiv.2412.16075
16. Google DeepMind. AI achieves silver-medal standard solving International Mathematical Olympiad problems (AlphaProof announcement). 2024. Accessed: 18.03.2026. https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/
17. P. Song et al., Lean Copilot: large language models as copilots for theorem proving in Lean, *arXiv:2404.12534* (2024) 1–26. doi: 10.48550/arXiv.2404.12534
18. X. Hu et al., APOLLO: automated LLM and Lean collaboration for advanced formal reasoning, *arXiv:2505.05758* (2025) 1–28. doi: 10.48550/arXiv.2505.05758
19. P. A. Abdulla, M. F. Atig, J. Cailler, C. Liang, P. Rümmer, When GNNs met a Word equations solver: learning to rank equations, in: *arXiv:2506.23784* (2025) doi: 10.48550/arXiv.2506.23784
20. R. Shevchenko, A. Doroshenko, O. Yatsenko, A. Nemish, Merging logic and the coinductive selection monad: mixing machine learning into logical search, in: *ICTERI 2025. Communications in Computer and Information Science* 2763 (2025) 33–46. doi: 10.1007/978-3-032-10477-9_3

Дата першого надходження до видання: 12.03.2026
Внутрішня рецензія отримана: 17.03.2026
Зовнішня рецензія отримана: 18.03.2026
Дата прийняття статті до друку: 19.03.2026
Дата публікації: 16.04.2026

Про авторів:

[1]Шевченко Руслан Сергійович,
кандидат технічних наук
старший науковий співробітник
Shevchenko Ruslan,
Ph.D (technical sciences), senior scientist
https://orcid.org/0000-0002-1554-2019.

[1,2]Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор,
провідний науковий співробітник
Doroshenko Anatoliy,
Ph.D (doctor, physical and mathematical sciences), professor, leading scientist
http://orcid.org/0000-0002-8435-1451.

[1]Яценко Олена Анатоліївна,
кандидат фізико-математичних наук,
старший науковий співробітник
Yatsenko Olena,
Ph.D (physical and mathematical sciences), senior scientist
http://orcid.org/0000-0002-4700-6704.

Місце роботи авторів:

[1] Інститут програмних систем
НАН України,
[1] Institute of Software Systems.
National Academy of Sciences of Ukraine
тел. +38-067-407-32-33
E-mail: doroshenkoanatoliy2@gmail.com,
ruslan@shevchenko.kiev.ua,
oayat@ukr.net
Сайт: https://iss.nas.gov.ua

[2] Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
[2] National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”
Сайт: https://ist.kpi.ua

УДК: 004.658:004.65 https://doi.org/10.15407/pp2026.01.014

Д.В. Зважій , С.С. Гороховський

ВПРОВАДЖЕННЯ ІНДЕКСІВ У POSTGRESQL

У статті досліджено процес проєктування та реалізації нового індексного методу доступу в системі управління базами даних PostgreSQL на прикладі індексу на основі суфіксного дерева. Суфіксні дерева забезпечують оптимальну теоретичну складність пошуку, однак класичні описи цієї структури даних припускають необмежену пам'ять, довільні маніпуляції з вказівниками та однопотоковий доступ — умови, які PostgreSQL не забезпечує. У роботі послідовно розглянуто ключові архітектурні підсистеми PostgreSQL, що безпосередньо впливають на проєктування індексних структур: багатопроцесну модель «один процес на з'єднання», сторінкову організацію зберігання даних із фіксованим розміром блоку 8 КБ, а також дворівневу систему керування пам'яттю на основі спільного пулу буферів та ієрархічних контекстів пам'яті. На прикладі конкретної реалізації проаналізовано ключові проєктні рішення, з якими стикається розробник нового індексу: вибір стратегії відображення логічних вузлів дерева на фізичні сторінки, організацію внутрішньосторінкового зберігання ребер змінної довжини з розділенням на ідентифікаційну та змістову частини, проєктування спеціальної області сторінки для навігаційних метаданих і суфіксних посилань, а також механізми ідентифікації типів сторінок. Окрему увагу приділено відкритим проблемам реалізації — стисненню міток ребер, обмеженням паралелізму алгоритму Укконена та можливостям підтримки index-only scan. Показано, що впровадження нової індексної структури в системі управління базами даних рівня PostgreSQL є не лише задачею адаптації алгоритму до конкретного програмного інтерфейсу, а й узгодженням його з внутрішніми інваріантами, протоколами та логікою взаємодії підсистем, які визначають допустимий простір архітектурних рішень.
Ключові слова: PostgreSQL, індекси, методи доступу, структури даних, керування пам'яттю, узагальнені суфіксні дерева, архітектура баз даних

D. Zvazhii , S. Gorokhovsky

IMPLEMENTING INDEXES IN POSTGRESQL

This article explores the process of designing and implementing a new index access method in the PostgreSQL database management system, using a suffix tree–based index as a case study. Suffix trees provide optimal theoretical search complexity; however, classical descriptions of this data structure assume unlimited memory, arbitrary pointer manipulation, and single-threaded access — conditions that PostgreSQL does not provide. The paper systematically examines the key architectural subsystems of PostgreSQL that directly affect index design: the "one process per connection" multi-process model, page-based storage organization with a fixed 8 KB block size, and the two-level memory management system based on the shared buffer pool and hierarchical memory contexts. Using a concrete implementation as an example, the paper analyzes the key design decisions a developer faces when building a new index: the strategy for mapping logical tree nodes onto physical pages, the organization of intra-page storage for variable-length edges with separation into identification and payload components, the design of the page special area for navigational metadata and suffix links, and mechanisms for page type identification. Particular attention is given to open implementation challenges — edge label compression, parallelism limitations of Ukkonen's algorithm, and the feasibility of supporting index-only scans. It is shown that implementing a new index structure in a DBMS of PostgreSQL's caliber is not merely a matter of adapting an algorithm to a specific programming interface, but also of aligning it with the internal invariants, protocols, and subsystem interaction logic that define the permissible space of architectural decisions.
Keywords: PostgreSQL, indexes, data structures, memory management, generalized suffix trees, database - architecture

Вступ

У сфері систем управління базами даних (СУБД) ефективність доступу до даних є визначальним чинником, що впливає на масштабованість системи та рівень затримок. Із переходом обсягів даних від гігабайтного до петабайтного масштабу обчислювальні витрати, пов'язані з операціями доступу, трансформуються з другорядного аспекту на критичне місце архітектури. Базовим механізмом доступу

до даних у системі PostgreSQL є послідовне сканування (Sequential Scan), яке характеризується часовою складністю $O(n)$, де n — це потужність множини, або таблиці. Попри прийнятність лінійного підходу для малих обсягів даних, у середовищах із високою пропускною здатністю така складність на практиці може виявитися неефективною.

У цих умовах індексні структури стають важливим елементом архітектури СУБД. PostgreSQL вирізняється серед багатьох інших СУБД не лише тим, що має відкритий вихідний код, а також завдяки наявності високорозширюваної індексної інфраструктури. Окрім традиційних B-дерев [1], PostgreSQL низку альтернативних методів доступу, — зокрема, Hash [2], GiST [3], SP-GiST [4][5], GIN [6] і BRIN [7] — кожен з яких орієнтований на обробку специфічних типів даних і семантик запитів. Що більш суттєво, PostgreSQL надає розробникам можливість визначати власні методи доступу до індексів, відкриваючи простір для експериментування з новими структурами даних і адаптації підходів індексування до предметно-орієнтованих задач. Така розширюваність робить PostgreSQL не лише промисловою системою керування базами даних, а й ефективною платформою для проведення наукових досліджень. Попри високу гнучкість, розроблення нового типу індексу для PostgreSQL залишається складним і нетривіальним завданням. Реалізація такого індексу потребує глибокого розуміння внутрішнього програмного інтерфейсу PostgreSQL, моделі конкурентного доступу, механізмів керування буферами, журналювання попереднього запису (write-ahead logging, WAL), а також особливостей взаємодії з планувальником запитів. Рішення, що ухвалюються на рівні індексу, зокрема, щодо організації сторінок, стратегій розбиття або гарантування узгодженості, мають далекосяжні наслідки для коректності, продуктивності та підтримуваності всієї системи. Як наслідок, багато потенційно цінних індексних концепцій залишаються недостатньо дослідженими або реалізуються поза межами СУБД, зокрема, в прикладних пошукових рушіях чи зовнішніх системах зберігання даних.

У попередній статті [8] основну увагу було зосереджено на адаптації специфікації Access Methods API (програмного інтерфейсу методів доступу) [9], яку PostgreSQL надає для імплементації індексу на базі суфіксного дерева. Натомість у цій статті процес розроблення індексів для PostgreSQL розглядається як із концептуальної, так і з практичної перспективи. Замість зосередження винятково на використанні наявних типів індексів аналізується, яким чином у межах розширюваного програмного інтерфейсу методів доступу PostgreSQL (Access Methods API) можна проєктувати та реалізовувати нові індексні структури. Окрему увагу приділено аналізу ключових особливостей архітектури PostgreSQL [10] та їхнього впливу на проєктування індексів. Як практичний приклад розглянуто розробку індексу на базі суфіксного дерева [11].

Аналіз останніх досліджень та публікацій

Як уже було зазначено вище, PostgreSQL надає чітко визначений інтерфейс для створення власних методів доступу до індексів і постачається з шістьома вбудованими типами індексів. Розширюваність цієї системи спирається на дві фундаментальні концепції. Геллерстайн, Нотон і Пфеффер [12] запропонували узагальнене дерево пошуку (GiST), яке об’єднує високо збалансовані індекси, зокрема, B+-дерева та R-дерева, у межах єдиного розширюваного програмного інтерфейсу. Проте GiST не здатне репрезентувати за своєю природою незбалансовані структури, зокрема, префіксні дерева (trie) або суфіксні дерева. Валід і Ільяс [4] запропонували для цього SP-GiST — фреймворк для просторово-роздільних незбалансованих дерев (trie, k-d-дерева, квадродерева). Елтабах, Елтаррас і Ареф [13] реалізували SP-GiST у PostgreSQL і продемонстрували суттєве зростання продуктивності, зокрема, представивши реалізацію суфіксного дерева для пошуку підрядків.

Попри підтримку незбалансованих структур, SP-GiST має критичне обмеження: кожен запис листка відповідає рівно одному ключу та одному ідентифікатору кортежу (TID). Для суфіксних структур це створює проблему, оскільки один рядок s довжини l породжує l суфіксів, які мають посилатися на той самий TID (tuple identifier). Обмеження «один ключ — один TID» унеможливлює повноцінну емуляцію поведінки суфіксного дерева або суфіксного масиву в межах SP-GiST. Нещодавно Шьоманс, Ареф, Зіманьї та Сакр [14] запропонували MGiST і MSP-GiST — розширення GiST і SP-GiST із підтримкою множинних записів, що дозволяють декомпозувати один об'єкт бази даних на кілька індексних записів за допомогою користувацького методу ExtractValue. Хоча ці підходи розроблялися для індексування траєкторій і визнають проблему обмеження «один запис на кортеж», їх досі не застосовано до суфіксних структур.

Що ж до стуктур даних, які зберігають суфікси: суфіксні дерева (Вайнер [15], Маккрейт [16], Укконен [17]) і суфіксні масиви (Манбер і Майєрс [18]) - то вони давно є ґрунтовно дослідженими. Проте практичні реалізації цих структур у межах СУБД залишаються вкрай рідкісними. Сюй, Чен, Хуан, Ху та Нонг [19] запропонували SAES — систему, що замінює інвертований індекс Elasticsearch індексом на основі суфіксного масиву для повнотекстового пошуку в гетерогенних даних; це одна з небагатьох спроб інтегрувати суфіксне індексування в наявну пошукову інфраструктуру.

Таким чином, існує очевидна прогалина: попри значну кількість алгоритмічних розробок, лише небагато структур реалізовано як вбудовані методи доступу до індексів у СУБД, а суфіксні дерева та масиви практично відсутні в промислових рушіях баз даних. Запропонована робота спрямована на заповнення цієї прогалини шляхом реалізації суфіксного індексу як власного методу доступу PostgreSQL та документування проєктних рішень, зумовлених її архітектурними обмеженнями (фіксований розмір сторінки 8 КБ, менеджер буферів, вимоги WAL), які алгоритмічна література здебільшого залишає поза увагою.

Реалізація індексу на базі суфіксного дерева

Задача пошуку підрядків — знаходження всіх рядків, у яких текстовий стовпець містить заданий шаблон, є поширеною в базах даних. Стандартні B-деревні індекси ефективно прискорюють префіксні запити (наприклад, LIKE 'pattern%'), однак не допомагають у випадку ведучих байдужих символів (LIKE '%pattern%'), що змушує систему виконувати послідовне сканування колонки. Суфіксні дерева забезпечують пошук за шаблоном за час $O(m + k)$ [20], де m — довжина шаблону, незалежно від розміру тексту, а k - кількість можливих повторів шаблону. Це робить суфіксні дерева теоретично привабливими для цієї задачі. Втім, класичні описи суфіксних дерев у підручниках припускають необмежену пам'ять, довільні маніпуляції з вказівниками та однопотоковий доступ. PostgreSQL не надає жодної з цих передумов. Натомість він оперує сторінками фіксованого розміру, спільним пулом буферів із суворими протоколами pinning, обов'язковим журналюванням попереднього запису (WAL) і конкурентним доступом з боку кількох бекендів. Інтеграція суфіксного дерева в таке середовище вимагає ухвалення проєктних рішень, які теоретична література зазвичай не розглядає. Цей розділ зосереджений на деталях реалізації індексу на базі суфіксного дерева. Спочатку окреслено архітектурні обмеження PostgreSQL, після чого розглядаються важливі проєктні рішення з якими зіштовхується розробник під час реалізації. Для конкретизації наводяться приклади із конкретної реалізації індексу на базі суфіксного дерева [11].

PostgreSQL зберігає всі дані у сторінках фіксованого розміру — зазвичай 8192 байти (8 КБ), який задається на етапі компіляції параметром BLCKSZ. Це обмеження є всеосяжним: у його межах працюють кожна індексна сторінка,

сторінка heap і записи журналу WAL. Для суфіксного дерева це одразу породжує низку питань. Вузол із великою кількістю вихідних ребер може не вміститися в одну сторінку. Мітка ребра може мати довільну довжину. Розмір сторінки не підлягає зміні — реалізації мають проєктуватися з урахуванням цього обмеження. Фактично доступний для даних простір сторінки є меншим за BLCKSZ. Кожна сторінка містить 24-байтовий заголовок (PageHeaderData) і також зазвичай резервує певний простір наприкінці сторінки для типоспецифічних метаданих (так звана special area). Для сторінки розміром 8 КБ приблизно 8140 байтів доступні для безпосереднього вмісту, а у випадку використання значної спеціальної області — навіть менше. Кожна сторінка PostgreSQL має стандартну внутрішню структуру. Заголовок сторінки містить вказівники (pd_lower, pd_upper), які окреслюють межі між зайнятою та вільною частинами сторінки. Методи доступу можуть зарезервувати спеціальну область (special area) наприкінці сторінки для типоспецифічних метаданих.

Розмір спеціальної області фіксується під час ініціалізації сторінки. Саме тут методи доступу зазвичай зберігають навігаційні вказівники (на батьківські та сусідні сторінки), прапорці типу сторінки та ідентифікаційні маркери. Проєктування цієї структури є одним із ключових ранніх рішень, оскільки її розмір і вміст безпосередньо впливають як на доступний простір для даних, так і на можливості еволюції індексної структури.

Менеджер буферів PostgreSQL опосередковує весь доступ до сторінок. Сторінки ніколи не зчитуються безпосередньо з диска; натомість бекенд-процеси

Лістинг 1: Ініціалізація спеціальної частини сторінки

```
PageInit(page, BLCKSZ,
sizeof(MyOpaqueData));
opaque = (MyOpaque) PageGetSpe-
cialPointer(page);
```

запитують буфери за чітко визначеним протоколом:

- Читання: ReadBuffer(relation, blockno) — фіксує (pin) сторінку в пулі буферів.
- Блокування: LockBuffer(buffer, mode) — захоплює спільне або виключне блокування.
- Доступ: BufferGetPage(buffer) — повертає вказівник на сторінку.
- Модифікація: MarkBufferDirty(buffer) — сигналізує, що сторінку було змінено.
- Звільнення: UnlockReleaseBuffer(buffer) — знімає блокування.

Утримання буфера в стані pinned запобігає витісненню сторінки з пулу, а утримання блокування перешкоджає конкурентним модифікаціям або забезпечує стабільність читання. Критично важливим наслідком є унеможливлення збереження сирих вказівників на сторінках між операціями з буферами: після звільнення буфера будь-який вказівник на його вміст стає недійсним. Для алгоритмів обходу дерев, які концептуально “спускаються” шляхом від кореня до листка, це означає необхідність вибору між двома підходами: або одночасно утримувати кілька буферів у стані pinned (що підвищує ризик взаємних блокувань), або повторно зчитувати сторінки після їхнього звільнення. Обидва варіанти накладають суттєві обмеження на проєктування структур даних і алгоритмів навігації в межах індексів PostgreSQL.

Розподілювач пам’яті PostgreSQL (palloc) працює в межах контекстів пам’яті — ієрархічних арен, які можна скидати або знищувати масово. Код методів доступу зазвичай виконується в контекстах відповідного кортежу (per-tuple contexts), що скидаються після оброблення кожного кортежу.

Із цього випливає важливе правило: не можна зберігати вказівники на пам’ять, виділену через palloc, за межами оброб-

Лістинг 2: Приклад зміни контексту під час індексації кортежу

```
oldCtx = MemoryCon-
textSwitchTo(buildState->tmpMem-
Ctx);
MemoryContextSwitchTo(oldCtx);
MemoryContextReset(buildState-
>tmpMemCtx); PageGetSpe-
cialPointer(page);
```

лення окремого кортежу. Будь-які допоміжні структури даних мають або розміщуватися в контексті з довшим часом життя, або відновлюватися заново для кожного кортежу.

PostgreSQL гарантує надійність збереження даних за допомогою журналювання попереднього запису (write-ahead logging, WAL): усі зміни мають бути зафіксовані в журналі до того, як модифікована сторінка буде записана на диск. Для методів доступу це має два ключові прояви:

- Критичні секції. Модифікації, які повинні бути атомарними, обгортаються в START_CRIT_SECTION() / END_CRIT_SECTION(). Усередині критичної секції будь-яка помилка призводить до PANIC (перезапуску бази даних), а не до подовження неузгодженого стану системи.

- WAL-записи. Складні методи доступу генерують власні типи WAL-записів. Простіші реалізації можуть покладатися на повні зображення сторінок (full-page images) під час початкової побудови індексу, уникаючи детального журналювання кожної окремої зміни.

Для інкрементних вставок коректне журналювання WAL вимагає проєктування форматів записів і реалізації процедур повторного відтворення (redo), що є складним і трудомістким завданням.

PostgreSQL визначає методи доступу до індексів за допомогою структури

Лістинг 3: Ключові функції зворотнього виклику методу доступу

```
amroutine->ambuild =
stree_build;

amroutine->ambuildempty =
stree_empty;

amroutine->aminsert = stree_in-
sert;

amroutine->ambeginscan =
stree_beginscan;

amroutine->amgettuple =
stree_gettuple;

amroutine->amendscan =
stree_endscan;
```

IndexAmRoutine [8], яка задає підтримувані можливості та набір зворотних викликів (callbacks), показаний на лістингу 3. Інтерфейс також декларує можливості методу доступу: чи може індекс забезпечувати унікальність, чи підтримує багатостовпцеві ключі, чи здатен повертати кортежі у впорядкованому вигляді тощо. Для суфіксного дерева, орієнтованого на пошук підрядків, більшість із цих можливостей відсутня: індекс не забезпечує унікальність, не підтримує впорядкування результатів і працює лише з одним текстовим стовпцем.

Під час проєктування індексу виникає необхідність зберігання різних структур, тому індекс може містити сторінки різних типів — сторінки метаданих, вузли дерева, сторінки переповнення для великих записів, а також сторінки для відстеження вільного простору. Під час читання сторінки реалізація методу доступу повинна однозначно визначити її тип. Крім того, діагностичні інструменти на кшталт pg_filedump отримують істотну користь від можливості ідентифікувати тип індексної сторінки без знання прикладного контексту. Для розв’язання цієї задачі застосовуються різні підходи:

- фіксовані номери блоків — окремі номери блоків мають фіксоване призначення (наприклад, мета-сторінка в блоці 0);

- використання відведених констант, записаних у визначене місце сторінки;

- збереження міток типу сторінки у спеціальній області сторінки (special area).

Схема з фіксованими номерами блоків може спрацювати, якщо є можливість попередньо зарезервувати під них пам'ять, але непридатна для сторінок, які будуть виділені динамічно під час вставок. Числа-константи забезпечують надійну ідентифікацію, проте споживають простір і потребують координації, щоб уникнути колізій між різними методами доступу. У практиці PostgreSQL зазвичай поєднують компактні мітки типу сторінки з фіксованим ідентифікатором методу доступу, розміщеним у спеціальній області сторінки (наприклад, streePageId = 0xCDEE). Такий підхід дає змогу розрізняти типи сторінок як усередині індексу, так і між різними індексними структурами, забезпечуючи зручність під час налагодження та валідації сторінок під час обходу — за незначної додаткової вартості простору, оскільки спеціальна область зазвичай уже містить навігаційні метадані.

Якщо говорити про стуктури, які повинен зберігати індекс, то можна виділити такі складові дерева як вузли, ребра та ідентифікатори кортежів (TID - tuple identifier), які буде повертати індекс. Ці типи даних суттєво відрізняються за розмірами, шаблонами доступу та характером зростання, що створює нетривіальну задачу відображення їх на сторінки фіксованого розміру PostgreSQL так, щоб забезпечити ефективну побудову індексу та швидкий пошук. Розміри вузлів суфіксного дерева можуть істотно варіюватися. Вузол поблизу кореня може мати вихідні ребра для сотень різних символів, тоді як вузол глибоко в дереві часто має лише одне ребро. Це породжує питання відображення: як

Лістинг 4: Приклад використання міток сторінки

```
#define STREE_META
(1<<0)
#define STREE_ROOT
(1<<1)
#define STREE_INTERNAL_NODE
(1<<2)
#define STREE_DATA_PAGE
(1<<3)

typedef struct STreeNodePage-
OpaqueData {
    uint16 flags;
    /* ... інші поля ... */
} STreeNodePageOpaqueData;
```

співвіднести логічні вузли дерева для зберігання на фізичних сторінках. Можна виділити три основні підходи:

- Один вузол на сторінку: кожен вузол дерева займає рівно одну сторінку розміром 8 КБ, а номер блоку (BlockNumber) слугує його ідентифікатором — без потреби в додатковій адресації.

- Кілька вузлів на сторінці: дрібні вузли пакуються разом для зменшення втрат простору. Це вимагає внутрішньосторінкової адресації (номер блоку + зсув) і ускладнює логіку росту вузлів.

- Вузли, що розтягуються на кілька сторінок: великі вузли можуть мати сторінки-продовження, що дозволяє підтримувати довільні розміри, але додає складності під час обходу й синхронізації.

Модель "один вузол — одна сторінка" природно узгоджується з менеджером буферів PostgreSQL. Закріплення (pinning) вузла означає фіксацію рівно одного буфера, а його звільнення — звільнення того самого буфера. Відсутні часткові блокування вузла, міжсторінкова узгодженість і складна адресна арифметика. Основний недолік цього підходу — неефективне використання простору: листковий вузол з

Лістинг 5: Представлення ребра

```
typedef struct EdgeId {
    pg_wchar  firstChar;
    uint16    payloadOffset;
    uint16    payloadLength;
} EdgeId;

typedef struct EdgePayload {
    BlockNumber destination;
    uint16      labelLength;
    char        label[];
} EdgePayload;
```

одним ребром і мінімальними метаданими може займати лише близько 100 байтів із 8 КБ, що означає до 98% втрат. Для дерева з мільйонами дрібних вузлів це може становити значні накладні витрати. Водночас суфіксні дерева мають корисну властивість: кількість внутрішніх вузлів обмежена сумарною довжиною індексованих рядків. Дерево, що індексує N символів, містить не більше ніж N внутрішніх вузлів. Отже, просторові накладні витрати, хоч і відчутні, зростають лінійно — $O(n)$ сторінок — і не є неконтрольованими. З огляду на це, для початкової реалізації підхід "один вузол на сторінку" пропонує переконливий баланс між простотою та коректністю. У власній реалізації [11] був використаний розширений варіант, який дозволяє вузлу розростатися на нову сторінку, якщо було використано увесь вільний простір. Оптимізації на кшталт пакування дрібних вузлів доцільно відкласти до наступної ітерації — коли стане зрозуміло, що саме використання простору, а не складність алгоритмів, є головним вузьким місцем.

Вузол суфіксного дерева також зберігає набір вихідних ребер, кожне з яких складається з мітки змінної довжини та вказівника на цільовий вузол. У процесі побудови суфіксного дерева алгоритм Укконена [17] багаторазово виконує три базові операції над ребрами:

- пошук ребра, що починається з заданого символу;
- вставка нового ребра з певною міткою та призначенням;
- поділ наявного ребра, коли його мітка скорочується на позиції k та перенаправляється до нового внутрішнього вузла.

Дизайн сторінки індексу має ефективно підтримувати всі ці операції, оскільки кожна з них по-різному модифікує вміст сторінки, а обсяг змін безпосередньо впливає на розміри WAL-записів і частоту позначення буферів як "брудних". Найчастішою операцією є пошук: на кожному кроці оброблення вхідного рядка алгоритм перевіряє, чи існує ребро з відповідним початковим символом. Це вимагає організації ребер у формі, зручній для пошуку — наприклад, шляхом зберігання у відсортованому вигляді з можливістю бінарного пошуку або за допомогою хешування. Вставка додає нові дані до сторінки: якщо ребра зберігаються компактно й упорядковано, вставка потребує зсуву всіх наступних елементів. У схемах з апендиксним додаванням корисних даних і окремим індексом вставка самих даних має амортизовану складність $O(1)$, а підтримка впорядкування — $O(n)$ за кількістю ребер. Ключовою операцією, яка впливає на дизайн структури сторінки, є операція поділу ребра. Під час розщеплення ребра e на позиції k його мітка, наприклад, ABCDEF, перетворюється на префікс AB, створюється новий внутрішній вузол із ребром CDEF, а призначення початкового ребра змінюється на цей новий вузол. Вирішальним є питання: чи можна скоротити мітку на місці, не переміщуючи байти в пам'яті, адже переміщення даних істотно збільшує обсяг модифікацій сторінки та відповідне WAL-журналювання. Ефективним розв'язанням цієї проблеми є відокремлення структури ребра, необхідної для пошуку (наприклад, першого символу), від його корисного навантаження — повної мітки та вказівника призначення.

Це дозволяє організувати сторінку, яка зростатиме у два напрямки (Рис.1):

- масив ідентифікаторів ребер зростає від заголовка сторінки, зберігаючи впорядкованість за першим символом;

- корисні дані ребер додаються апендиксом з боку спеціальної області сторінки, тобто у протилежному напрямку.

За такої організації вставка нового ребра з початковим символом c виконується через бінарний пошук позиції, зсув елементів масиву ідентифікаторів, запис нового ідентифікатора та додавання корисних даних у верхній частині вільного простору.

Архітектура сторінки, що дозволяє рости в обидва напрямки, розв'язує задачу зберігання ребер, однак вузол суфіксного дерева містить значно більше, ніж лише ребра. Під час побудови та використання дерева алгоритм повинен переміщатися між вузлами різними шляхами:

- переходити за суфіксними посиланнями під час побудови за Укконеном,

- підніматися до батьківських вузлів у певних сценаріях відновлення або перевірки коректности,

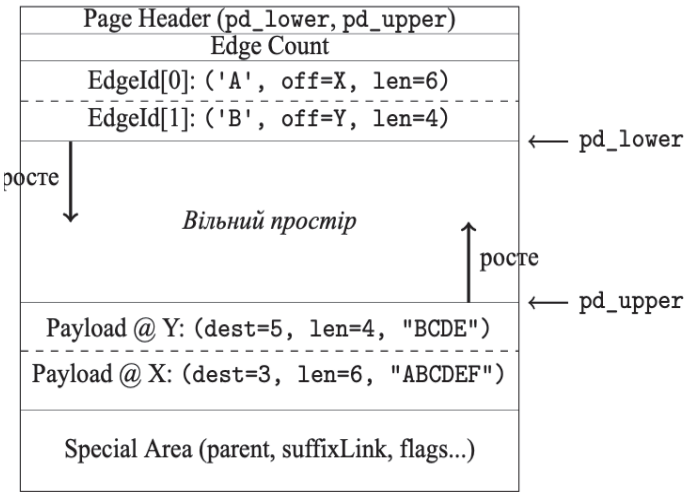

Рис. 1. Дизайн сторінки, яка зберігає ребра

- ітеруватися по сторінках даних під час збирання ідентифікаторів кортежів (TID).

PostgreSQL резервує наприкінці кожної сторінки простір для даних, специфічних для методу доступу, — до якого звертаються через PageGetSpecialPointer(). Для вузлів суфіксного дерева ця спеціальна область містить як навігаційні вказівники, так і ідентифікаційну інформацію про тип сторінки. Зокрема, тут зберігаються:

- **parentNode** - забезпечує можливість руху вгору по ієрархії. Хоча алгоритм Укконена здебільшого виконує обхід дерева від кореня до листків, у деяких ситуаціях, наприклад, під час відновлення або перевірки цілісності структури, потрібен підйом до батьківського вузла. Явне зберігання посилання на батька дозволяє уникнути підтримки окремого стеку обходу.

- **suffixLink** - ключове поле для досягнення лінійної складності O(n) під час побудови дерева за алгоритмом Укконена [17]. Після створення або відвідування внутрішнього вузла, що відповідає суфіксу $S[i..j]$, це посилання вказує на вузол для $S[i+1..j]$. Такий перехід між спорідненими суфіксами виконується за сталий час. Без механізму суфіксних посилань складність побудови зростала б до $O(n^2)$.

- **itemPointersStart** - вказує на першу сторінку в ланцюжку сторінок, що зберігають TID. Вузол, який відповідає поширеному підрядку, може відповідати тисячам рядків у heap; ідентифікатори кортежів не зберігаються у вузлі разом із ребрами, а виносяться в окремі пов'язані сторінки, щоб не перевантажувати основну структуру вузла.

- **prevSiblingNode** та **nextSiblingNode** - формують двобічно зв'язаний список сторінок переповнення для TID та ребер. Коли сторінка даних заповнюється, виділяється нова й приєднується до ланцюжка через ці вказівники.

- **streePageId** - число-константа (наприклад, 0xCDEE), що однозначно ідентифікує сторінку як сторінку суфіксного дерева. Це дозволяє при відлагодженні розпізнавати тип сторінки без потреби інтерпретувати її внутрішню структуру.

- **flags** - містить прапорці, що визначають роль сторінки: коренева, внутрішня, сторінка даних (TID). Це дає змогу застосовувати спеціалізовану логіку під час обходу та оброблення індексу.

Попри отримане рішення 11, низка проєктних аспектів залишається складною й потребує додаткового осмислення. Однією з найбільш проблемних зон є взаємодія з механізмом VACUUM і оброблення видалень. Коли рядок у heap видаляється, його TID необхідно вилучити з усіх вузлів індексу, де він зберігається. Це означає або повне сканування структури індексу, або підтримку зворотних посилань від TID до відповідних вузлів. Обидва підходи мають суттєві недоліки: перший пов'язаний із високими обчислювальними витратами, другий — ускладнює структуру індексу та збільшує накладні витрати на зберігання й супровід додаткових метаданих. Іншим складним напрямом є стиснення. Мітки ребер у суфіксному дереві часто мають спільні префікси, оскільки вони є суфіксами тих самих або подібних рядків. Теоретично можливо застосовувати словникове стиснення або схеми на основі посилань для зменшення обсягу зберігання. Однак такі оптимізації ускладнюють логіку вставки, розщеплення ребер і журналювання змін у WAL, що своєю чергою, збільшує складність реалізації та підвищує ризик помилок. Питання паралелізму також є нетривіальним. Хоча PostgreSQL підтримує паралельну побудову індексів, класичний алгоритм Укконена по суті є послідовним — через залежності, пов'язані з підтримкою суфіксних посилань. Це обмежує можливість прямого використання механізмів паралельного виконання. Альтернативні підходи, зокрема, алгоритми паралельної побудови суфіксних масивів, потенційно могли б забезпечити кращу масштабованість, однак потребують докорінно іншої організації індексної структури. Нарешті, слід враховувати обмеження механізму index-only scan. У PostgreSQL цей режим дає змогу уникати звернень до heap, якщо індекс містить усі необхідні для запиту стовпці. У випадку суфіксного дерева, яке індексує лише текстовий стовпець, повернення самого тексту без звернення до heap вимагало б зберігання його в самому індексі, що ускладнює завдання зменшення обсягу структури.

Лістинг 6: Структура спеціального простору сторінки

```
typedef struct STreeNodePage-
OpaqueData {
    BlockNumber parentNode;
    BlockNumber suffixLink;
    BlockNumber
    BlockNumber prevSiblingNode;
    BlockNumber nextSiblingNode;
    uint16      streePageId;
    uint16      flags;
} STreeNodePageOpaqueData;
```

Висновки

Реалізація суфіксного дерева як індексної структури в PostgreSQL передбачає інтеграцію теоретично елегантної моделі в складну, багаторівневу архітектуру промислової СУБД. У цьому процесі кожне проєктне рішення, пов'язане з необхідністю балансувати між просторовою ефективністю, часовими характеристиками операцій, складністю реалізації

та вимогами до транзакційної надійності й відновлення після збоїв.

Проаналізовані питання — ідентифікація сторінок, організація внутрішньої архітектури сторінки, відображення логічних вузлів на фізичні сторінки, зберігання міток, розміщення TID, взаємодія з буферним менеджером репрезентують лише частину спектру рішень, які необхідно ухвалити. Повноцінна реалізація неминуче потребує десятків додаткових виборів на більш дрібному рівні, кожен з яких впливає на загальну поведінку системи.

Водночас кожна система рівня складності сучасної СУБД фактично формує власний внутрішній програмний каркас — набір інтерфейсів, інваріантів і протоколів взаємодії, які визначають, як саме окремі компоненти можуть співпрацювати. Цей каркас не є нейтральним: він задає допустимі способи розширення, обмежує архітектурні рішення та формує характер компромісів, які доводиться приймати розробникові. Тому впровадження нової індексної структури — це не лише адаптація алгоритму до конкретного програмного інтерфейсу, а й узгодження його з глибинною логікою взаємодії підсистем.

Література

1. Comer D. Ubiquitous B-Tree. *ACM Comput. Surv.* 11. 1979. Vol. 11., no.2. P.121–37. URL: https://doi.org/10.1145/356770.356776. (date of access: 08.05.2025)
2. Hash Indexes. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/17/hash-index.html. (date of access: 08.05.2025)
3. GiST Indexes. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/17/gist.html. (date of access: 08.05.2025)
4. Walid A.G., Ilyas I. F. SP-GiST: An Extensible Database Index for Supporting Space Partitioning Trees. *Journal of Intelligent Information Systems*. 2001. Vol. 11., no.2. P.215–40. URL: https://doi.org/10.1023/A:1012809914301. (date of access: 25.01.2026)
5. SP-GiST Indexes. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/17/spgist.html. (date of access: 08.05.2025)
6. GIN Indexes. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/17/gin.html. (date of access: 08.05.2025)
7. BRIN Indexes. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/17/brin.html. (date of access: 08.05.2025)
8. Зважій Д. Особливості Індексації у PostgreSQL. *Наукові Записки НаУКМА. Комп'ютерні Науки.* 2025. *№*. 8. С. 113–17. DOI: 10.18523/2617-3808.2025.8.113-117. (дата звернення: 25.01.2026)
9. Basic API Structure for Indexes. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/17/index-api.html. (date of access: 08.05.2025)
10. Stonebraker M., Rowe L. A. The design of POSTGRES. *SIGMOD.* 1986. Vol. 15., no. 2 P. 340—355. DOI: 10.1145/16856.16888. URL: https://dl.acm.org/doi/10.1145/16856.16888. (дата зверн. 25.01.2026)
11. Suffix Tree Based AM Index Method Source Code. URL: https://github.com/Uaman/postgres/tree/feature-string-search-am. (date of access: 10.02.2026)
12. Hellerstein J. M., Naughton J. F., Pfeffer A. Generalized Search Trees for Database Systems. *Encyclopedia of Database Systems* Morgan Kaufmann Publishers Inc. 1998. P.1-3. URL: https://doi.org/10.1007/978-1-4899-7993-3_743-2 (date of access: 15.02.2026)
13. Eltabakh M., Ramy E., Walid A. Space-Partitioning Trees in PostgreSQL: Realization and Performance. 2006. URL: https://doi.org/10.1109/ICDE.2006.146. (date of access: 08.05.2025)
14. Schoemans M., Walid G. A., Zimányi E., Sakr M. Multi-Entry Generalized Search Trees for Indexing Trajectories. *Proceedings of the 32nd ACM International Conference on Advances in Geographic Information Systems. 2024.* P.21–31. URL: https://doi.org/10.1145/3678717.3691320. (date of access: 15.02.2026)
15. Weiner P. Linear Pattern Matching Algorithm. 1973. URL: https://doi.org/10.1109/SWAT.1973.13. (date of access: 10.02.2026)

16. McCreight E. M. A Space-Economical Suffix Tree Construction Algorithm. *J. ACM* . 1976. Vol.23., no.2. P.262–72. URL: https://doi.org/10.1145/321941.321946. (date of access: 15.02.2026)
17. Ukkonen E.. On-Line Construction of Suffix Trees. *Algorithmica.* 1995. Vol.14., no.3 P.249–60. URL: https://doi.org/10.1007/BF01206331. (date of access: 10.02.2026)
18. Manber U., Gene M. Suffix Arrays: A New Method for on-Line String Searches. *SIAM Journal on Computing*. 1993. Vol. 22., no. 5: P.935–48. URL: https://doi.org/10.1137/0222058. (date of access: 15.02.2026)
19. Xu W., Haoyu C., Yidong H., Xuedong H., Ge N. Full-Text Search Engine with Suffix Index for Massive Heterogeneous Data. *Information Systems.* 2022. URL: https://doi.org/10.1016/j.is.2021.101893. (date of access: 10.02.2026)
20. Gusfield D. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. *Cambridge University Press*. 1997. URL: https://doi.org/10.1017/CBO9780511574931. (date of access: 08.05.2025)

Дата першого надходження до видання: 17.02.2026
Внутрішня рецензія отримана: 25.02.2026
Зовнішня рецензія отримана: 25.02.2026
Дата прийняття статті до друку: 19.03.2026
Дата публікації: 16.04.2026

Про авторів:

Зважій Дмитро Володимирович,
аспірант,
старший викладач кафедри мережних технологій факультету інформатики,
Zvazhii Dmytro,
Post-graduate student, senior tutor
https://orcid.org/0000-0003-1705-3590
E-mail: d.zvazhii@ukma.edu.ua

Гороховський Семен Самуїлович,
кандидат фізико-математичних наук,
доцент кафедри інформатики факультету інформатики,
Gorokhovsky Semen,
Ph.D (physical and mathematical sciences),
associate professor
E-mail: gor@ukma.edu.ua

Місце роботи авторів:

Національний університет
«Києво-Могилянська академія»,
04655, Україна, Київ,
вул. Григорія Сковороди 2,
National University
“Kyiv–Mohyla Academy”,
faculty of informatics
E-mail: fin@ukma.edu.ua

УДК 004.94:628.4:504.064 https://doi.org/10.15407/pp2026.01.025

А. М. Тригуба, Н. Я. Коваль, І. Л. Тригуба, І. Р. Фірман, В. Ю. Фамуляк

АРХІТЕКТУРА ПРОГРАМНОЇ СИСТЕМИ БАГАТОМОДЕЛЬНОГО ПРОГНОЗУВАННЯ ЕКОЛОГІЧНИХ РИЗИКІВ ДЛЯ ЦИФРОВОГО УПРАВЛІННЯ МУНІЦИПАЛЬНИМИ ОРГАНІЧНИМИ ВІДХОДАМИ

У статті досліджується проблема проєктування програмної системи багатомодельного прогнозування екологічних ризиків для цифрового управління муніципальними органічними відходами в умовах зростання обсягів значущих даних і потреби оперативного ухвалення рішень на локальному рівні. Проведено аналіз сучасних підходів до прогнозування екологічних ризиків у системах муніципального управління. Встановлено, що використання окремих моделей не забезпечує достатньої стійкості результатів за умов неоднорідності вхідних даних і нелінійності факторів впливу. Обґрунтовано доцільність побудови програмної системи на основі багаторівневої архітектури, яка включає модулі підготовки даних, прогнозного аналізу, оцінювання моделей, інтерпретації ознак і підтримки цифрових управлінських рішень. Запропоновано багатомодельний алгоритм, до складу якого включено Random Forest, Gradient Boosting та XGBoost, що дозволяє автоматично визначати найефективнішу модель за критеріями MAE, RMSE і R^2. Розроблено UML-архітектуру взаємодії класів DataManager, RiskModelEngine, BenchmarkController і DecisionSupport. Для пояснення результатів використано метод перестановочного оцінювання важливості ознак, що дозволило визначити домінування просторових та інфраструктурних факторів у формуванні ризику. Результати експериментального тестування показали, що найкращу точність забезпечує модель XGBoost (MAE=0.039, RMSE=0.055, R^2=0.930), що підтверджує ефективність запропонованого підходу.

Ключові слова: екологічний ризик, багатомодельне прогнозування, муніципальні органічні відходи, машинне навчання, Random Forest, Gradient Boosting, XGBoost, UML-архітектура, permutation importance, цифрове управління, інформаційна система, підтримка ухвалення рішень.

A.M. Tryhuba, N.Ya. Koval, I.L. Tryhuba, I.R. Firman, V.Yu. Famuliak

ARCHITECTURE OF A SOFTWARE SYSTEM FOR MULTI-MODEL ECOLOGICAL RISK FORECASTING FOR DIGITAL MANAGEMENT OF MUNICIPAL ORGANIC WASTE

The article investigates the problem of designing a software system for multi-model ecological risk forecasting for digital management of municipal organic waste under conditions of increasing volumes of significant data and the need for prompt decision-making at the local level. An analysis of modern approaches to ecological risk forecasting in municipal management systems has been carried out. It has been concluded that the use of individual models does not ensure sufficient stability of results under conditions of heterogeneous input data and nonlinear influence factors. The expediency of constructing a software system based on a multi-level architecture has been substantiated, including modules for data preparation, predictive analysis, model evaluation, feature interpretation, and digital decision support. A multi-model algorithm has been proposed, incorporating Random Forest, Gradient Boosting, and XGBoost, which makes it possible to automatically determine the most effective model according to the criteria MAE, RMSE, and R^2. A UML architecture of interaction between the classes DataManager, RiskModelEngine, BenchmarkController, and DecisionSupport has been developed. To explain the results, the permutation feature importance method has been applied, which made it possible to determine the dominance of spatial and infrastructural factors in risk formation. Experimental testing results showed that the highest accuracy is provided by the XGBoost model (MAE=0.039, RMSE=0.055, R^2=0.930), confirming the effectiveness of the proposed approach.

Keywords: ecological risk, multi-model forecasting, municipal organic waste, machine learning, Random Forest, Gradient Boosting, XGBoost, UML architecture, permutation importance, digital management, information system, decision support.

Вступ

Цифрова трансформація муніципального управління відходами поступово переходить від рівня електронного обліку до використання інтелектуальних систем, здатних прогнозувати екологічні наслідки функціонування міської інфраструктури в умовах зростаючого антропогенного навантаження. Особливо це стосується органічних відходів, які у структурі міських потоків формують найбільш динамічну частину біологічно активної маси, що супроводжується утворенням метану, зміною температурних режимів, накопиченням фільтрату та підвищенням ризику локального екологічного перевантаження. У таких умовах управлінські рішення повинні ґрунтуватися не лише на фактичних показниках утворення відходів, а й на прогнозі розвитку екологічних ризиків у короткота середньостроковій перспективі [1].

Формування цифрових систем управління муніципальними органічними відходами ускладняється тим, що екологічний ризик має багатофакторну природу і виникає під впливом одночасної дії демографічних, просторових, інфраструктурних, сезонних і технологічних чинників. Підвищення щільності забудови, нерівномірність логістичних потоків, зміна складу органічної фракції та перевантаження переробних потужностей створюють нелінійні залежності, які адекватно не описуються традиційними аналітичними моделями. Саме тому сучасні цифрові платформи управління відходами потребують програмних архітектур, здатних інтегрувати кілька моделей прогнозування, порівнювати їхню ефективність та формувати адаптивні сценарії підтримки управлінських рішень [2].

Складність задачі полягає також у тому, що екологічні процеси в системах органічного поводження з відходами характеризуються різною швидкістю розвитку, неоднорідністю просторового розподілу та високою залежністю від зовнішніх факторів середовища. Наприклад, збільшення частки урбанізованої території одночасно впливає на обсяги накопичення органічної маси, інтенсивність транспортного навантаження та рівень локальних викидів, що унеможливлює використання лише одного прогнозного алгоритму для всіх типів територій. За таких умов багатомодельний підхід стає не просто альтернативою, а необхідною архітектурною основою цифрової системи прогнозування [3].

У сучасних умовах екологічного моніторингу особливої ваги набуває здатність програмної системи працювати з різнорідними даними: статистичними показниками, геопросторовою інформацією, часовими рядами, результатами сенсорного моніторингу та інфраструктурними характеристиками територій. Це вимагає багаторівневої архітектури, в якій окремі модулі виконують функції збору, нормалізації, моделювання, оцінювання та інтерпретації результатів [4].

Стан предметної області

Упродовж останніх років у міжнародній практиці спостерігається активне впровадження підходів, орієнтованих на даних у сфері поводження з муніципальними відходами, де прогнозування виконується на основі алгоритмів машинного навчання, просторової аналітики та цифрових двійників міської інфраструктури [5]. Водночас більшість досліджень орієнтована або на прогнозування кількості відходів, або на оптимізацію маршрутів перевезення, тоді як екологічний ризик як інтегральний показник розглядається значно рідше.

Проведений аналіз публікацій показав, що сучасні системи прогнозування екологічного ризику використовують переважно окремі алгоритми, серед яких найчастіше застосовуються Random Forest, Gradient Boosting, Support Vector Regression та XGBoost [6]. Однак у більшості випадків відсутня програмна архітектура, яка дозволяє виконувати системний бенчмаркінг моделей, автоматично оцінювати якість прогнозу та забезпечувати адаптивний вибір алгоритму залежно від структури вхідних даних.

Виявлено, що існуючі цифрові рішення не забезпечують повної інтеграції

просторових, інфраструктурних та часових змінних у межах єдиної програмної системи. Частина платформ працює лише з табличними даними, інші – лише з GIS-компонентами, а модулі пояснюваності моделей часто взагалі відсутні [7].

Аналіз сучасних підходів

Сучасні технології прогнозування екологічних процесів базуються на кількох групах методів. Першу групу становлять класичні статистичні підходи, зокрема, регресійні моделі та часові ряди, які дозволяють оцінювати середні тенденції, однак мають обмежену здатність описувати складні нелінійні взаємодії [8-10].

Другу групу формують ансамблеві алгоритми машинного навчання, які забезпечують кращу стійкість до шуму даних і здатні працювати з високою розмірністю ознак. Зокрема, Random Forest добре виявляє себе у задачах екологічного ризику через здатність до автоматичного виявлення значущих взаємодій між змінними [11-13].

Третя група пов’язана з використанням пояснювального штучного інтелекту, де поряд із прогнозом формується пояснення впливу кожної ознаки на результат моделі. Це особливо важливо для муніципального управління, де рішення мають бути інтерпретованими для органів місцевого самоврядування [14-15].

Виявлені протиріччя. Аналіз літератури дозволив виявити три основні протиріччя. Перше протиріччя виникає між багатофакторністю екологічного ризику та використанням у більшості систем лише одного алгоритму прогнозування. Друге протиріччя виникає між динамічністю екологічних процесів та статичністю існуючих програмних архітектур, які не підтримують автоматичне оновлення моделей. Третє протиріччя виявляється між високою точністю сучасних ансамблевих моделей і недостатнім рівнем їх інтерпретованості у прикладних муніципальних інформаційних системах.

Мета статті полягає у розробленні архітектури програмної системи багатомодельного прогнозування екологічних ризиків для цифрового управління муніципальними органічними відходами на основі інтеграції ансамблевих методів машинного навчання, модулів оцінювання якості прогнозу та засобів інтерпретації результатів.

Виклад основного матеріалу дослідження

Формування архітектури програмної системи багатомодельного прогнозування екологічних ризиків. Цей етап є базовим. Він забезпечує побудову інтелектуального інструментарію, здатного забезпечувати адаптивне аналітичне оцінювання стану екологічних систем в умовах динамічної зміни зовнішнього середовища. Особливість такої програмної системи полягає у необхідності одночасного використання кількох математичних моделей прогнозування, що функціонують паралельно, забезпечуючи підвищення точності оцінювання ризиків за рахунок комбінування результатів різних алгоритмічних підходів.

У сучасних інформаційних системах екологічного аналізу архітектурна побудова має ґрунтуватися на принципі багатошаровості, який забезпечує незалежність функціональних компонентів, масштабованість, можливість модифікації окремих модулів без порушення роботи всієї системи, а також адаптацію до нових джерел даних і нових моделей прогнозування. У запропонованій структурі архітектури система поділяється на чотири основні функціональні рівні: 1) рівень збору та підготовки даних; 2) аналітичний рівень; 3) рівень бізнес-логіки; 4) рівень представлення результатів.

Перший рівень формують джерела первинної інформації, які включають результати сенсорного моніторингу, геопросторові набори даних, статистичну екологічну інформацію, метеорологічні параметри, характеристики техногенного навантаження території, а також часові ряди спостережень. Формально множину вхідних даних системи доцільно подати як:

$$D = \{(x_i, y_i)\}_{i=1}^{N}, \quad (1)$$

де x_i – вектор факторних ознак, що характеризують екологічний стан території; y_i – цільове значення показника ризику; N – кількість спостережень у навчальній вибірці.

Перед надходженням до аналітичного рівня дані проходять процедури очищення, нормалізації, усунення пропусків та приведення до єдиного масштабу. Нормалізація ознак здійснюється за виразом:

$$x_i^{norm} = \frac{x_i - x_{min}}{x_{max} - x_{min}}, \quad (2)$$

де x_i – поточне значення ознаки; x_{min} – мінімальне значення ознаки у вибірці; x_{max} – максимальне значення ознаки у вибірці.

Нормалізація ознак дозволяє забезпечити коректність навчання різнорідних моделей.

Другий рівень архітектури становить аналітичне ядро системи, в якому функціонують незалежні програмні модулі для прогнозування. Кожен модуль реалізує окрему модель варіативності – нейромережеву, ансамблеву, регресійну або статистичну. Відповідно до принципу багатомодельності результати окремих моделей об'єднуються у єдиний інтегрований прогноз рівня екологічного ризику шляхом агрегування виходів окремих прогнозних модулів:

$$R(t) = \sum_{k=1}^{M} w_k f_k(x_t), \quad (3)$$

де $f_k(x_t)$ – результат *k*-ої моделі прогнозування рівня екологічного ризику; w_k – ваговий коефіцієнт моделі; M – кількість моделей у системі.

Використання кількох моделей одночасно дозволяє мінімізувати похибку окремих алгоритмів та підвищити стійкість прогнозу рівня екологічного ризику до нестабільності даних.

У межах аналітичного рівня особливу роль виконує модуль аналізу поведінкової інформації, який акумулює результати функціонування окремих моделей і виконує оцінювання їхньої адаптивності до поточних змін проєктного середовища. Саме цей модуль забезпечує перехід від статичної моделі прогнозування до динамічної архітектури, в якій система здатна змінювати ваги моделей залежно від поточної ефективності.

Третій рівень формує бізнес-логіку системи. Тут реалізуються правила інтерпретації прогнозних результатів, порогові значення небезпеки, механізми генерації попереджень та сценарії ухвалення рішень. Для цього вводиться функція ризикової інтерпретації:

$$Z = \begin{cases} 1, & R(t) \geq R_{crit} \\ 0, & R(t) < R_{crit} \end{cases}, \quad (4)$$

де R_{crit} – порогове значення ризику.

Якщо прогнозований рівень ризику перевищує порогове значення, система формує сигнал для управлінського реагування.

Четвертий рівень архітектури забезпечує представлення результатів користувачу. Він включає інтерфейс візуалізації, аналітичні панелі, графіки часової динаміки, карти ризиків і таблиці показників. Саме цей рівень забезпечує інтеграцію системи з кінцевим користувачем і дозволяє застосовувати результати прогнозування у практичній діяльності.

Запропонована архітектура програмної системи побудована за принципом модульної декомпозиції, що дозволяє незалежно розширювати кількість моделей прогнозування без перебудови всієї системи. Кожен новий програмний модуль інтегрується у загальну архітектуру через уніфікований інтерфейс передачі функціональних потоків даних.

На відміну від класичних однотипних систем прогнозування, багатомодельна архітектура забезпечує одночасне врахування різних закономірностей у структурі екологічних даних, що особливо важливо в умовах нелінійності екологічних процесів, сезонної варіативності та багатофакторності ризиків.

Багатомодельний алгоритм прогнозування екологічного ризику. Основною особливістю запропонованої програмної системи є реалізація багатомодельного алгоритму прогнозування екологічного ризику, який ґрунтується на паралельному використанні кількох ансамблевих методів машинного навчання. Такий підхід зумовлений тим, що екологічні ризики у системах управління муніципальними органічними відходами формуються під впливом великої кількості неоднорідних факторів, між якими існують складні нелінійні зв'язки. За цих умов використання лише одного алгоритму не завжди забезпечує достатню стійкість прогнозу, тоді як порівняльне застосування кількох моделей дає змогу виявити ту архітектуру, яка найкраще відображає структуру даних і забезпечує найменшу похибку.

У межах дослідження до складу багатомодельного прогнозного контуру включено три ансамблеві алгоритми: Random Forest, Gradient Boosting та XGBoost. Їх вибір обумовлений тим, що всі три моделі добре працюють із багатовимірними наборами даних, здатні враховувати нелінійність взаємозв'язків між ознаками та цільовою змінною, а також демонструють високу ефективність у задачах регресійного прогнозування зі складною внутрішньою структурою факторів.

У загальному вигляді множину моделей, що входять до складу системи, представлено виразом:

$$M = \{M_{RF}, M_{GB}, M_{XGB}\}\ , \qquad (5)$$

де M_{RF} – модель Random Forest; M_{GB} – модель Gradient Boosting; M_{XGB} – модель XGBoost.

Для кожної моделі прогнозне значення інтегрального екологічного ризику визначається залежністю:

$$\hat{y}_i^{(k)} = M_k(x_i)\ , \qquad (6)$$

де x_i – вектор ознак для i-го об'єкта спостереження; M_k – k-та модель із множини M; $\hat{y}_i^{(k)}$ – прогнозоване значення ризику.

Алгоритм Random Forest належить до методів bagging і формує підсумковий прогноз шляхом усереднення результатів великої кількості дерев рішень, побудова-

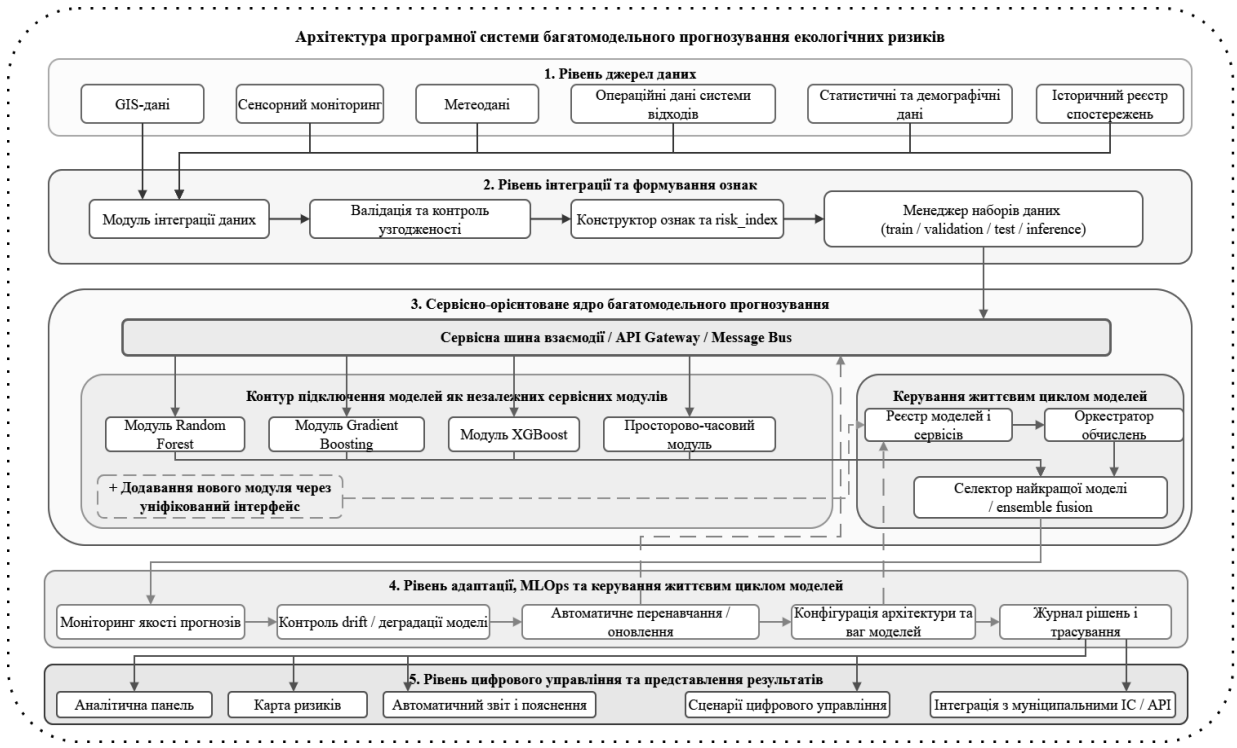

Рис. 1. Схема архітектури програмної системи багатомодельного прогнозування

них на випадкових підвибірках даних та випадкових підмножинах ознак. Завдяки цьому знижується дисперсія прогнозу, підвищується стійкість моделі до шуму та зменшується ризик перенавчання. У задачах екологічного прогнозування така модель є особливо корисною тоді, коли фактори ризику мають складну структуру, а частина ознак є слабоформалізованими або корельованими між собою.

Модель Gradient Boosting реалізує іншу логіку побудови ансамблю. На відміну від Random Forest, у цій архітектурі дерева формуються послідовно, причому кожне наступне дерево намагається компенсувати помилки попередніх. Такий механізм дозволяє краще вловлювати тонкі нелінійні закономірності у даних та забезпечує вищу чутливість до складних локальних залежностей. Водночас ця модель потребує ретельнішого налаштування глибини дерев, швидкості навчання та кількості ітерацій, оскільки надмірне підсилення призводить до погіршення узагальнювальної здатності.

Модель XGBoost є розвитком ідеології бустингу та поєднує високу прогностичну потужність із засобами регуляризації, контролем складності дерев і ефективною обробкою великих масивів даних. Для задач прогнозування екологічного ризику ця модель є особливо цінною, оскільки дозволяє не лише підвищити точність прогнозу, а й забезпечити гнучке врахування складних комбінацій ознак, які формують інтегральний показник ризику. Саме тому XGBoost доцільно розглядати як одну з базових архітектур у системі багатомодельного прогнозування.

Оцінювання ефективності моделей здійснюється за допомогою метрик, що характеризують середнє відхилення прогнозу від фактичного значення цільової змінної. Однією з базових метрик є середня абсолютна похибка:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}\left|y_i - \hat{y}_i\right|, \tag{7}$$

де y_i – фактичне значення інтегрального ризику; $\hat{y}_i$ – прогнозоване значення інтегрального ризику; N – кількість спостережень.

Середньоквадратична похибка визначається за формулою:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}\ . \tag{8}$$

На відміну від MAE, ця метрика сильніше реагує на великі відхилення, тому є особливо корисною у тих випадках, коли необхідно контролювати точність прогнозів у сегментах підвищеного екологічного ризику.

Для повнішого аналізу якості прогнозу доцільно також використовувати коефіцієнт детермінації:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}\ . \tag{9}$$

де $\bar{y}$ – середнє значення цільової змінної у вибірці.

Метрика (9) відображає, яку частку варіації інтегрального індексу ризику пояснює модель.

Багатомодельний алгоритм у межах програмної системи реалізується як послідовність взаємопов'язаних операцій. Спочатку підготовлений набір ознак надходить на вхід усіх трьох моделей. Далі кожна з них виконує незалежне навчання та формує прогнозні значення. Після цього розраховуються метрики якості, на підставі яких виконується порівняння моделей і визначається та архітектура, що найкраще відображає структуру конкретних даних. У такий спосіб система не фіксує одну "жорстко задану" модель, а працює в адаптивному режимі вибору найефективнішого інструмента прогнозування.

З методологічної точки зору такий підхід є виправданим, оскільки екологічний ризик не має універсальної аналітичної форми. Для одних територіальних сукупностей кращі результати забезпечують моделі типу bagging, для інших – boosting-підходи. Саме тому багатомодельність у

даній роботі розглядається не як дублювання алгоритмів, а як спосіб підвищення достовірності прогнозного контуру та зменшення залежності системи від структурних обмежень одного методу.

Таблиця 1. Порівняння моделей для прогнозування екологічних ризиків

Модель	Принцип побудови	Переваги	Обмеження	Доцільність у задачі прогнозування екологічного ризику
Random Forest	Bagging ансамбль дерев рішень	Стійкість до шуму, низька чутливість до перенавчання, добра робота з великою кількістю ознак	Менша чутливість до слабких локальних залежностей	Ефективна для стабільного прогнозу за неоднорідних і частково зашумлених даних
Gradient Boosting	Послідовне підсилення дерев	Висока точність, здатність моделювати складні нелінійні залежності	Чутливість до параметрів і ризик перенавчання	Доцільна для виявлення складних взаємозв'язків між чинниками ризику
XGBoost	Регуляризований boosting	Висока прогностична потужність, швидкодія, контроль складності моделі	Вища складність налаштування та інтерпретації	Найбільш придатна для побудови точного прогнозу інтегрального індексу ризику

Таким чином, багатомодельний алгоритм прогнозування екологічного ризику формує аналітичне ядро всієї програмної системи. Його використання дозволяє не лише порівнювати різні ансамблеві архітектури, а й обґрунтовано обирати ту модель, яка забезпечує найкращу якість прогнозування для конкретної структури екологічних, просторових та інфраструктурних даних. Саме це створює передумови для подальшого формування адаптивної системи цифрового управління муніципальними органічними відходами на основі надійних прогнозних оцінок.

UML-архітектура програмної взаємодії модулів. Для забезпечення структурованої реалізації багатомодельного прогнозування екологічного ризику програмну систему доцільно організувати за модульним принципом, у якому окремі функціональні блоки реалізують незалежні, але логічно пов'язані етапи обробки даних, навчання моделей, оцінювання результатів та формування управлінських висновків. Такий підхід дозволяє підтримувати масштабованість архітектури, розширювати систему новими аналітичними компонентами без зміни базової логіки та забезпечувати повторне використання окремих програмних модулів у суміжних задачах екологічного аналізу.

У структурі програмної архітектури центральне місце займає клас DataManager, який відповідає за організацію вхідних даних. Його функціональна роль полягає у завантаженні даних із різних джерел, приведенні їх до єдиного формату, виконанні нормалізації, перевірці повноти записів, формуванні навчальної та тестової вибірок, а також підготовці маси-

вів ознак для передачі у прогнозні модулі. Саме через цей клас реалізується первинна взаємодія між табличними даними, геопросторовими характеристиками територій та інтегральними показниками, що використовуються в моделюванні ризику.

Після завершення підготовчого етапу інформаційний потік передається до класу RiskModelEngine, який реалізує ядро багатомодельного прогнозування. У межах цього класу відбувається ініціалізація моделей машинного навчання, їх навчання, тестування, генерація прогнозів та збереження результатів. Архітектурно цей модуль повинен забезпечувати можливість підключення різних алгоритмів через уніфікований інтерфейс, що дозволяє легко додавати нові методи без зміни загальної логіки системи. У цьому класі формується множина прогнозних моделей, описаних виразом (5). Водночас кожна модель реалізується як окремий внутрішній об'єкт прогнозного контуру.

Для порівняння результатів різних моделей використовується клас BenchmarkController, який виконує аналітичну функцію оцінювання точності прогнозів. У цьому модулі обчислюються метрики якості, формується ранжування моделей за ефективністю, визначається модель з мінімальною похибкою та здійснюється формування порівняльної таблиці результатів. Саме цей клас забезпечує адаптивність системи, оскільки дозволяє автоматично вибирати найкращу модель для конкретної структури вхідних даних.

Остаточний етап програмної логіки реалізується класом DecisionSupport, який перетворює числовий прогноз ризику на управлінське рішення. Його функціональна роль полягає у класифікації територій за рівнями ризику, генерації текстових рекомендацій, підготовці аналітичних повідомлень та передачі результатів у зовнішні цифрові сервіси.

Таким чином, UML-архітектура системи відображає логіку послідовної взаємодії між чотирма базовими класами: 1) підготовка даних; 2) прогнозування; 3) оцінювання; 4) підтримка управлінських рішень. Саме така структурна декомпозиція забезпечує оновлення окремих частин системи без зміни її загальної функціональної логіки.

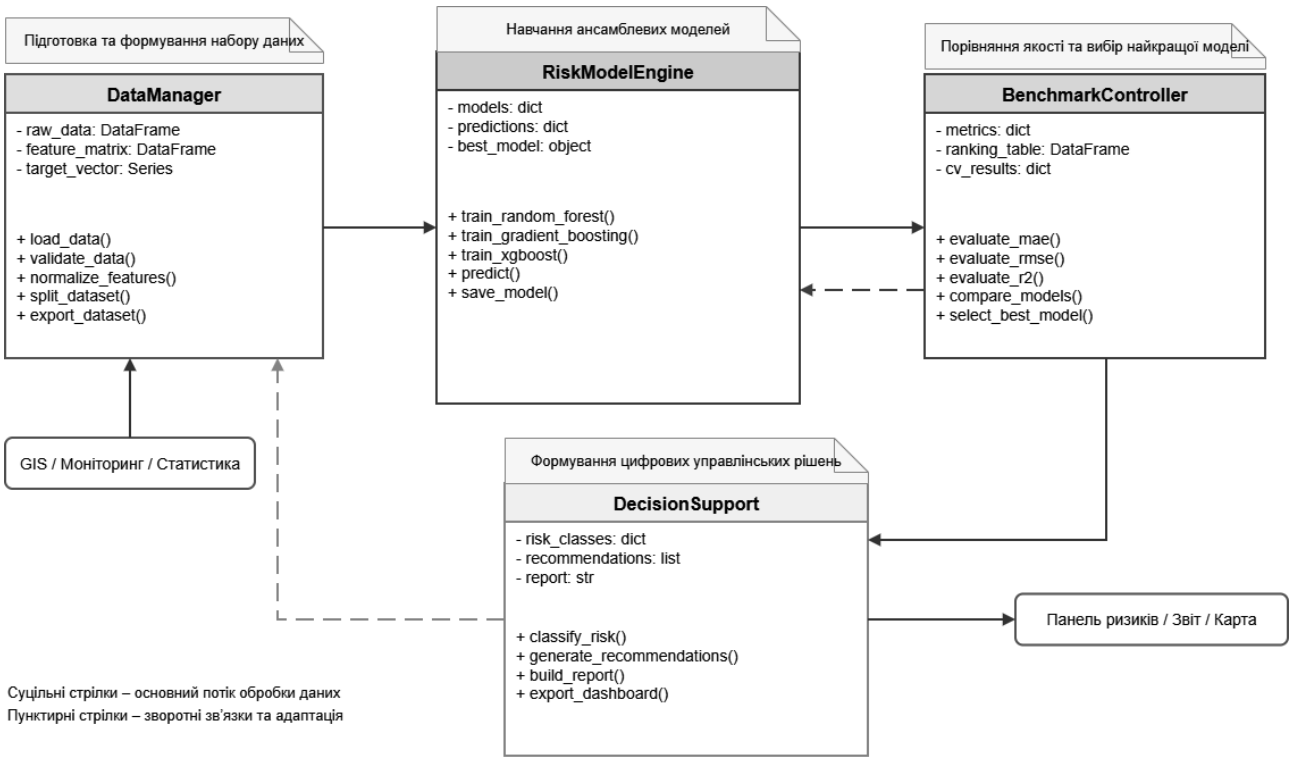

Рис. 2. UML-архітектура програмної системи

Модуль інтерпретації важливості ознак. Однією з принципових вимог до сучасних інтелектуальних систем прогнозування є не лише отримання точного результату, а й пояснення причин, які формують конкретне прогнозне значення. У задачах екологічного ризику це особливо важливо, оскільки управлінські рішення повинні спиратися на зрозумілу аналітичну аргументацію. Для цього у програмній системі реалізовано модуль інтерпретації важливості ознак, який дозволяє оцінити внесок кожного фактора у формування інтегрального індексу ризику.

В межах дослідження використовується підхід permutation importance, що базується на вимірюванні зміни функції втрат після випадкового перемішування значень окремої ознаки:

$$PI_j = E\left[L(y, \hat{f}(X_j^{perm})) - L(y, \hat{f}(X)) \right] \quad (10)$$

.

де PI_j – відображає важливість j-ї ознаки; $L(\cdot)$ – функція втрат моделі; $\hat{f}(X)$ – базовий прогноз моделі; X_j^{perm} – набір даних із випадково переставленими значеннями j-ю ознаки.

Ідея цього підходу полягає в тому, що у разі, якщо після перемішування значень конкретної змінної якість прогнозу екологічного ризику різко погіршується, це означає високу аналітичну значущість відповідного фактора.

У прикладних умовах екологічного прогнозування система часто виявляє домінування просторових ознак. Це пояснюється тим, що територіальні параметри безпосередньо визначають доступність інфраструктури, логістику збору відходів, щільність забудови, близькість до природних об'єктів і транспортних коридорів. Саме тому змінні типу площі території, відстані до водних об'єктів, щільності забудови або просторової структури населеного пункту часто демонструють найбільші значення permutation importance.

Водночас інфраструктурні змінні також мають високу аналітичну вагу. Наприклад, показники інтенсивності транспортної доступності, наявності полігонів, кількості логістичних вузлів чи рівня комунального забезпечення визначають потенційну здатність території до ефективного управління органічними потоками. У цьому випадку високий внесок змінної означає, що саме вона формує найбільший вплив на зміну інтегрального ризику.

Отож, модуль інтерпретації дозволяє перейти від "чорної скриньки" прогнозу до аналітично поясненого механізму ухвалення рішень.

Формування цифрових управлінських рішень. Отримане числове значення прогнозного ризику має бути перетворене на форму, придатну для практичного використання в системах муніципального управління. Для цього у програмній системі реалізовано механізм класифікації прогнозу за трьома рівнями ризику:

$$C(x) = \begin{cases} C_1, & R < 0.33 \\ C_2, & 0.33 \le R < 0.66 \\ C_3, & R \ge 0.66 \end{cases}, \quad (11)$$

де R – прогнозне значення інтегрального індексу ризику.

Клас C_1 відповідає низькому рівню ризику. У цьому випадку система фіксує стабільний стан екологічної ситуації, відсутність критичних тенденцій накопичення негативних факторів та достатню ефективність існуючих управлінських механізмів. Клас C_2 характеризує середній рівень ризику. Такий стан означає наявність факторів, які у середньостроковій перспективі призводять до погіршення ситуації. У цьому випадку система формує рекомендації щодо посиленого моніторингу, уточнення логістичних маршрутів, корекції графіків вивезення відходів або підвищення частоти контролю окремих зон.

Клас C_3 відображає високий рівень ризику. Для таких територій система генерує рекомендації щодо пріоритетного управлінського втручання: перегляду інфраструктурної конфігурації, коригування просторової схеми збору відходів, підсилення контролю критичних ділянок, зміни сценарію управління ресурсами.

Цифрові рекомендації формуються автоматично через набір правил, які пов'язують клас ризику з типовими сценаріями реагування. Таким чином прогноз переходить у форму готового управлінського інструмента.

Особливості програмної реалізації системи. Програмна реалізація системи виконана мовою Python, що забезпечує високу гнучкість побудови аналітичних модулів, широкі можливості інтеграції з бібліотеками машинного навчання та просторового аналізу.

Основним середовищем експериментальної реалізації виступає Google Colab, що дозволяє організовувати повний цикл дослідження без локального розгортання серверної інфраструктури, виконувати навчання моделей у хмарному середовищі та зберігати результати експериментів у єдиному репозиторії.

Для табличної обробки використовується бібліотека Pandas, через яку реалізуються операції очищення, агрегації, нормалізації та трансформації ознак. Саме на цьому рівні формується аналітичний датафрейм, який далі передається у модельний контур.

Моделі Random Forest і Gradient Boosting реалізовані через бібліотеку Scikit-learn, що дозволяє використовувати стандартизовані механізми навчання, кросвалідації, оцінювання та побудови інтерпретаційних характеристик.

Для XGBoost використовується окремий пакет XGBoost, який забезпечує ефективне регуляризоване навчання ансамблевих дерев та високу швидкодію у роботі з багатовимірними наборами даних.

Особливістю системи є інтеграція з GIS-даними, що реалізується через сумісність із геопросторовими структурами даних. Це дозволяє використовувати координатні характеристики, просторові індекси, територіальні відстані та інфраструктурні параметри як повноцінні ознаки - моделі.

У результаті сформована програмна система поєднує аналітичну гнучкість Python, обчислювальні можливості ансамблевих алгоритмів та просторову логіку цифрового управління, що створює основу для масштабованого використання у задачах екологічного прогнозування.

Результати експериментального тестування системи. Експериментальне тестування розробленої програмної системи виконувалося з метою перевірки здатності багатомодельного алгоритму забезпечувати достовірне прогнозування інтегрального екологічного ризику в умовах неоднорідності просторових, інфраструктурних та соціально-економічних характеристик територій. Основна увага приділялася порівнянню точності ансамблевих моделей, аналізу стійкості прогнозу та визначенню тієї архітектури, яка найкраще відображає закономірності формування ризику в умовах муніципального управління органічними відходами.

Для проведення тестування сформовано експериментальний масив даних, що включав територіальні одиниці з різними характеристиками щільності населення, площі, транспортної доступності, близькості до водних об'єктів, рівня урбанізації, доходів населення та показників утворення органічних відходів. Перед початком навчання всі змінні були приведені до безрозмірної форми, а вибірка поділена на навчальну та тестову частини у співвідношенні 80:20. Такий підхід дозволив забезпечити коректну перевірку узагальнюючої здатності моделей.

На першому етапі проведено навчання трьох моделей: Random Forest, Gradient Boosting та XGBoost. Для кожної з них виконано однакову процедуру побудови прогнозу, після чого здійснено оцінювання за критеріями середньої абсолютної похибки, середньоквадратичної похибки та коефіцієнта детермінації. Порівняльний аналіз показав, що всі три алгоритми забезпечують прийнятну якість прогнозування, однак між ними спостерігається суттєва різниця в точності.

У випадку моделі Random Forest отримано стабільний результат із низькою чутливістю до локальних коливань вибірки. Середня абсолютна похибка цієї моделі склала 0.061, середньоквадратична похибка – 0.082, а коефіцієнт детермінації становив 0.881. Такі значення свідчать про достатньо високу узагальнювальну здат-

ність алгоритму, однак модель менш чутливо відображає слабкі нелінійні залежності між окремими просторовими параметрами.

Модель Gradient Boosting продемонструвала вищу адаптивність до складних нелінійних залежностей. Отримане значення середньої абсолютної похибки становило 0.048, а RMSE – 0.069. Коефіцієнт детермінації досяг 0.907, що підтверджує кращу здатність моделі пояснювати варіацію інтегрального індексу ризику порівняно з Random Forest.

Найвищу точність прогнозування забезпечила модель XGBoost. Для неї зафіксовано мінімальне значення середньої абсолютної похибки $MAE = 0.039$ та найменше значення середньоквадратичної похибки – $RMSE = 0.055$. Коефіцієнт детермінації досяг рівня – $R^2 = 0.93$.

Отримані результати свідчать про найвищу прогностичну стійкість саме цієї архітектури. Це пояснюється тим, що XGBoost поєднує механізм послідовного підсилення слабких моделей із регуляризацією, яка стримує перенавчання та забезпечує кращу узагальнюючу здатність.

Таблиця 2. Результати експериментального тестування моделей

Модель	MAE	RMSE	R^2	Узагальнена оцінка
Random Forest	0.061	0.082	0.881	Стабільний прогноз за високої неоднорідності даних
Gradient Boosting	0.048	0.069	0.907	Висока чутливість до нелінійних залежностей
XGBoost	0.039	0.055	0.930	Найвища точність прогнозування

Порівняння прогнозних значень із фактичними показало, що найбільші відхилення виникають у територіальних одиницях із комбінованим впливом просторових та інфраструктурних факторів. Саме у цих випадках роль складних взаємодій між ознаками найбільш виражена, а перевага XGBoost проявляється найчіткіше.

Додатковий аналіз permutation importance підтвердив, що найбільший внесок у формування прогнозу мають змінні, пов'язані з просторовою структурою території. Найвищі значення важливості було зафіксовано для ознак, що характеризують площу території, частку урбанізованої забудови, відстань до водних об'єктів і рівень транспортної доступності. Це свідчить про те, що геопросторові параметри визначають базову архітектуру формування екологічного ризику.

Окремо проведено аналіз поведінки системи у процесі класифікації прогнозів за рівнями ризику. Встановлено, що більшість територій із високими значеннями індексу ризику належать до класу C_3, де необхідне активне управлінське втручання. Для середнього рівня ризику система формує сценарії посиленого моніторингу, тоді як для низького рівня ризику пропонується збереження чинної організації управління.

Із технологічної точки зору тестування підтвердило стабільність програмної реалізації у середовищі Python та відсутність критичних обмежень при масштабуванні набору даних. Навіть у разі збільшення кількості спостережень система зберігала стійкість виконання, а час побудови прогнозу залишався прийнятним для практичного використання.

Таким чином, результати експериментального тестування підтвердили, що запропонована програмна система здатна забезпечувати високоточне прогнозування екологічного ризику, а багатомодельний підхід створює аналітичну основу для адаптивного цифрового управління муніципальними екологічними процесами.

Висновки

У роботі розроблено архітектуру програмної системи багатомодельного прогнозування екологічного ризику для вирішення задачі цифрового управління муніципальними органічними відходами. Вона базується на інтеграції методів машинного навчання, модульної програмної організації, механізмів інтерпретації важливості ознак та цифрової підтримки управлінських рішень. Запропонований підхід орієнтований на обробку неоднорідних просторових, інфраструктурних і соціально-економічних даних, що дозволяє формувати аналітично обґрунтовані прогнози інтегрального екологічного ризику.

У межах дослідження сформовано багатомодельний алгоритм прогнозування екологічного ризику, до складу якого включено ансамблеві методи Random Forest, Gradient Boosting та XGBoost. Реалізовано формальну схему побудови прогнозу екологічного ризику, в якій кожна модель функціонує як окремий аналітичний модуль, а підсумкова оцінка якості здійснюється за критеріями MAE, RMSE та R^2. Це дозволило забезпечити не лише порівняння моделей, а й забезпечити адаптивний вибір тієї архітектури, яка найкраще відповідає структурі конкретних даних.

Розроблено UML-архітектуру програмної системи, у якій передбачена функціональна взаємодія між класами DataManager, RiskModelEngine, BenchmarkController та DecisionSupport, що забезпечує повний цикл обробки – від завантаження і нормалізації даних до формування цифрових управлінських рекомендацій. Така модульна побудова уможливлює масштабування системи, підключення нових моделей та інтеграції із зовнішніми аналітичними сервісами.

Важливою складовою системи є модуль інтерпретації важливості ознак, реалізований на основі підходу permutation importance, що дозволяє оцінювати реальний внесок окремих факторів у формування прогнозу. Це забезпечує пояснюваність результатів машинного навчання та дозволяє виявляти фактори, які найбільше впливають на зміну екологічного ризику. В результаті досліджень встановлено, що найбільшу аналітичну вагу мають просторові характеристики територій, показники транспортної доступності, рівень урбанізації та інфраструктурні параметри.

Експериментальне тестування програмної системи багатомодельного прогнозування екологічних ризиків підтвердило високу ефективність запропонованої архітектури. Найкращі результати продемонструвала модель XGBoost, для якої отримано MAE = 0.039, RMSE = 0.055 та $R^2 = 0.930$, що свідчить про високий рівень узагальнювальної здатності моделі та її придатність до використання у прикладних задачах цифрового екологічного управління.

Запропонований підхід дозволяє переводити числові результати прогнозування екологічного ризику у цифрові управлінські рішення шляхом автоматичного віднесення територій до класів ризику, що створює основу для формування рекомендацій щодо організації логістики, моніторингу, просторового планування та екологічного реагування на муніципальному рівні.

Подальші дослідження доцільно проводити у напрямку розширення набору моделей за рахунок нейромережевих архітектур, інтеграцію потокових сенсорних даних у реальному часі, використання геоінформаційних платформ для автоматичного оновлення просторових ознак, а також розроблення цифрового двійника екологічної системи муніципального управління органічними відходами. Перспективним напрямом також є інтеграція системи з міськими інформаційними платформами для підтримки стратегічного управління сталим розвитком територій міських - громад.

Література

1. P. Xu, H. Zheng, A multi-AI approach to predicting municipal solid waste generation and recycling demand in Hong Kong, *Resources, Conservation and Recycling*. 2026. Vol. 225. Art. 108590. DOI: https://doi.org/10.1016/j.resconrec.2025.108590

2. W. Xia, J. Chen, X. Li, Application of machine learning algorithms in municipal solid waste management: a review, *Waste Management & Research*. 2022. Vol. 40, № 1. P. 7–19. DOI: https://doi.org/10.1177/0734242X211033716
3. S. D. Latif, N. A. B. Hazrin, M. K. Younes, A. N. Ahmed, A. Elshafie, Evaluating different machine learning models for predicting municipal solid waste generation: A case study of Malaysia, *Environment, Development and Sustainability*. 2024. Vol. 26, № 5. P. 12875–12898. DOI: https://doi.org/10.1007/s10668-023-03882-x.
4. 4. K. Ameri, M. Hempel, H. Sharif, J. Lopez, K. Perumalla, Design of a novel information system for semi-automated management of cybersecurity in industrial control systems, *ACM Transactions on Management Information Systems*. 2022. Vol. 14, № 1. P. 4–35. DOI: https://doi.org/10.1145/3546580.
5. A. Tryhuba, T. Hutsol, S. Glowacki, European Green Deal: Justification of the relationships between the functional indicators of bioenergy production systems using organic residential waste, *Energies*. 2024. Vol. 17, № 6. Art. 1461. DOI: https://doi.org/10.3390/en17061461
6. A. N. Ahmed, A. F. Ghani, N. S. A. Wahid, Comparative analysis of machine learning algorithms for municipal waste prediction, *Science of the Total Environment*. 2023. Vol. 857. Art. 159493. DOI: https://doi.org/10.1016/j.scitotenv.2022.159493.
7. M. S. Islam, R. Hyder, Explainable artificial intelligence in environmental decision systems: limitations and future challenges, *Environmental Modelling & Software*. 2024. Vol. 176. Art. 106008. DOI: https://doi.org/10.1016/j.envsoft.2024.106008.
8. H. Zheng, P. Xu, Statistical forecasting approaches for municipal waste generation under urban dynamics, Waste Management. 2022. Vol. 138. P. 1–10. DOI: https://doi.org/10.1016/j.wasman.2021.11.018.
9. B. Liu, J. Chen, Forecasting municipal solid waste generation using long short-term memory neural networks, *Environmental Earth Sciences*. 2024. Vol. 83. Art. 399. DOI: https://doi.org/10.1007/s12665-024-11702-2.
10. J. Kaza, L. Yao, S. Bhada-Tata, F. Van Woerden, What a Waste 2.0: A Global Snapshot of Solid Waste Management to 2050. World Bank, 2023 update. Accessed: 28.01.2026. URL: https://www.worldbank.org/en/publication/what-a-waste
11. J. Chen, Y. Yang, Z. Feng, R. Huang, G. Zhou, H. You, X. Han, Ecological Risk Assessment and Prediction Based on Scale Optimization - A Case Study of Nanning, a Landscape Garden City in China, *Remote Sensing*. 2023. Vol. 15, № 5. Art. 1304. DOI: https://doi.org/10.3390/rs15051304.
12. D. Zhou, Y. Jia, N. Zhang, H. Hou, F. Wang, Z. Wang, An ensemble machine-learning model with online learning strategy for building load forecasting, *Applied Thermal Engineering*. 2026. Vol. 288, Part 1. Art. 129434. DOI: https://doi.org/10.1016/j.applthermaleng.2025.129434.
13. A. Tryhuba, T. Hutsol, S. Glowacki, V. Tryhuba, Prediction of biogas production volumes from household organic waste based on machine learning, *Energies*. 2024. Vol. 17, № 7. Art. 1786. DOI: https://doi.org/10.3390/en17071786.
14. А. М. Тригуба, О. Я. Андрушків, І. Л. Тригуба, Циркуляційно-ціннісна модель управління проєктами енергозабезпечення житлових масивів, Управління розвитком складних систем. 2025. № 64. С. 138–152. DOI: https://doi.org/10.32347/2412-9933.2025.64.138-152
15. А. Тригуба, Р. Шолудько, О. Андрушків, Р. Олійник, М. Коциловський, Інтелектуальні моделі управління інфраструктурними проєктами розвитку громад в умовах багаторівневих ризиків, Вісник Львівського державного університету безпеки життєдіяльності. 2025. № 31. С. 213–226. DOI: https://doi.org/10.32447/20784643.31.2025.21

References

1. P. Xu, H. Zheng, A multi-AI approach to predicting municipal solid waste generation and recycling demand in Hong Kong, in: Resources, Conservation and Recycling 225 (2026) 108590. doi: 10.1016/j.resconrec.2025.108590.
2. W. Xia, J. Chen, X. Li, Application of machine learning algorithms in municipal solid waste management: a review, in: Waste Management & Research 40(1) (2022) 7–19. doi: 10.1177/0734242X211033716.
3. S. D. Latif, N. A. B. Hazrin, M. K. Younes, A. N. Ahmed, A. Elshafie, Evaluating different machine learning models for predicting municipal solid waste generation: a

case study of Malaysia, in: Environment, Development and Sustainability 26(5) (2024) 12875–12898. doi: 10.1007/s10668-023-03882-x.

4. K. Ameri, M. Hempel, H. Sharif, J. Lopez, K. Perumalla, Design of a novel information system for semi-automated management of cybersecurity in industrial control systems, in: ACM Transactions on Management Information Systems 14(1) (2022) 4–35. doi: 10.1145/3546580.
5. A. Tryhuba, T. Hutsol, S. Glowacki, European Green Deal: justification of the relationships between the functional indicators of bioenergy production systems using organic residential waste, in: Energies 17(6) (2024) 1461. doi: 10.3390/en17061461.
6. A. N. Ahmed, A. F. Ghani, N. S. A. Wahid, Comparative analysis of machine learning algorithms for municipal waste prediction, in: Science of the Total Environment 857 (2023) 159493. doi: 10.1016/j.scitotenv.2022.159493.
7. M. S. Islam, R. Hyder, Explainable artificial intelligence in environmental decision systems: limitations and future challenges, in: Environmental Modelling & Software 176 (2024) 106008. doi: 10.1016/j.envsoft.2024.106008.
8. H. Zheng, P. Xu, Statistical forecasting approaches for municipal waste generation under urban dynamics, in: Waste Management 138 (2022) 1–10. doi: 10.1016/j.wasman.2021.11.018.
9. B. Liu, J. Chen, Forecasting municipal solid waste generation using long short-term memory neural networks, in: Environmental Earth Sciences 83 (2024) 399. doi: 10.1007/s12665-024-11702-2.
10. J. Kaza, L. Yao, S. Bhada-Tata, F. Van Woerden, What a Waste 2.0: A Global Snapshot of Solid Waste Management to 2050, World Bank (2023 update). URL: https://www.worldbank.org/en/publication/what-a-waste.
11. J. Chen, Y. Yang, Z. Feng, R. Huang, G. Zhou, H. You, X. Han, Ecological risk assessment and prediction based on scale optimization: a case study of Nanning, a landscape garden city in China, in: Remote Sensing 15(5) (2023) 1304. doi: 10.3390/rs15051304.
12. D. Zhou, Y. Jia, N. Zhang, H. Hou, F. Wang, Z. Wang, An ensemble machine-learning model with online learning strategy for building load forecasting, in: Applied Thermal Engineering 288 (2026) 129434. doi: 10.1016/j.applthermaleng.2025.129434.
13. A. Tryhuba, T. Hutsol, S. Glowacki, V. Tryhuba, Prediction of biogas production volumes from household organic waste based on machine learning, in: Energies 17(7) (2024) 1786. doi: 10.3390/en17071786.
14. A. M. Tryhuba, O. Ya. Andrushkiv, I. L. Tryhuba, Circular-value model of management of energy supply projects for residential areas, in: Management of Development of Complex Systems 64 (2025) 138–152. doi: 10.32347/2412-9933.2025.64.138-152. [in Ukrainian]
15. A. Tryhuba, R. Sholudko, O. Andrushkiv, R. Oliinyk, M. Kotsylovskyi, Intelligent models of management of infrastructure development projects of communities under multilevel risks, in: Bulletin of Lviv State University of Life Safety 31 (2025) 213–226. doi: 10.32447/20784643.31.2025.21. [in Ukrainian]

Дата першого надходження до видання: 29.01.2026
Внутрішня рецензія отримана: 06.02.2026
Зовнішня рецензія отримана: 08.02.2026
Дата прийняття статті до друку: 19.03.2026
Дата публікації: 16.04.2026

Про авторів:

[1]Тригуба Анатолій Миколайович,
доктор технічних наук, професор,
завідувач кафедри інформаційних
технологій
[1] Tryhuba Anatoliy,
Ph.D (doctor, technical sciences), professor,
head of department
https://orcid.org/0000-0001-8014-5661

[2]Коваль Назарій Ярославович,
доктор філософії, проректор
[2] Koval Nazariy,
Ph.D, deputy president
https://orcid.org/0000-0001-7846-2924

[1]Тригуба Інна Леонтіївна,
кандидат сільськогосподарських наук,
доцент, завідувач кафедри генетики,
селекції та захисту рослин
[1] Tryhuba Inna,
Ph.D (agricaltural sciences),
associate professor, head of department
https://orcid.org/0000-0002-5239-5951

[2]Фірман Ігор Романович,
здобувач кафедри інформаційних
технологій та систем електронних
комунікацій

[2] Firman Igor,
Candidate for a scientific degree
https://orcid.org/0009-0005-5840-9815

[1]Фамуляк Володимир Юрійович,
аспірант кафедри інформаційних
технологій
[1] Famuliak Volodymyr,
Post-graduate student
https://orcid.org/0009-0005-5347-9427

Місце роботи авторів:

[1] Львівський національний університет
ветеринарної медицини та біотехнологій
імені Степана Ґжицького
[1] Lviv National University of Veterinary
Medicine and Biotechnology named
after S. Gzhytsky
E-mail: trianamik@gmail.com,
trinle@ukr.net, vovfam@gmail.com
Сайт: https://lvet.edu.ua/

[2] Львівський державний університет
безпеки життєдіяльності
[2] Lviv State University of Life Safety
E-mail: kovaln870@gmail.com,
firmanigorromanovuch@gmail.com
Сайт: https://ldubgd.edu.ua

УДК 004.4 https://doi.org/10.15407/pp2026.01.040

В.І. Шинкаренко, О.*В. Макаров*

КОНСТРУКТИВНО-ПРОДУКЦІЙНЕ ФОРМУВАННЯ ПРОГРАМ СОРТУВАННЯ, АДАПТОВАНИХ ГЕНЕТИЧНИМ АЛГОРИТМОМ

У попередніх роботах представлені механізми конструктивно-продукційного моделювання для адаптації алгоритмів сортування. У зв'язку з цим виникли задачі перетворення хромосом генетичного алгоритму на текст програм сортування для подальшого застосування, оцінки можливостей еволюційного розвитку. Розглядається підхід до перетворення хромосом, що кодують алгоритми сортування на тексти програм готових до застосування у реальних умовах. Розроблено конструктор-трансформер, який реалізує перетворення хромосоми-дерева на лінійну послідовність генів. Інший конструктор-трансформер призначений для перетворення послідовності генів на код програми сортування. Наведено приклади послідовності обходу дерева-хромосоми, додавання генів до лінійної послідовності і формування тексту програми. Проведено експерименти із вхідними даними різної структури і обсягів. Результати експериментів підтвердили, що запропонована методика може бути використана для автоматичної генерації ефективних алгоритмів сортування. А застосування конструктивно-продукційного моделювання у сукупності із генетичним алгоритмом дозволяє ефективно виконувати структурну адаптацію алгоритмів.
Ключові слова: конструктивно-продукційне моделювання, алгоритми сортування, часова ефективність, генетичний алгоритм, програмне забезпечення, інформаційні технології.

V.I. Shynkarenko, O.V. Makarov

CONSTRUCTIVE-SYNTHESIZING PRODUCTION OF SORTING PROGRAMS ADAPTED BY GENETIC ALGORITHM

In previous works, the mechanisms of constructive-synthesizing modeling for the adaptation of sorting algorithms were presented. In this regard, the task of transforming the chromosomes of a genetic algorithm into the text of sorting programs for further application, evaluation and the possibility of evolutionary development arose. An approach to the transformation of the chromosome encoding a sorting algorithm into the text of a sorting program ready for use in real conditions is considered. A transformer constructor has been developed that implements the transformation of a chromosome tree into a linear sequence of genes. Another transformer constructor is designed to transform a sequence of genes into the code of a sorting program. Examples of the sequence of traversing a chromosome tree, adding genes to a linear sequence and forming the text of the program are given. Experiments were conducted with input data of different structures and volumes. The results of the experiments confirmed that the proposed method can be used for the automatic generation of effective sorting algorithms. And the use of constructive-synthesizing modeling in conjunction with a genetic algorithm allows for the effective structural adaptation of algorithms.
Key words: constructive-synthesizing modeling, sorting algorithm, time efficiency, genetic algorithm, software, information technology.

Вступ

Сортування – один із фундаментальних будівельних блоків алгоритмічної інженерії. Класичні порівняльні методи, такі як quicksort [1], mergesort [2] і heapsort [3] формують «базову лінійку» загального призначення, демонструючи очікувану обчислювальну складність $O(n \log n)$ у середньому чи в гіршому випадку. Непорівняльні підходи – counting sort [4], radix sort [5], bucket sort [6] – досягають лінійних меж $O(n)$ за умови обмеженого діапазону або специфічного розподілу ключів у вхідних даних. Не дивлячись на класичні алгоритми, які розробляються давно, зараз продовжується різностороння діяльність з їхнього вдосконалення.

Одним із яскравих прикладів сучасного прогресу є робота "Fast and Simple Sorting Using Partial Information" [7], де розглядається проблема сортування з частко-

вою інформацією, коли деякі попарні порівняння вже відомі. Деяка кількість попередніх порівнянь доступні, і завдання полягає в тому, щоб використати мінімальну кількість нових порівнянь для повного відсортування даних.

Паралельно тривають дослідження алгоритмів, які використовують локальність даних та їхню кластерну структуру для прискорення сортування. Новітній представник цього напряму – Cluster Sort [8], запропонований як гібридний in-place підхід: елементи групуються (кластеризуються) з урахуванням просторової локальності, після чого в межах кластерів застосовуються ефективні процедури впорядкування.

Наразі запропоновано декілька видів детермінованих передобробок даних, призначених для сортування [9, 10]. Проведені експерименти, які показали, що використання передобробок може підвищити часову ефективність алгоритмів сортування. Тому доцільно використати алгоритми передобробки у процесі формування алгоритмів сортування. Це розширює можливості для конструювання, значно збільшує варіативність сформованих алгоритмів, однак привносить додаткові правила та обмеження. Через те, що алгоритми передобробки мають схожий принцип дії, недоцільне послідовне використання кількох алгоритмів передобробки, що унеможливлює використання двох алгоритмів передобробки одночасно.

У статті [11] була запропонована конструктивна модель хромосоми, в якій алгоритм сортування кодується як набір елементарних складових (production-based fragments), що можуть комбінуватися, розширюватися та еволюціонувати. Подання алгоритму у вигляді конструктора, де частини алгоритмів сортування виступають як будівельні блоки, дозволяє:

- поєднувати фрагменти різних класичних алгоритмів сортування;
- автоматично створювати гібридні алгоритми;
- адаптувати структуру алгоритму під конкретні властивості даних і умови застосування.

У продовження попередньої статті [12], де було сформовано підхід до конструктивно-продукційного моделювання хромосом, застосовано розроблену методику для дослідження та структурної адаптації алгоритмів сортування. За допомогою конструктора-трансформера, виконується декодування хромосоми у послідовність генів. Конструктор формування тексту програм перетворює отриману послідовність на код програми обраною мовою програмування.

Виконані експерименти у яких конструйовані алгоритми сортування застосовуються у різних обсягах і даних. За допомогою генетичного алгоритму [13] виконується адаптація структури сформованих алгоритмів до специфічних характеристик вхідних даних, що дозволяє отримувати алгоритми з кращими часовими показниками у заданих умовах використання.

Конструктор-трансформер

Конструктор-трансформер реалізує перетворення дерева-хромосоми із закодованими складовими алгоритму сортування на масив (послідовність) генів.

Визначимо спеціалізацію конструктора C_{CD} на основі узагальненого коструктора C [14]:

$$C = \langle \mathrm{M}, \Sigma, \Lambda \rangle \ {}_{s}\!\longmapsto\ C_{CD} = \langle \mathrm{M}_{CD}, \Sigma_{CD}, \Lambda_{CD} \rangle, \quad (1)$$

де M – неоднорідний розширюваний носій структури, Σ – сигнатура, що складається з множин операцій зв'язування, підстановки і виводу, операцій над атрибутами і відношення підстановки, Λ – інформаційне забезпечення конструювання, яке включає онтологію, цілі, правила і обмеження конструювання; M_{CD} – носій, що включає термінальний (T_{CD}) і нетермінальний (N_{CD}) алфавіт, а також множину правил продукції Ψ з правилами $\psi_i = \langle s_{\mathrm{i}}, g_{\mathrm{i}} \rangle \in \Psi$, де i – номер правила, s_{i} – послідовність відношень підстановки та g_{i} – операцій над атрибутами. Σ_{CD} – операції та відносини на елементах M_{CD}. Інформаційне забезпечення конструювання (ІЗК) $\Lambda_{CD} \supset \Lambda$.

Конструктор C_{CD} містить правила підстановки $\psi_i = \langle \langle s_{i,1}, s_{i,2} \rangle, \langle g_{\mathrm{i},1}, g_{\mathrm{i},2} \rangle \rangle$, де

відношення $s_{i,1}$ реалізує розбір хромосоми з використанням стеку для збереження проміжних результатів. Відношення $s_{i,2}$ призначене для формування списку генів.

Операції виконуються у наступній послідовності: спочатку виконуються операції над атрибутами $g_{i,1}$, потім операції підстановки $s_{i,1}$ та $s_{i,2}$, і в кінці – $g_{i,2}$.

Нетермінальний алфавіт $N = \{\alpha_i\}$ складається з допоміжних символів.

Термінальний алфавіт – множина генів хромосоми, що трансформується. Наприклад, ген швидкої передобробки QP – передбачення позиції елемента у відсортованому масиві і перестановка, ген, який відповідає одному проходу алгоритму сортування бульбашкою у зворотному порядку – BSB. Повний перелік генів представлено у [12]. Термінали 'start' та 'end' відповідають початковій і кінцевій послідовностям генів вузла – 'Start' і 'End' на рис. 1.

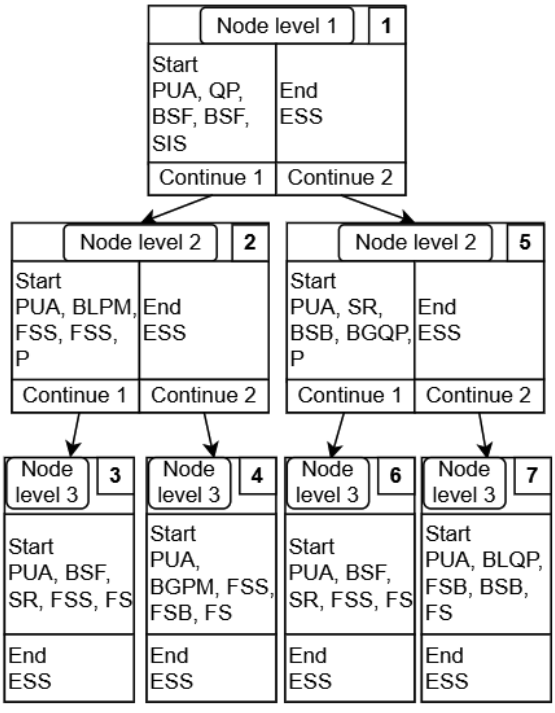

Рис. 1. Приклад хромосоми у вигляді дерева

Кожний вузол дерева на рис. 1 має початкову і кінцеву послідовності генів а також вказівники на вузли-нащадки. Листкові вузли мають порожні вказівники, тобто не мають нащадків.

У процесі трансформації дерева-хромосоми операції підстановки реалізують додавання терміналів (генів) до результуючої послідовності (масиву). Для обходу дерева вузли додаються до допоміжної структури – стеку, використання якого замінює рекурсію.

Атрибути поточної форми:

- current – вказівник на поточний вузол дерева-хромосоми;
- stack – стек вказівників на вузли дерева, які не були трансформовані;
- stackSize – розмір стеку вказівників на вузли дерева.

Розглянемо особливості відношень підстановки.

При використанні відношення $\alpha \to S \downarrow \beta \cdot \alpha$ у процесі конструювання β може бути додано до стеку S. Подальші підстановки будуть продовжено для α.

Використання відношення $\alpha \ {}_{\tau}\!\to \beta \uparrow S \cdot \alpha$ дає можливість дістати верхній елемент із стеку S і покласти його у β, якщо атрибут доступності τ відношення підстановки дорівнює true.

Виконаємо конкретизацію конструктора C_{CD}.

$$C_{CD} = \langle \mathrm{M}_{CD}, \Sigma_{CD}, \Lambda_{CD} \rangle \ {}_{K}\!\longmapsto\ C_{CDK} = \langle \mathrm{M}_{CDK}, \Sigma_{CDK}, \Lambda_{CDK} \rangle, \quad (2)$$

де $\mathrm{M}_{CDK} = \mathrm{M}_{CD}, \Sigma_{CDK} = \Sigma_{CD}, \Lambda_{CDK} \supset \Lambda_{CD}$.

Інформаційне забезпечення конструювання Λ_{CDK} розширене наступними правилами.

Перше правило $\psi_1 = \langle \langle s_{1,1}, s_{1,2} \rangle, \langle g_{1,1}, g_{1,2} \rangle \rangle$ реалізує додавання кореневого вузла дерева-хромосоми до стеку вузлів з яких ще не були скопійовані послідовності генів. Буде виконуватись тільки один раз на початку процесу трансформації.

$$\begin{gathered} s_{1,1} = \langle \alpha \to stack \downarrow root \downarrow chromosome \cdot \alpha \rangle; \\ s_{1,2} = \langle \varepsilon \rangle; \\ g_{1,1} = \langle \varepsilon \rangle; \\ g_{1,2} = \langle =: (stackSize, 1) \rangle \end{gathered} \quad (2)$$

Правило $\psi_2 = \langle \langle s_{2,1}, s_{2,2} \rangle, \langle g_{2,1}, g_{2,2} \rangle \rangle$ реалізує діставання вузла зі стеку вузлів і додавання початкової і кінцевої послідовностей генів до послідовності терміналів.

$$
\begin{aligned}
s_{2,1} &= \langle \alpha \xrightarrow{\tau_1} current \uparrow stack \cdot \alpha \rangle; \\
s_{2,2} &= \langle \rho \rightarrow start \downarrow current \cdot \rho \cdot \\
&\quad end \downarrow current \rangle; \\
&> (f_1, \text{stackSize}, 0), \\
g_{2,1} &= \langle == (f_2, current, null), \rangle; \\
&\wedge (\tau_1, f_1, f_2) \\
g_{2,2} &= \langle -(stackSize, stackSize, 1) \rangle
\end{aligned} \tag{3}
$$

Третє правило $\psi_3 = \langle \langle s_{3,1}, s_{3,2} \rangle, \langle g_{3,1}, g_{3,2} \rangle \rangle$ реалізує додавання вузлів нащадків поточного вузла до стеку. У подальшому вузли будуть діставатись зі стеку у зворотному порядку, тому спочатку додається правий, а потім лівий вузол.

$$
\begin{aligned}
s_{3,1} &= \\
&\left\langle \begin{matrix} \alpha \xrightarrow{\tau_1} stack \downarrow right \downarrow current \cdot \alpha \\ \alpha \xrightarrow{\tau_1} stack \downarrow left \downarrow current \cdot \alpha \end{matrix} \right\rangle; \\
s_{3,2} &= \langle \varepsilon \rangle; \\
g_{3,1} &= \langle ! = (\tau_1, left \downarrow \\
&\quad current, null) \rangle; \\
g_{3,2} &= \left\langle \begin{matrix} +(stackSize, stackSize, 2) \\ =: (current, null) \end{matrix} \right\rangle
\end{aligned} \tag{4}
$$

Початкові умови конструювання: хромосома деревоподібної структури, де у кожного вузла є початкова і кінцева послідовності генів, і всі вузли, окрім листових, мають два вузли-нащадки.

Таблиця 1.

Приклад використання стеку у процесі трансформації хромосоми

Стек вузлів	Вузол, який дістається зі стеку	Вузол, з якого копіюються гени і нащадки додаються до стеку
[1]		
[]	1	
[5,2]		1
[5]	2	
[5,4,3]		2
[5,4]	3	
[5,4]		3
[5]	4	
[5]		4
[]	5	
[7,6]		5
[7]	6	
[7]		6
[]	7	
[]		7

У табл. 1 представлений процес обходу дерева з використанням стеку. Числа у таблиці відповідають номерам вузлів на рис. 1. Вузли будуть опрацьовані у наступній послідовності: 1, 2, 3, 4, 5, 6, 7.

Результатом трансформації буде наступна послідовність генів: [PUA, QP, BSF, BSF, SIS, PUA, BLPM, FSS, FSS, P, PUA, BSF, SR, FSS, FS, ESS, BGPM, FSS, FSB, FS, ESS, ESS, PUA, SR, BSB, BGQP, P, PUA, BSF, SR, FSS, FS, ESS, PUA, BLQP, FSB, BSB, FS, ESS, ESS, ESS].

Конструктор формування тексту програм

Конструктор формування тексту програм (теж конструктор-трансформер) перетворює послідовність генів на текст програми обраною мовою програмування (у даній роботі – C++).

Визначимо спеціалізацію конструктора C_{PF} – моделі декодування послідовності генів, що кодує алгоритм сортування, у текст програми:

$$
\begin{aligned}
C = \langle \mathrm{M}, \Sigma, \Lambda \rangle \ &\xrightarrow{s} \ C_{PF} \\
&= \langle \mathrm{M}_{PF}, \Sigma_{PF}, \Lambda_{PF} \rangle,
\end{aligned} \tag{5}
$$

де M_{PF} – носій, що включає термінальний (T_{PF}) і нетермінальний (N_{PF}) алфавіт, а також множину правил продукції Ψ з правилами $\psi_i = \langle s_\mathrm{i}, g_\mathrm{i} \rangle \in \Psi$, де i – номер правила, s_i – послідовність відношень підстановки, g_i – послідовність операцій над атрибутами. Σ_{PF} – операції та відносини на елементах M_{PF}. ІЗК $\Lambda_{PF} \supset \Lambda$.

Нетермінали – гени хромосоми, яка кодує алгоритм сортування.

Термінали – текстові фрагменти програм обраною мовою програмування, які відповідають частинам відомих алгоритмів сортування, передобробок і допоміжних функцій, і кодуються генами. Відповідно до гена X будемо використовувати термінал $\mathrm{T_X}$. Наведемо список усіх терміналів: T_S, T_E, $T_{\mathrm{B}SF}$, $T_{\mathrm{B}SB}$, $T_{\mathrm{F}SB}$, $T_{\mathrm{F}SS}$, T_{SEEI}, $T_{\mathrm{SE}I}$, $T_{\mathrm{SI}S}$, T_P, T_FS, $T_{\mathrm{P}UA}$, T_ESS, $T_{\mathrm{M}L}$, $T_{\mathrm{M}R}$, $T_{\mathrm{Q}P}$, $T_{\mathrm{P}M}$, T_SR, $T_{\mathrm{BL}QP}$, $T_{\mathrm{BL}PM}$, $T_{\mathrm{BG}QP}$, $T_{\mathrm{BG}PM}$.

Усі термінали $T^* \in T_{PF}$ мають атрибут $text \downarrow T^*$ – текст програми відповідний алгоритму, який кодується геном.

Атрибути терміналів $text \downarrow T_{\mathrm{S}}$ і $text \downarrow T_{\mathrm{E}}$ є незмінними (для любого сформованого алгоритму) частинами тексту програми, які додаються відповідно на початку і в кінці файлу. На початку (T_{S}) включаються заголовкові файли із допоміжними функціями і типами даних, оголошуються допоміжні структури, початок реалізації функції сортування. Кінцева частина (T_{E}) включає код злиття усіх пар із стеку відсортованих масивів для злиття.

Наступні термінали включають перевірку флагу відсортованості, оновлення значень змінних, що входять до глобального контексту, і код програми відповідно до:

- T_{BSF} – одного проходу сортуванням бульбашкою в прямому напрямку;
- T_{BSB} – одного проходу сортуванням бульбашкою у зворотному напрямку;
- T_{FSB} – пошуку найбільшого елемента у невідсортованій частині масиву і перестановка на поточне місце;
- T_{FSS} – пошуку найменшого елемента у невідсортованій частині масиву і перестановка на поточне місце;
- T_{SEEI} – вставки поточного елемента у відсортовану частину у кінці масиву;
- T_{SEI} – вставки поточного елемента у відсортовану частину на початку масиву;
- T_{QP} – швидкої передобробки;
- T_{PM} – передобробки із пам'яттю;
- T_{SR} – передобробки із розворотом;
- T_{BLQP} – блочної локальної швидкої передобробки;
- T_{BLPM} – блочної локальної передобробки з пам'яттю;
- T_{BGQP} – блочної глобальної швидкої передобробки;
- T_{BGPM} – блочної глобальної передобробки з пам'яттю.

Вказані передобробки викладені в [10].

Атрибут $text \downarrow T_{SIS}$ представляє текст програми, який розділяє масив на дві частини, зберігає вказівники і розміри у стеку невідсортованих підмасивів (для подальшого окремного сортування) і у стеку масивів, для яких виконуватиметься злиття у кінці конструйованої функції.

Атрибут $text \downarrow T_{\mathrm{P}}$ є текстом, який обирає елемент, переставляє менші і більші елементи відповідно зліва і справа, зберігає вказівники і розміри частин масиву для подальшого сортування.

$text \downarrow T_{\mathrm{FS}}$ – атрибут, який є текстом загальновідомого алгоритму сортування. У поточній роботі використовується текст швидкого сортування.

Атрибути $text \downarrow T_{PUA}$ і $text \downarrow T_{ESS}$ представляють код допоміжних функцій. Перший дістає невідсортований масив зі стеку невідсортованих масивів для подальшого сортування. Другий додає закриті фігурні дужки до тексту програми у кількості, потрібній для успішної компіляції програми.

Атрибути терміналів $text \downarrow T_{ML}$ і $text \downarrow T_{MR}$ містять код, який оновлює значення змінних із глобального контексту і додає до стеку масивів для подальшого злиття відсортований підмасив (відповідно зліва і справа) та решту масиву.

Результатом закінчення конструювання буде цілісний текст програми, який включає конструйовану функцію сортування, допоміжні структури і функції.

Виконаємо конкретизацію конструктора C_{PF}:

$$C_{PF} = \langle \mathrm{M}_{PF}, \Sigma_{PF}, \Lambda_{PF} \rangle \underset{K}{\longmapsto} C_{PFK} = \langle \mathrm{M}_{PFK}, \Sigma_{PFK}, \Lambda_{PFK} \rangle, \quad (6)$$

де $\mathrm{M}_{PFK} = \mathrm{M}_{PF}$, $\Sigma_{PFK} = \Sigma_{PF} \cup \{+, :=, \downarrow, \uparrow\}$, $\Lambda_{PFK} \supset \Lambda_{CD}$.

Визначені наступні операції над атрибутами: +(a, b, c) – додавання аргументів b і c, результат зберігається у a; :=(a, b) –

присвоєння значення b змінній a; ↓(a, b) – додає елемент b у стек a; ↑(a, b) – дістає верхній елемент із стеку a, зберігає його значення у b.

Інформаційне забезпечення Λ_{PFK} збагачено наступними положеннями.

Атрибути терміналів і нетерміналів поточної форми: nBrackets – поточна кількість відкритих фігурних дужок, відповідна кількість закритих дужок буде додана атрибутом $text \downarrow ESS$ для успішної компіляції коду програми; closingBrackets – стек лічильників відкритих фігурних дужок, верхній елемент відповідає поточній області видимості.

Початкові умови конструювання: послідовність (масив) генів, які кодують частини алгоритмів сортування і допоміжні операції.

Умови завершення конструювання: сформована програма сортування (поточна форма не містить нетерміналів).

Перше правило $\psi_1 = \langle \langle s_{1,1}, s_{1,2} \rangle, \langle g_{1,1}, g_{1,2} \rangle \rangle$ виконує підстановку постійної частини коду, однакової для усіх програм. Операції над атрибутами не виконуються.

$$\begin{gathered} s_{1,1} = \langle \sigma \to \gamma \rangle; \\ s_{1,2} = \langle \alpha \to text \downarrow T_s \cdot \alpha \cdot text \downarrow T_e \rangle; \\ g_{1,1} = \langle \varepsilon \rangle;\ g_{1,2} = \langle \varepsilon \rangle \end{gathered} \quad (7)$$

Для α будуть продовжені підстановки.

Наступне правило ψ_2 містить відношення підстановки програмного коду для гена PUA.

$$\begin{gathered} s_{2,1} = \langle \sigma \to \alpha \rangle; \\ s_{2,2} = \langle \alpha \to text \downarrow T_{PUA} \cdot \alpha \rangle; \\ g_{2,1} = \langle \varepsilon \rangle; \\ g_{2,2} = \langle \downarrow (\text{closingBrackets}, 0) \rangle \end{gathered} \quad (8)$$

Операції над атрибутами додають новий елемент до стеку лічильників фігурних дужок. Наступні операції над атрибутами, які збільшують лічильник фігурних дужок, будуть збільшувати останню додану змінну.

Правило ψ_3 містить відношення підстановки програмного коду для гена ESS.

$$\begin{gathered} s_{3,1} = \langle \sigma \to \alpha \rangle; \\ s_{3,2} = \langle \gamma \to text \downarrow T_{ESS} \cdot \gamma \rangle; \\ g_{3,1} = \langle \varepsilon \rangle; \\ g_{3,2} = \langle \uparrow (closingBranckets, nBrackets) \rangle \end{gathered} \quad (9)$$

Програмний код, що кодується геном ESS, виконує додавання фігурних дужок, що закриваються у кількості, рівній значенню атрибута nBrackets. Це потрібно для закриття областей видимості, відкритих додаванням попередніх фрагментів тексту програм.

Правило ψ_4 містить відношення підстановки для додавання програмного коду, що відповідає генам, які відкривають нову область видимості і збільшують лічильник дужок, які необхідно закрити. Для мінімізації кількості правил позначимо символом Ter^* будь-який терміналген із множини {FSB, FSS, SEEI, SEI, BSB, BSF, P, QP, PM, SR, BLQP, BLPM, BGQP, BGPM}.

$$\begin{gathered} s_{4,1} = \langle \gamma \to Ter^* \cdot \gamma \rangle; \\ s_{4,2} = \langle \alpha \to text \downarrow Ter^* \cdot \alpha \rangle; \\ g_{4,1} = \langle \varepsilon \rangle; \\ g_{4,2} = \langle +(top \downarrow \text{closingBrackets}, top \downarrow \text{closingBrackets}, 1) \rangle \end{gathered} \quad (10)$$

Правило ψ_5 містить відношення підстановки для додавання програмного коду, що відповідає генам, які не відкривають нову область видимості. Позначимо символом Ter^{**} будь-який терміналген із множини {SIS, ML, MR, FS}.

$$\begin{gathered} s_{5,1} = \langle \gamma \to Ter^{**} \cdot \gamma \rangle; \\ s_{5,2} = \langle \alpha \to text \downarrow Ter^{**} \cdot \alpha \rangle; \\ g_{5,1} = \langle \varepsilon \rangle; \\ g_{5,2} = \langle \varepsilon \rangle \end{gathered} \quad (11)$$

Результатом конструювання функції сортування буде не тільки текст безпосередньо функції сортування, а й текстове представлення коду всього файлу реалізації (.cpp). Відповідний заголовковий файл (.h) не змінюється залежно від хромосоми і створюється заздалегідь. Він включає оголошення функції сортування, що конструюється і має наступний вигляд:

```
#pragma once
void ConstructedSort(int* arr, int n);
```

Відповідний файл реалізації складається із двох частин: незмінної частини і реалізації функції сортування, що конструюється із хромосоми. У незмінній частині включається вищезгаданий заголовковий файл та заголовкові файли із допоміжними утилітами і типами даних. Оголошуються допоміжні структури і функції. І початок саме реалізації функції сортування, що конструюється, де оголошуються усі змінні-атрибути із початковими значеннями. Початкова незмінна частина файлу реалізації, яка відповідає терміналу T_S, має наступний вигляд (код у статті відрізняється форматуванням від коду програми, для економії місця):

```
// text ↲ T_S:
#include <algorithm>
#include <iostream>
#include <stack>
#include <vector>
#include "Sort/Sort.h"
#include "Utilities/Utilities.h"
void Merge(int* arr, int size1, int size2) {
std::vector<int> tempArray1;
tempArray1.assign(arr, arr + size1);
int indexOfMergedArray = 0; // Index of the first element in the merged array
int indexOfSubArrayOne = 0; // Index of the first element in the first sub-array
int indexOfSubArrayTwo = 0; // Index of the first element in the second sub-array
while (indexOfSubArrayOne < size1 && indexOfSubArrayTwo < size2) {
    if (tempArray1[indexOfSubArrayOne] < arr[ size1 + indexOfSubArrayTwo]) {
        arr[indexOfMergedArray] = tempArray1[indexOfSubArrayOne];
        indexOfSubArrayOne++;
        } else {
        arr[indexOfMergedArray] = arr[ size1 + indexOfSubArrayTwo];
        indexOfSubArrayTwo++; }
    indexOfMergedArray++; }
while (indexOfSubArrayOne < size1) {
    arr[indexOfMergedArray] = tempArray1[indexOfSubArrayOne];
    indexOfSubArrayOne++; indexOfMergedArray++; }
while (indexOfSubArrayTwo < size2) {
    arr[indexOfMergedArray] = arr[size1 + indexOfSubArrayTwo];
    indexOfSubArrayTwo++; indexOfMergedArray++; }}
template <typename T>
struct SubarrayInfo {
    T* Arr = nullptr;
    int Size1;
    int Size2; };
void ConstructedSort(int* arr, int n) {
    const int minBlockSize = 32, maxBlockSize = 10000;
    std::stack<SubarrayInfo<int>> subarraysToMerge;
    std::stack<std::pair<int*, int>> unsortedArrays;
    int size1 = 0, size2 = 0, start = 1, end = 0, localSize = 0;
    bool swapped = false;
    int s = 0, e = n – 1, idx = 0;
    bool isSorted = false;
    auto pivotIdx = (s + e) / 2;
    int min, max;
```

Текст програми відповідний поточній хромосомі буде додано між початковою і кінцевою незмінними частинами. Далі представлено кінцеву незмінну частину, якій відповідає термінал T_E, у якій гарантовано відбувається злиття усіх відсортованих частин масиву і закриття останньої фігурної дужки функції.

```
// text ↲ T_E:
while(!subarraysToMerge.empty()) {
```

```
        const auto& stm =
subarraysToMerge.top();
        subarraysToMerge.pop();
        Merge(stm.Arr, stm.Size1, stm.Size2);
    }}
```

У результаті буде сформований готовий текст програми. Можливе як додавання файлів із сформованим кодом сортування до файлів програми, так і компіляція, і збірка окремої мінімалістичної бібліотеки.

Для перевірки спроможності адаптованих алгоритмів конкурувати з відомими алгоритмами сортування виконані експерименти із застосуванням запропонованого конструктивно-продукційного підходу на основі генетичного алгоритму.

Експериментальні дослідження

Експерименти проводились над даними різної структури:

- повністю відсортовані дані;
- дані, відсортовані у зворотному порядку;
- повністю відсортований масив, у якому виконується декілька перестановок пар елементів, вибраних випадковим чином;
- відсоток невідсортованості. У повністю відсортованому масиві випадковим чином обирається певна кількість послідовних ділянок елеменів сумарною довжиною рівною даному відсотку довжини усього масиву. В межах кожної ділянки випадково обрані елементи міняються місцями певну кількість разів;
- повністю випадкові дані.

Проведені експерименти із вищезазначеними типами даних обсягом один мільйон і десять мільйонів елементів.

Для кожної популяції генетичного алгоритму генерувався 51 масив сортованих даних відповідної структури. Вони зберігались у 51 бінарний файл. Кожний індивід популяції, який представляє собою конкретний алгоритм сортування, вичитував масив даних із файла. Виконувалось сортування, перевірялось, чи сортування виконане успішно і чи зберігався результат – час виконання – у масив. Після сортування усіх масивів вхідних даних вираховувався медіанний час і зберігався у окремий бінарний файл. Таким чином використовувались ідентичні дані для кожного індивіда.

Під час формування наступної популяції, 20% хромосом переносились з попередньої, 40% - додавалися після схрещування, решта заповнювалась новими хромосомами.

Задля обмеження довжини хромосоми і, відповідно, довжини генерованої функції сортування, процес генерації хромосоми керувався за допомогою наступних параметрів. Параметр MGBS = 1 визначає мінімальну кількість генів, необхідну для того, щоб стали доступними для вибору гени розбиття, які додають нові вузли-нащадки до поточного вузла. Максимальна глибина дерева-хромосоми MD = 3. На початку генерації хромосоми для кореневого вузла глибина дорівнювала одиниці. При виборі гена розбиття масиву до поточного вузла додавались два вузли-нащадки із глибиною більшою, ніж у поточного вузла на одиницю. У разі глибини меншій, ніж максимальна, правило вибору гена фінального сортування недоступне. Натомість доступні правила вибору генів розбиття. При досягненні максимальної глибини замість гену розбиття масива підставляється ген фінального сортування і вузли нащадки не додаються. Таким чином будується збалансоване дерево-хромосома із заданою глибиною.

Результати експериментів наведені у табл. 2. У рядку із відповідним типом даних представлений кращий із загальновідомих алгоритмів і кращий із конструйованих (адаптованих).

Для повністю відсортованих даних покращення становить 35-37% порівняно з кращим із загальновідомих алгоритмів, а саме сортуванням вставками. Для даних відсортованих у зворотному порядку, конструйовані алгоритми показали найвищий рівень покращення – 92-94%. Це досягається наявністю гена SR на початку хромосоми.

Таблиця 2. Результати експериментів із сортування

Обсяг сортованих даних	Тип вхідних даних	Алгоритм	Медіанний час, мкс	Покращення/Погіршення, %
Один мільйон	Повністю відсортовані	BSB_FSB_FSB_P_PUA_SEEI_MR_P_PUA_BGQP_BSB_BSF_FS_ESS_PUA_QP_FSB_BLQP_SEEI_MR_FS_ESS_ESS_PUA_BSF_BSF_QP_SEI_ML_SIS_PUA_FSB_FS_ESS_PUA_QP_SEI_ML_FS_ESS_ESS_ESS	329	37.33%
		InsertionSort	525	
	Відсортовані у зворотному порядку	SR_SEI_ML_FSS_BGPM_SIS_PUA_BSF_BSF_BLPM_FSB_P_PUA_SEI_ML_FSS_QP_FS_ESS_PUA_BSF_SEEI_MR_SEEI_MR_FS_ESS_ESS_PUA_BSF_QP_P_PUA_SEEI_SEEI_MR_BSF_SEI_ML_FS_ESS_PUA_FSB_SEI_ML_FS_ESS_ESS_ESS	896	92.7%
		QuickSort	12276	
	Декілька перестановок	InsertionSort	2562	-484%
		P_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_ESS	14985	
	Відсоток невідсортованих елементів	P_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_ESS	17847	3.41%
		QuickSort	18475	
	Випадкові	P_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_ESS	61831	0.24%
		QuickSort	61980	
Десять мільйонів	Повністю відсортовані	BSF_SEEI_MR_SIS_PUA_SEEI_MR_BGQP_BSF_P_PUA_BSF_BSB_BSF_FSS_FS_ESS_PUA_SR_FS_ESS_ESS_PUA_SEEI_MR_PM_SIS_PUA_QP_FS_ESS_PUA_BLPM_SEEI_SEEI_MR_SR_FS_ESS_ESS_ESS	3706	35.06%
		InsertionSort	5707	
	Відсортовані у зворотному порядку	SR_BSF_P_PUA_FSB_SIS_PUA_FSS_BSB_FSB_QP_FS_ESS_PUA_FSB_SEI_SEEI_ML_MR_QP_FS_ESS_ESS_PUA_BSF_SR_P_PUA_SR_BSF_BLQP_FSS_FS_ESS_PUA_BSF_BSF_BSF_SEI_ML_FS_ESS_ESS_ESS	8364	94.07%
		QuickSort	141151	
	Декілька перестановок	P_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_ESS	193621	0.137%
		QuickSort	193887	
	Відсоток невідсортованих елементів	P_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_ESS	201806	0.033%
		QuickSort	201872	
	Випадкові	SIS_PUA_SIS_PUA_FS_ESS_PUA_FS_ESS_ESS_PUA_SIS_PUA_FS_ESS_PUA_FS_ESS_ESS_ESS	715804	0.322%
		QuickSort	718110	

Сортування масивів із кількома перестановками обсягом один мільйон елементів показало значне погіршення – 484%. Для даних обсягом десять мільйонів наявне незначне покращення – 0.137%. Варто зазначити, що на масивах обсягом один мільйон сортування вставками показало значну ефективність, однак для десяти мільйонів елементів кращим із загальновідомих виявилось швидке сортування.

Для експериментальних даних із відсотком невідсортованих елементів покащення становить 0.33% (десять мільйонів) і 3.41% (один мільйон).

Покращення для сортування випадкових даних обсягом один мільйон становить 0.24%. Конструйований алгоритм включає гени розбиття із перестановкою елементів, що відповідають частині логіки швидкого сортування. Для даних обсягом десять мільйонів розраховане покращення 0.32%. Конструйонавий алгоритм включає гени розбиття, які відповідають частинам сортування злиттям.

Висновки

У попередніх роботах була запропонована конструктивна модель хромосоми деревовидної структури яка кодує алгоритм сортування. Викладена спеціалізація та конкретизація конструктору формування хромосом.

У цій статті як логічне продовження попередньої роботи розроблено підхід до декодування хромосом у послідовність генів і їх подальшого перетворення на текст програми мовою програмування.

Застосування конструктивно-продукційного моделювання є ефективним підходом до побудови алгоритмів, здатних адаптивно формувати структуру рішення на основі чітко визначених продукційних правил та конструктивних принципів. Запропонований метод дав змогу системно описати процес створення алгоритмів як послідовність перетворень, у межах яких кожне правило робило внесок у формування цілісного алгоритму сортування.

Продемонстровано ефективність застосування генетичного алгоритму для структурної адаптації конструйованих алгоритмів сортування і вибору найбільш пристосованого до заданих умов використання. Проведені експерименти підтвердили, що еволюційний підхід здатний адаптивно вдосконалювати структуру алгоритму, поступово підвищуючи його продуктивність.

Література

1. R. Sedgewick, ‘Implementing quicksort programs’, Communications of the ACM, 21(10), 847–857 (1978).
2. D. E. Knuth. Sorting by Exchanging. The Art of Computer Programming. – Vol. 3: Sorting and Searching. – 1st ed. – Addison-Wesley. – pp. 110–111. – 1973
3. J. W. J. Williams, Algorithm 132 (heapsort), Communications of the ACM, 7, 347–348 (1964)
4. C. Song and H. Li, “Improvement of Counting Sorting Algorithm,” in Journal of Computer and Communications, vol. 11, pp. 12–22, 2023
5. J. Wassenberg and P. Sanders, “Engineering a multi-core radix sort,” in Proc. Euro-Par 2011, Part II, LNCS 6853, Bordeaux, France, 2011, pp. 160–169. doi: 10.1007/978-3-642-23397-5_16.
6. N. Faujdar and S. Saraswat, “The detailed experimental analysis of bucket sort,” in Proceedings of the 7th International Conference on Cloud Computing, Data Science & Engineering – Confluence, IEEE, pp. 12–13, Jan. 2017
7. B. Haeupler, R. Hladík, J. Iacono, V. Rozhoň, R. E. Tarjan and J. Tětek, “Fast and Simple Sorting Using Partial Information,” in Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 3953–3973, 2025, doi: 10.1137/1.9781611978322.134
8. M. Subramaniam, T. Tripathi and O. Chandraumakantham, “Cluster Sort: A Novel Hybrid Approach to Efficient In-Place Sorting Using Data Clustering,” in IEEE Access, vol. 13, pp. 74359–74374, 2025.
9. В.І. Шинкаренко, А.Ю. Дорошенко, О.А. Яценко, В.В. Разносілін, К.К. Галанін, Двокомпонентні алгоритми сортування, Проблеми програмування, 2022, № 3-4. С. 32-41. doi: 10.15407/pp2022.03-04.032.
10. В.І. Шинкаренко, О.В. Макаров, Дослідження ефективності деяких детермінованих методів передобробки сортування даних, Проблеми програмування, 2023, № 4, С. 3-14. doi: 10.15407/pp2023.04.003.
11. V.I. Shinkarenko, O.V. Makarov, Structural Adaptation of Sorting Algorithms Based on Constructive Fragments, CEUR Workshop Proceedings. – Vol. 3806. – pp. 16–29. – 2024

12. В.І. Шинкаренко, О.В. Макаров Конструктивно-продукційне моделювання хромосом генетичного алгоритму з закодованими алгоритмами сортування, Проблеми програмування, 2025, № 3, С. 39-52. doi: 10.15407/pp2025.03.039
13. D. Beasley, D. R. Bull and R. R. Martin, "An Overview of Genetic Algorithms: Part 1, Fundamentals," in University Computing, vol. 15, pp. 58–69, 1993
14. В.І. Шинкаренко, В.М. Ільман. Конструктивно-продукційні структури та їх граматичні інтерпретації. I. Узагальнена формальна конструктивно-продукційна структура . Кібернетика і системний аналіз. – 2014. – Т. 50, № 5. – С. 8–16

References

1. R. Sedgewick, 'Implementing quicksort programs', Communications of the ACM, 21(10), 847–857 (1978).
2. D. E. Knuth. Sorting by Exchanging. The Art of Computer Programming. – Vol. 3: Sorting and Searching. – 1st ed. – Addison-Wesley. – pp. 110–111. – 1973
3. J. W. J. Williams, Algorithm 132 (heapsort), Communications of the ACM, 7, 347–348 (1964).
4. C. Song and H. Li, "Improvement of Counting Sorting Algorithm," in Journal of Computer and Communications, vol. 11, pp. 12–22, 2023
5. J. Wassenberg and P. Sanders, "Engineering a multi-core radix sort," in Proc. Euro-Par 2011, Part II, LNCS 6853, Bordeaux, France, 2011, pp. 160–169. doi: 10.1007/978-3-642-23397-5_16.
6. N. Faujdar and S. Saraswat, "The detailed experimental analysis of bucket sort," in Proceedings of the 7th International Conference on Cloud Computing, Data Science & Engineering – Confluence, IEEE, pp. 12–13, Jan. 2017
7. B. Haeupler, R. Hladík, J. Iacono, V. Rozhoň, R. E. Tarjan and J. Tětek, "Fast and Simple Sorting Using Partial Information," in Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 3953–3973, 2025, doi: 10.1137/1.9781611978322.134
8. M. Subramaniam, T. Tripathi and O. Chandraumakantham, "Cluster Sort: A Novel Hybrid Approach to Efficient In-Place Sorting Using Data Clustering," in IEEE Access, vol. 13, pp. 74359–74374, 2025.
9. V. I. Shinkarenko, A. Yu. Doroshenko, O. A. Yatsenko, V. V. Raznosilin, K. K. Galanin. Bicomponent sorting algorithms. Problems of Programming. – 2022. – No. 3–4. – pp. 32–41. – DOI: 10.15407/pp2022.03-04.032
10. V. I. Shinkarenko, O. V. Makarov. Study of the effectiveness of some deterministic preprocessing methods of data sorting. Problems of Programming. – 2023. – No. 4. – pp. 3–14. – DOI: 10.15407/pp2023.04.003
11. V.I. Shinkarenko, O.V. Makarov, Structural Adaptation of Sorting Algorithms Based on Constructive Fragments, CEUR Workshop Proceedings. – Vol. 3806. – pp. 16–29. – 2024
12. V. I. Shinkarenko, O. V. Makarov. Constructive-synthesizing modeling of the genetic algorithm chromosomes with encoded sorting algorithms, Problems of Programming. – 2025. – No. 3. – pp. 39–52. – DOI: 10.15407/pp2025.03.039
13. D. Beasley, D. R. Bull and R. R. Martin, "An Overview of Genetic Algorithms: Part 1, Fundamentals," in University Computing, vol. 15, pp. 58–69, 1993
14. V. I. Shynkarenko, V. M. Ilman. Constructive-Synthesizing Structures and Their Grammatical Interpretations. I. Generalized Formal Constructive-Synthesizing Structure. Cybernetics and Systems Analysis. – Vol. 50, No. 5. – pp. 8–16

Дата першого надходження до видання: 09.03.2026
Внутрішня рецензія отримана: 13.03.2026
Зовнішня рецензія отримана: 14.03.2026
Дата прийняття статті до друку: 19.03.2026
Дата публікації: 16.04.2026

Про авторів:

Шинкаренко Віктор Іванович,
доктор технічних наук, професор
Shynkarenko Victor,
Ph.D (doctor), professor
https://orcid.org/0000-0001-8738-7225
E-mail: shinkarenko_vi@ua.fm

Макаров Олексій Вікторович, аспірант
Makarov Olexiy, Post-graduate student
https://orcid.org/0009-0003-0921-155X
E-mail: makarovov@hotmail.com

Місце роботи авторів:

Український державний університет науки і технологій,
49010, Україна, Дніпро,
вул. академіка Лазаряна, 2.
Ukrainian State University of Science and Technology (Dnipro)
E-mail: office@ust.edu.ua

УДК 004.41:004.05 https://doi.org/10.15407/pp2026.01.051

Ю.В. Єрофеєв, І.П. Сініцин

ПРОБЛЕМИ ЕФЕКТИВНОГО ТЕСТУВАННЯ СПЕЦІАЛЬНИХ ПРОГРАМНИХ ЗАСОБІВ ВБУДОВАНИХ СИСТЕМ ТА ЇХ ОПРАЦЮВАННЯ

Розглянуто проблему результативного й ресурсно ефективного тестування спеціальних програмних засобів (СПЗ) вбудованих систем (ВС) в умовах істотної різнорідності й швидкої еволюції апаратних платформ, мінливості конфігурацій і скорочення циклів еволюції СПЗ. Показано, що для сучасних вбудованих виробів характерні часткова незалежність життєвих циклів апаратної та програмної складових і зростання ролі керованої модернізованості СПЗ ВС. Обґрунтовано потребу в таких умовах не лише виявлення дефектів, а й додаткових функцій тестування: прогнозування (не)задовільної якості СПЗ і ВС загалом, підтримку рішень щодо інтеграції СПЗ і накопичення повторно використовуваних тестових активів. Узагальнено особливості типових архітектур ВС і підходів до конструювання СПЗ ВС, значущі для забезпечення вищезазначених додаткових функцій тестування, зокрема, рамкову та мікросервісну архітектури, розроблення, кероване тестами, гнучкі методології, розроблення на основі моделей, а також підходи безперервної інтеграції й безперервного тестування. Показано роль XiL-підходів, керування конфігураціями та контролювання зв’язків між артефактами для забезпечення відтворюваності середовища тестування. Запропоновано розглядати стале повторне використання робочих продуктів тестування в руслі парадигми лінійок програмних продуктів як основу для сталого підвищення зрілості процесів тестування СПЗ ВС та їх конструювання загалом.

Ключові слова: вбудована система, спеціальні програмні засоби, тестування, повторне використання, лінійка програмних продуктів, безперервна інтеграція, безперервне тестування, життєвий цикл, програмна архітектура, керування конфігурацією.

Yu.V. Yerofieiev, I.P. Sinitsyn

THE PROBLEMS OF EMBEDDED SYSTEMS SPECIAL SOFTWARE EFFECTIVE TESTING AND THEIR ELABORATION

The paper examines the problem of sustainable testing of special software for embedded systems under conditions of rapid hardware evolution, configuration variability, and shortened update cycles. It shows that modern embedded products are characterized by a partial separation of hardware and software life cycles, a growing role of controlled firmware updates, component reuse, and the integration of computer vision and artificial intelligence algorithms. It is substantiated that, in such conditions, testing should not be limited to defect detection, but should also support quality forecasting, integration decisions, and the accumulation of reusable testing assets. The study summarizes typical embedded-system architectures and approaches to constructing special software, including layered and microservice-oriented architectures, test-driven development, agile methods, model-based development, and continuous integration, continuous testing, quality assurance, and security practices. The role of XiL approaches, configuration management, and artifact traceability in ensuring environment reproducibility is highlighted. Sustainable reuse of testing work products within the software product line paradigm is proposed as a basis for increasing the maturity of development and testing processes for special embedded software.

Keywords: embedded system, special software, testing, reuse, software product line, continuous integration, continuous testing, life cycle, software architecture, configuration management.

Вступ

У сучасних вбудованих системах (ВС) спеціальні програмні засоби (СПЗ) є одним із найскладніших, найуразливіших і водночас найбільш критичних складників [1, 2]. На відміну від програмних засобів загального призначення, СПЗ ВС функціонують в умовах жорстких ресурсних обмежень, вимог реального часу або близьких до нього, обмежень енергоспоживання, а також тісної залежності від апаратної платформи й фізичного оточення через взаємодію з датчиками та виконавчими механізмами. У зв'язку з цим дефекти СПЗ можуть спричиняти не лише функціональні збої, а й відмови ВС загалом, порушення вимог безпеки, загрози й втрати для довкілля та життя й здоров'я людини.

За цих умов тестування СПЗ ВС має, крім традиційного виявлення дефектів СПЗ, виконувати ще й додаткові функції:

а) надавати адекватні дані для (не)кількісного прогнозування якості СПЗ і ухвалення рішень щодо їх інтеграції з операційною системою, процесором, апаратними засобами;

б) підтримувати прогнозування вірогідних технічних та організаційних проблем у життєвому циклі (ЖЦ) ВС;

в) забезпечувати накопичення тестових артефактів і допоміжних даних для постійного вдосконалення процесів тестування й конструювання СПЗ в умовах обмежених ресурсів.

Разом функції а)-в) зумовлюють зростання актуальності проблеми сталого повторного використання робочих продуктів тестування в ЖЦ СПЗ і ВС. загалом. Особливо важливою вона є для організацій-розробників, що створюють ВС різних типів у різних предметних областях і не обмежуються однією апаратною та програмною платформою. Саме розв'язання цієї проблеми створює підґрунтя для узгодженого досягнення вищезазначених цілей тестування та зрештою підвищення зрілості процесу конструювання ВС.

Метою статті є окреслення підходу до сталого повторного використання робочих продуктів тестування СПЗ ВС у рамках парадигми лінійок програмних продуктів (Software Product Lines), зумовленого особливостями ЖЦ сучасних ВС: різнорідністю апаратно-програмних платформ, стислими термінами, обмеженими обчислювальними ресурсами.

1. Особливості СПЗ ВС, значущі для їх тестування

Типові архітектури ВС і СПЗ ВС. Детальний аналіз сучасних ВС у різних предметних областях [2] висвітлює три узагальнені типи їхньої архітектури: рамкову (reference), із застосуванням шару абстракції обладнання та мікросервісну.

В історично першій *рамковій* архітектурі ВС (рис. 1) СПЗ безпосередньо взаємодіє з драйверами пристроїв та операційною системою, які в свою чергу взаємодіють з апаратними засобами [2, 3].

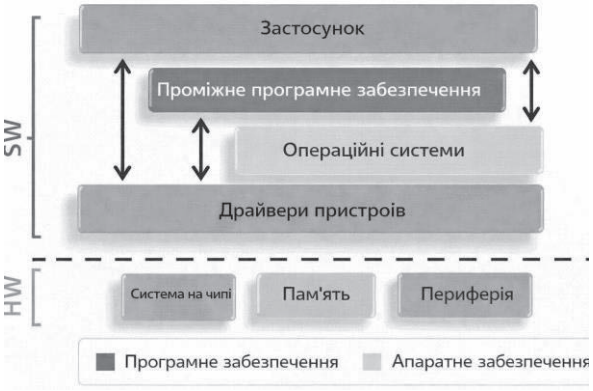

Рис. 1. Рамкова архітектура ВС

Подальший розвиток апаратних засобів зумовив поширення прошарованої архітектури [2] для СПЗ ВС, що являють собою монолітну збірку (так звану прошивку (firmware)). Ці СПЗ подано переліком програмних шарів. Кожний шар має принаймні один інтерфейс щодо суміжного шару, визначений його типом. Інтерфейси надають узгоджений доступ до функцій СПЗ, приховують внутрішні деталі їх реалізації та можуть виступати як обгортки для інтеграції несумісного коду між шарами [1].

Найпоширенішим і водночас найпрактичнішим прошарком прошивки є шар *абстракції обладнання* (hardware abstraction level) [1]. По суті це прикладний програмний інтерфейс для взаємодії з облад-

нанням, який замінює доступи на рівні апаратних засобів викликами функцій вищого рівня (рис. 2).

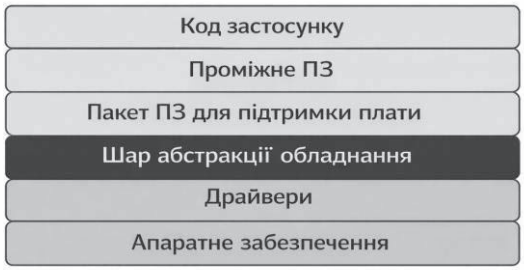

Рис. 2. Архітектура ВС з використанням шару абстракції обладнання

Саме він має забезпечити переносність і повторне використання СПЗ ВС та його позитивні ефекти: зниження вартості конструювання СПЗ і ВС загалом; підвищення рівня абстракції; зменшення кількості помилок завдяки багаторазовій експлуатації; спрощення масштабування, зокрема, перенесення на нові апаратні засоби й ВС у межах одного сімейства.

Нарешті для опрацювання обмежень розглянутих архітектур ВС протягом останньої декади набула активного розвитку мікросервісна архітектура (рис. 3).

Рис. 3. Мікросервісна архітектура ВС на базі вбудованої операційної системи

Мікросервісна архітектура подає користувацький застосунок множиною взаємодіючих автономних мікросервісів, зорієнтованих на його прикладні функції. Типова реалізація мікросервісу у ВС відповідає окремому процесу або задачі/набору задач ОС із власними ресурсами та механізмами ізоляції, якщо вони доступні. Концептуально мікросервіс містить код сервісу, вхідну чергу повідомлень, механізми надсилання вихідних повідомлень, логування та індикатори стану для моніторингу працездатності.

Мікросервісна архітектура є дефакто стандартною для гнучких методологій розроблення СПЗ, практик DevSecOps і конвеєра безперервної інтеграції й доставки (CI/CD). Низький рівень зв'язності сервісів полегшує й пришвидшує процес тестування, спрощує масштабування та заміну сервісів без істотного порушення роботи всього СПЗ.

Однак, водночас із окресленими перевагами, ця архітектура підвищує складність проєктування та створює накладні витрати на комунікації та пам'ять; децентралізований обмін повідомленнями може ускладнювати досягнення детермінованої поведінки в реальному часі та збільшувати час відгуку. Повна незалежність розгортання сервісів не завжди має місце в ресурсно-обмежених ВС, особливо за відсутності розвиненого механізму керування або в спрощених операційних - середовищах.

Тому застосування мікросервісів у ВС потребує попереднього аналізу вимог, обмежень платформи та декомпозиції функцій СПЗ ВС.

Зовнішнє ділове середовище конструювання СПЗ ВС. Однією з визначальних особливостей зовнішнього ділового середовища конструювання ВС є зростання впродовж останнього десятиліття ролі Інтернету речей – класу розподілених ВС, здатних збирати й передавати дані мережею без безпосередньої участі - людини.

Ця тенденція зумовлює істотну різнорідність методів і технологій конструювання СПЗ ВС, які зручно розподілити за двома форматами:

а) «традиційним», згідно з яким СПЗ ВС розробляють із вузькоспеціалізованою функціональною метою для певної апаратної платформи й визначеного сценарію застосування;

б) «диверсифікованим», який передбачає конструювання низки мінливих СПЗ ВС (і, можливо, самих ВС як послугу повного циклу) для різних споживачів та/або предметних областей із використанням подібних між собою, але мінливих, апаратних платформ (рис. 4).

а)

б)

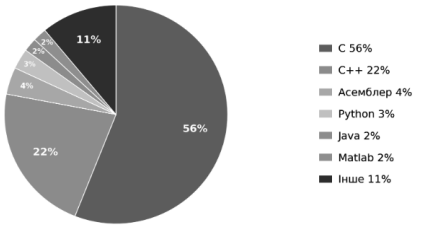

Рис. 4. Розподіл ВС: а) за предметними областями; б) за мовами програмування

Такі СПЗ не лише допускають певну повторюваність операцій конструювання як на низькому (драйвери, первинні завантажувачі, bootloaders) так і на вищих рівнях (модулі й бібліотеки), а й потребують її для підвищення якості та зрілості процесів конструювання. Основними мовами програмування для конструювання СПЗ ВС в обох форматах є C та C++. Однак розвиток сучасних архітектур СПЗ ВС, насамперед мікросервісної, зумовлює зростання частки інших МП, зокрема, Python і Java (рис. 4).

Внутрішнє ділове середовище конструювання СПЗ ВС. Для врахування описаних вище особливостей зовнішнього ділового середовища компанії-розробники активно вдосконалюють власні процеси та інструменти для забезпечення конкурентоспроможної якості СПЗ ВС.

«Традиційний» формат а), за якого СПЗ конструюють у де-факто спільному життєвому циклі (ЖЦ) з пристроєм, поступово вичерпує потенціал [1, 3]: він ускладнює швидке нарощування функцій, адаптацію до нових умов застосування та підтримку ВС упродовж тривалого строку експлуатації.

Ключовою ідеєю новітніх підходів, властивих «диверсифікованому» формату б), є організація відносно незалежних ЖЦ для СПЗ та апаратних засобів. Апаратна платформа еволюціонує за власним циклом (ревізії, компонентна база). Натомість СПЗ конструюють як постійно оновлюваний продукт: із можливістю регулярних випусків, підтримкою сумісності й контрольованими змінами без повного реінжинірингу СПЗ і тим паче ВС під кожне вдосконалення [1, 2].

Зокрема, у перспективній сфері автономних наземних комплексів вдосконалення процесів конструювання відбувається одночасно двома «траєкторіями» – апаратній і програмній, із різними темпами та обмеженнями.

Вирішальним чинником реалізації такого підходу стає гнучкість архітектури СПЗ і здатність до керованого оновлення як СПЗ загалом («прошивки»), так і окремих компонентів СПЗ – бібліотек, сервісів, модулів оброблення даних. Це дозволяє прийнятно швидко покращувати функції та якісні характеристики СПЗ, зберігаючи контроль над версіями, залежностями та конфігураціями в умовах мінливості платформ і сценаріїв використання.

Вдосконалювальні зміни стосуються двох взаємопов’язаних аспектів: організаційного (процесного) та інфраструктурного (технологічного).

До інфраструктурних тенденцій належать: удосконалення тестових стендів, зокрема, із підтримкою віддаленого доступу; розроблення та впровадження засобів автоматизації тестування СПЗ ВС у конфігураціях XiL (X-in-the-Loop), включно з MiL (Model-in-the-Loop), SiL (Software-in-the-Loop), PiL (Processor-in-the-Loop) та HiL (Hardware-in-the-Loop) [4], а також відповідних моделей середовища; наскрізна автоматизація тестів на різних рівнях; керування конфігураціями й артефактами повторного використання. Організаційно ключовими є перехід до безперервної інтеграції та регресійного тестування

(Continuous Integration (CI)), а також адаптація гнучких методів розроблення (Agile) до специфіки СПЗ ВС та обмежень апаратної платформи.

Доцільно розрізняти такі XiL-конфігурації:

– MiL (Model-in-the-Loop) – верифікація алгоритмів на рівні моделей;

– SiL (Software-in-the-Loop) – тестування виконуваного коду в симульованому середовищі;

– PiL (Processor-in-the-Loop) – виконання коду на цільовому процесорі з модельованими периферійними умовами;

– HiL (Hardware-in-the-Loop) – інтеграційне тестування із реальним апаратним компонентом у контурі.

Фактично конструювання СПЗ ВС перебуває в стані технологічної конкуренції, у межах якої постійно з'являються нові пристрої та конфігурації, а самі СПЗ безперервно оновлюються й ускладнюються (зокрема, бібліотеками й компонентами для реалізації алгоритмів комп'ютерного зору й штучного інтелекту). У цьому контексті дедалі більш запитаним стає підхід QAOps [5, 6] – сукупність організаційно-інфраструктурних практик, зорієнтована на спільну відповідальність за якість (Dev/QA/Ops), запровадження керованих критеріїв допуску змін (quality gates), забезпечення відтворюваності середовищ і накопичення повторно використовуваних тестових активів (зокрема, в межах лінійок програмних продуктів).

Актуальні особливості процесів ЖЦ СПЗ ВС і ВС. У таблиці 1 співставлено особливості процесів технічного керування та технічних процесів, що безпосередньо впливають на вимоги, архітектуру, інтеграцію та верифікацію/валідацію СПЗ у ЖЦ ВС, згідно з ДСТУ ISO/IEC/IEEE 12207, 1528.

Таблиця 1.

Особливості процесів ЖЦ СПЗ ВС

Процес ЖЦ	Об'єкт процесу		Особливість СПЗ ВС
	СПЗ	ВС	
Процеси технічного керування			
Керування ризиками (Risk Management)	Ризики системи: технічні, виробничі, експлуатаційні	Ризики ПЗ: дефекти, вразливості, технічний борг	Додаються ризики: обмежені ресурси, реальний час, залежність від платформи, кібербезпека, обмежений час доступності компонентів
Керування конфігурацією (Configuration Management)	Конфігурація виробу та варіантів: апаратна ревізія, BOM, інтерфейси	Конфігурація коду, збірок і релізів: версії, гілки, артефакти	Єдина ідентифікація конфігурації. Перелік (відомість) матеріалів програмного забезпечення. Специфікація складу апаратних компонентів
Управління інформацією (Information Management)	Артефакти системи: вимоги, архітектура, інтерфейси, протоколи випробувань	Артефакти ПЗ: код, збірки, тести, звіти	Простежуваність (traceability) артефактів від вимог до тестів і випусків/релізів
Забезпечення якості (Quality Assurance)	Забезпечення якості системи: відповідність вимогам та критеріям приймання	Забезпечення якості ПЗ: стандарти кодування, метрики, контроль дефектів	Атрибути якості: затримка, енергоспоживання, температурні режими, відмовостійкість, функціональна безпека та кібербезпека
Технічні процеси			
Визначення потреб і вимог заінтересованих сторін (Stakeholder Needs & Requirements)	Потреби користувачів/місії на рівні виробу	Потреби до ПЗ як складника системи	Уточнення потреб до автономності, сенсорів, зв'язку, експлуатаційних умов і обмежень

Визначення системних і програмних вимог (System/Software Requirements Definition)	Системні вимоги: функціональні та нефункціональні	Вимоги до ПЗ: функції, інтерфейси, обмеження	Інтерфейси з периферією, протоколи, обмеження пам'яті/CPU, вимоги до оновлення/відновлення, захищений завантажувач (secure boot)
Визначення архітектури (Architecture Definition)	Архітектура системи та розподіл функцій між підсистемами	Архітектура ПЗ: компоненти, інтерфейси, взаємодія	Код-дизайн апаратної і програмної частини , ізоляція безпечні межі взаємодії
Визначення проєктного рішення (Design Definition)	Проектування компонентів системи	Проектування модулів ПЗ	Проектування драйверів, станів і режимів
Системний аналіз (System Analysis)	Аналіз альтернатив і компромісів системного рівня	Аналіз рішень на рівні ПЗ	Компроміси: точність vs швидкодія, енергоспоживання vs продуктивність, пам'ять vs функціональність
Реалізація (Implementation)	Реалізація компонентів системи	Реалізація ПЗ, збірка та інтеграція залежностей	Крос-компіляція, оптимізації під платформу, захист ключів і секретів
Інтеграція (Integration)	Інтеграція на рівні системи	Інтеграція компонентів ПЗ	Робочий макет апартаного забеспечення, драйвери, інтеграція модулів комп'ютерного зору та AI-моделей; сумісність із ревізіями HW
Верифікація (Verification)	Перевірка відповідності вимогам	Верифікація ПЗ: огляди, аналіз, тестування	SiL/HiL, ін'єкція відмов (fault injection), fuzzing , аналіз покриття та часових характеристик
Перехід (Transition)	Передача в експлуатацію та/або виробництво	Реліз ПЗ та розгортання	Підписані образи, provisioning, OTA; перевірка у впливових умовах (температура/вібрації/зв'язок) і сценаріях відмов
Валідація (Validation)	Підтвердження придатності системи для потреб у цільовому середовищі	Валідація ПЗ в контексті системи	Польові випробування, acceptance-сценарії, оцінювання автономних функцій у репрезентативних умовах
Експлуатація (Operation)	Експлуатація системи	Експлуатація ПЗ	Телеметрія, логування, віддалена діагностика, контроль ресурсів і конфігурації
Супровід (Maintenance)	Супровід системи.	Супровід ПЗ.	Регулярні оновлення, підтримка та керування сумісністю варіантів.
Виведення з експлуатації (Disposal)	Виведення системи з експлуатації	Виведення ПЗ/даних з експлуатації	Безпечне стирання даних і ключів, deprovisioning; вимоги комплаєнсу (compliance)

Таблиця 1 надає процесне підґрунтя для проєктування стратегії тестування саме СПЗ ВС, визначення артефактів і меж відповідальності між рівнями СПЗ і ВС та ідентифікації специфічних для СПЗ ВС чинників – обмежених ресурсів, режиму реального часу, довготривалого супроводу, постійного оновлення та вимог - кібербезпеки.

Еволюція моделей ЖЦ СПЗ ВС, особливості якої підсумовано в табл. 2, зумовлена зростанням складності програмно-апаратної інтеграції, підвищенням вимог до швидкості внесення змін, а також необхідністю забезпечення простежуваності, надійності й безпеки на всіх рівнях розроблення. Якщо традиційнуV-модель зорієнтовано на послідовне проходження

етапів специфікації, проєктування, реалізації, верифікації та валідації, то подальший розвиток підходів пов'язаний із переходом до більш гнучких ітеративних схем, які скорочують цикл зворотного зв'язку між розробленням, інтеграцією та тестуванням.

Для сфери конструювання ВС така трансформація має особливе значення, оскільки програмні компоненти розробляють у тісному зв'язку з апаратним середовищем, обмеженнями цільової платформи та вимогами до функціональної безпеки й кібербезпеки.

Розвиток V-моделі до гнучких метолологій (Agile) і далі до Embedded DevSecOps відображає перехід від переважно послідовного та документно-орієнтованого підходу до безперервного, автоматизованого й ризик-орієнтованого керування ЖЦ ВС і СПЗ ВС. Якщо V-модель забезпечує високу формалізацію та простежуваність, то Agile підвищує адаптивність до змін, а Embedded DevSecOps поєднує гнучкість із безперервною інтеграцією, тестуванням, контролем безпеки та відповідності вимогам.

Таблиця 2.

Еволюція моделей ЖЦ СПЗ ВС

Модель	Ключова особливість	Вимоги	Інтеграція	Тестування СПЗ ВС	Безпека
V-модель	Послідовний, фазовий підхід	Фіксуються на початку	Переважно пізня	За рівнями верифікації та валідації	Окремий напрям робіт
Agile	Ітеративний та інкрементальний підхід	Уточнюються впродовж розроблення	Регулярна, поетапна	У межах ітерацій, з автоматизацією	Враховується в процесі
Embedded DevSecTestOps	Безперервний, автоматизований підхід	Динамічне керування і простежуваність	Безперервна (CI/CD/CT)	Безперервне, автоматизоване, багаторівневе	Інтегрована в повний цикл розробки

2. Актуальні методології конструювання СПЗ ВС

Одним із підходів до конструювання СПЗ ВС є адаптація принципів об'єктно-орієнтованого програмування до процедурної мови С. У цьому випадку інкапсуляція, композиція, обмежений поліморфізм та інші ООП-конструкції реалізують через структури даних, вказівники на функції, явні інтерфейси та усталені шаблони виклику. Такий підхід застосовують, коли необхідно забезпечити модульну організацію коду, повторне використання компонентів і формалізовану взаємодію між ними без переходу до повноцінного C++. У конструюванні СПЗ ВС це пов'язано з вимогами до їхнього функціонування й сумісності з ресурсними обмеженнями цільових платформ.

Подальший розвиток методологій конструювання СПЗ ВС пов'язаний із перенесенням акценту з побудови структури коду на керування процесом його поетапного формування. У [3] підхід Test-Driven Development (TDD) визначено як методологію, де реалізацію функціональності здійснюють через послідовність формування тестів, написання мінімально необхідного коду та подальший рефакторинг. За такого підходу тестування розглядають не як завершальний етап перевіряння, а як складову процесу конструювання програмної архітектури. Для ВС це означає можливість раннього виявлення дефектів, зниження складності налагодження та підвищення керованості змін у програмних компонентах.

У контексті СПЗ ВС TDD адаптують до апаратно-залежного середовища за

рахунок подвійно таргетованого тестування (dual-target testing), ізоляції залежностей від апаратної частини й операційної системи, а також використання тестових двійників (test doubles), зокрема, підробок (fakes), шпигунів (spies), імітацій (mocks). Це дає змогу перевіряти логіку модулів поза цільовою платформою, виконувати поетапне нарощування функціональності та підтримувати рефакторинг без безпосередньої залежності від наявності апаратного середовища. Таким чином, TDD у структурі сучасних методологій конструювання СПЗ ВС пов'язують з ітеративною розробкою, автоматизованим модульним тестуванням і поетапним формуванням архітектури СПЗ ВС.

Поряд із TDD у створенні ВС застосовують гнучкі (agile) методи розроблення, зорієнтовані на ітеративність, інкрементність і скорочення циклу зворотного зв'язку [7]. У межах такого підходу вимоги організують у журнал завдань (backlog), розробку виконують короткими ітераціями, а перевіряння ухвалених рішень інтегрують у поточний процес робіт. Для ВС це пов'язано з тим, що програмні складники зазвичай уточнюють одночасно з апаратною платформою, тому жорстко послідовні моделі не завжди забезпечують належну узгодженість між етапами розроблення.

Разом із тим у сфері СПЗ ВС agile-підходи зазвичай налаштовують з урахуванням вимог реального часу, апаратно-програмної розробки, обмеженості обчислювальних ресурсів і необхідності випробувань на обладнанні. На практиці це реалізують за рахунок окремих елементів, зокрема, коротких циклів планування, частішої інтеграції, раннього тестування та інтенсивної командної взаємодії. У такому трактуванні гнучкі методи розглядають як процесне підґрунтя для підвищення узгодженості між розробленням, інтеграцією та перевіркою програмних компонентів.

Іншим напрямом розвитку методологій конструювання СПЗ ВС є перенесення основного акценту з вихідного коду на модель системи. Model-based Development (MBD) [8, 9] визначають як підхід, де основним артефактом є модель, що описує структуру, поведінку, часові характеристики та взаємодію системи із зовнішнім середовищем. У межах цього підходу виконують моделювання, верифікаці. та раннє тестування до завершення реалізації на цільовій платформі. Надалі модель уточнюють до рівня, на якому можливе автоматизоване генерування програмного коду або інших реалізаційних артефактів.

Для ВС MBD пов'язано з інтеграцією етапів формування вимог, проєктування, тестування та реалізації в межах єдиного модельного подання. Застосування моделі як основного джерела опису системи забезпечує трасування вимог, повторне використання компонентів, аналіз поведінки та узгодження програмної й апаратної частин. Такий підхід застосовується під час розроблення складних кіберфізичних і багатодоменних систем, у яких поєднуються алгоритмічний опис, часовий аналіз, симуляція та автоматизоване генерування коду.

Наступний етап еволюції методологій конструювання СПЗ ВС пов'язаний з інтеграцією процесів розроблення, тестування, постачання та гарантування безпеки в межах безперервного циклу Embedded DevSecOps [3, 6], насамперед у конструюванні СПЗ ВС мовою С. Його поширення зумовлене апаратними обмеженнями цільових платформ, тривалим ЖЦ ВС, вимогами до надійності функціонування та необхідністю контролювання ризиків на пізніх етапах впровадження ВС.

В [1] сучасне проєктування ВПЗ розглянуто через архітектуру програмного забезпечення, процеси Agile і DevOps, а також практики розроблення. До складу відповідних інженерних практик віднесено проєктування безпечних застосунків, статичний аналіз, перевірка коду (code review), оцінювання покриття коду та використання CI/CD. У цьому контексті Embedded DevSecOps розглядають як процесну основу для інтеграції вимог безпеки та якості до повного циклу розроблення СПЗ ВС мовою С.

Окреме місце серед актуальних підходів займає фреймворк-орієнтоване розроблення СПЗ ВС на основі C++ і Qt. У цьому випадку C++ застосовують для реалізації прикладної логіки, системних серві-

сів і обчислювальних компонентів, тоді як комерційний фрейморк Qt забезпечує засоби конструювання інтерфейсу (насамперед Qt Quick і QML) організації взаємодії між програмними модулями (механізм signals and slots), оброблення подій і досягнення переносності між апаратно-програмними платформами. Такий підхід застосовують, зокрема, для ВС на базі вбудованої ОС Linux, де поєднують вимоги до продуктивності, модульної організації та супроводження СПЗ ВС. Тоді C++ і Qt становлять інструментальну основу для створення модульного та переносного СПЗ ВС, зокрема панелей керування, людино-машинних інтерфейсів та інших пристроїв із розвиненою візуальною складовою.

Актуальні методології конструювання СПЗ ВС охоплюють як підходи до структурної організації програмного коду, так і процесні, модельні та фреймворк-орієнтовані засоби розроблення. Їх застосуванню притаманне поєднання вимог до надійності, передбачуваності виконання, інтеграції з апаратною платформою, тестовності, безпеки та супроводженості СПЗ ВС.

У практиці Embedded DevSecOps безперервне тестування доцільно запроваджувати як багаторівневу систему перевірок, де окремі завдання можуть бути виконані послідовно або паралельно залежно від їхнього призначення та вартості виконання. До такої системи зазвичай належать автоматизоване збирання, версії СПЗ, модульне тестування, аналіз якості коду, перевірка залежностей, тестування інтеграції компонентів, а також спеціалізовані безпекові процедури, що доповнюють класичний CI/CD-конвеєр. Важливо, що результати кожного запуску накопичують фактичні дані про тривалість, успішність і чутливість тестових процедур, які стають підґрунтям для подальшої оптимізації CI/CD-конвеєра, раціонального розподілу тестових завдань і підвищення відтворюваності контролю якості. Таким чином, безперервне тестування в Embedded DevSecOps (рис. 5) слугує не лише засобом оперативного виявлення помилок, а й механізмом системного керування якістю, зокрема, та безпекою СПЗ ВС протягом усього ЖЦ ВС [6].

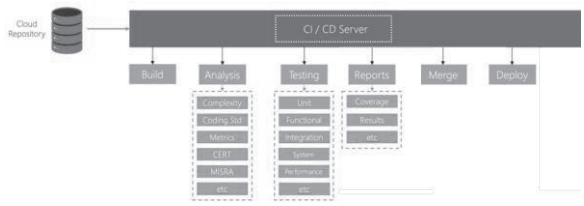

Рис. 5. Позиція безперервного тестування СПЗ ВС в Embedded DevSecOps

3. Особливості рівнів і видів тестування СПЗ ВС

Позиція тестування СПЗ ВС у життєвому циклі ВС. Рівні та види тестування спеціалізованого програмного забезпечення вбудованих систем доцільно розглядати як загальну методичну основу, успадковану від усталених практик тестування програмного забезпечення. У цьому підході рівні тестування охоплюють компонентний, інтеграційний, системний і приймальний рівні, тоді як види тестування описують аспект перевірки, зокрема, функційну та нефункційну відповідність. Однак для СПЗ ВС ця класифікація набуває прикладної специфіки: об’єкт тестування має бути оцінено не лише як програмний код, а й як складник програмно-апаратної системи, який функціонує у взаємодії з процесором, шинами обміну, сенсорами, виконавчими механізмами та часовими обмеженнями. Тому для СПЗ ВС тестові рівні фактично відображають не лише їхню логічну декомпозицію, а й ступінь наближення тестового оточення до реального середовища функціонування СПЗ.

Саме з цієї причини в ЖЦ ВС широко застосовують сімейство конфігурацій X-in-the-Loop (XiL) [4] як предметно-орієнтоване уточнення класичних рівнів тестування [5]. На ранніх етапах застосовують Model-in-the-Loop (MiL), де верифікують алгоритми та керувальну логіку на рівні моделей без реального апаратного виконання. Далі Software-in-the-Loop (SiL) дає змогу перевіряти згенерований або реалізований програмний код у середовищі розроблення, зіставляючи його поведінку з моделлю. Processor-in-the-Loop (PiL) переносить перевірку на цільовий процесор або його еквівалентний симулятор, що важливо для виявлення ефектів, пов’язаних із компі-

лятором, архітектурою процесора та часовими характеристиками виконання. Наступний етап Hardware-in-the-Loop (HiL), передбачає підключення реального контролера до моделі фізичного об'єкта, що виконується в реальному часі, і тому є ключовим засобом перевірки інтеграції програмної та апаратної частин. Таким чином, XiL-послідовність конкретизує, як саме класичні рівні тестування реалізуються в контексті СПЗ вбудованих систем.

Для складних кіберфізичних ВС (транспортних тощо) цю послідовність доповнюють конфігурації Driver-in-the-Loop (DiL) та Vehicle-in-the-Loop (ViL). У галузі автоматизованих транспортних засобів DiL-тестування зазвичай означає включення людини-оператора або водія до замкнутого контуру тестування, коли вже поведінку ВС загалом оцінюють не лише за формальними критеріями, а й з урахуванням реакції людини на сценарії керування, інтерфейси та динаміку об'єкта. VIL із свого боку поєднує реальний транспортний засіб із віртуальним середовищем. Його застосовують для зближення полігонних і віртуальних випробувань. У результаті конфігурації MiL, SiL, PiL, HiL, DiL і ViL слід трактувати не як альтернативу традиційним рівням тестування, а як ієрархію середовищ для перевірки, у межах якої послідовно зростає реалізм відтворення умов експлуатації і, відповідно, достовірність оцінювання властивостей СПЗ ВС. У цьому контексті концепція *зсуву ліворуч* (рис. 6) означає перенесення дій з тестування на максимально ранні стадії ЖЦ СПЗ, коли ще можна дешево усунути дефекти вимог, архітектури, моделей і програмної логіки.

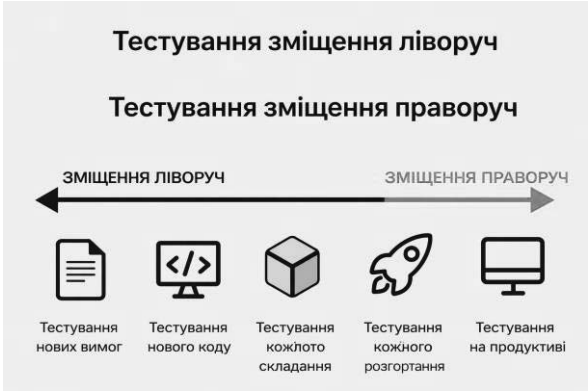

Рис. 6. Концепція зсуву ліворуч і зсуву праворуч у процесі тестування СПЗ ВС

Для СПЗ ВС це охоплює ранню верифікацію вимог, аналіз архітектурних рішень, MiL-перевірки, статичний аналіз, модульне тестування, а також SiL і PiL тестування до повної інтеграції з апаратними засобами. Натомість *зсув праворуч* акцентує потребу у продовженні перевірянь після інтеграції та розгортання, коли СПЗ ВС і ВС загалом демонструють свої критичні властивості вже в реальних або наближених до реальних умовах експлуатації. Для ВС це означає HiL, DiL, ViL тестування, аналіз телеметрії, експлуатаційний моніторинг, перевірка оновлень і довготривалі випробування стабільності. Отже, для СПЗ ВС зсув ліворуч і зсув праворуч доцільно розглядати як взаємодоповнювальні кращі практики, а XiL-конфігурації – як практичний механізм їх реалізації в безперервному контурі гарантування якості.

У контексті тестування СПЗ ВС доцільно розмежовувати стандартизовані техніки проєктування тестів і підходи типу чорної, білої та сірої скриньки. Згідно з ДСТУ ISO/IEC/IEEE 29119-3, 29119-4, до стандартизованих належать насамперед специфікаційно-орієнтовані, структурно-орієнтовані та досвідно-орієнтовані техніки. Їх застосовують під час проєктування та реалізації тестів на різних рівнях, зокрема, також і для СПЗ ВС з урахуванням апаратних обмежень, часових вимог та залежності від периферії.

Підходи фазингу [10, 11] та чорної й білої скриньки описуть не стандартну класифікацію технік тестування, а ступінь доступу тестувальника до внутрішньої структури об'єкта тестування: від перевірки лише зовнішньої поведінки до аналізу внутрішніх структур, покриття коду та проміжного комбінованого варіанта.

Серед цих технік *фазинг* слід розглядати як окрему спеціалізовану техніку динамічного автоматизованого тестування, а не як самостійний «рівень» тестування. Його сутність полягає в масованому поданні на вхід програми випадкових, спеціальним чином модифікованих або цілеспрямовано згенерованих даних для виявлення аварійних завершень, порушень роботи з пам'яттю та інших дефектів. У най-

простішому вигляді фазинг тяжіє до моделі чорної скриньки, але сучасні варіанти, зокрема, керований покриттям (coverage-guided) фазинг [10], фактично наближають його до сірої скриньки, оскільки передбачають внутрішній зворотний зв'язок щодо виконання СПЗ для спрямування генерації нових тестів. Для СПЗ ВС така техніка є особливо цінною, але водночас складною через тісну прив'язку СПЗ до регістрів, переривань і периферійних інтерфейсів.

У сучасних практиках тестування СПЗ ВС їх фазинг поєднують з емуляцією та автоматичним моделюванням периферії. У [12] прямо зазначено, що динамічне тестування та фазинг СПЗ ВС істотно обмежені апаратними залежностями й низькою масштабованістю, хоча запропонований авторами підхід дає змогу тестувати firmware у більш апаратно-незалежний спосіб.

Перспективні підходи до організації тестування СПЗ ВС. У практиці розроблення й тестування СПЗ ВС сформувалося кілька стійких центрів компетенцій, кожен з яких охоплює окремий клас інструментів, платформ і послуг, пов'язаних із забезпеченням якості, верифікацією та випробуванням.

Статичний аналіз і якість коду (Parasoft, IAR, MathWorks, Synopsys / Black Duck (Coverity)). Засоби цього напряму застосовують для тестування коду СПЗ без їх виконання, виявлення дефектів, потенційно небезпечних конструкцій, порушень стандартів програмування та підтвердження відповідності вимогам до надійності й безпеки.

HIL і випробувальні стенди (dSPACE, National Instruments). Компанії цього сегменту спеціалізуються на побудові апаратно-програмних стендів для випробувань у реальному часі, моделюванні середовищ функціонування та інтеграції програмних компонентів із реальними контролерами, платами й периферією.

Фазинг (Code Intelligence, Defensics). Методи та засоби цього класу зорієнтовано на автоматизоване формування великої кількості некоректних, граничних або несподіваних вхідних даних з метою виявлення відмов і вразливостей СПЗ ВС.

Новітні підходи: цифрові двійники та агентний штучний інтелект

Перспективним напрямом розвитку тестування СПЗ ВС є використання цифрових двійників як основи безперервного тестування. Цифровий двійник доцільно розглядати як виконувану модель СПЗ ВС та їх середовища, узгоджену з реальною конфігурацією за параметрами, інтерфейсами й сценаріями функціонування. Такий підхід дає змогу перенести значну частину регресійних тестів до автоматизованих конфігурацій із залученням реальних пристроїв. Практична реалізація зазвичай охоплює моделі середовища, бібліотеки (не)штатних і граничних сценаріїв, автоматизоване виконання в CI/CD-конвеєрі, а також валідування самого двійника шляхом калібрування та підтвердження прийнятної точності моделі.

Іншим новітнім підходом є застосування агентного штучного інтелекту для підтримки процесів тестування. Тоді агентний штучний інтелект виконує допоміжні функції в задачах, які складно масштабувати вручну: генерування тестових заготовок на основі вимог та інтерфейсів, адаптацію тестів у разі зміни конфігурацій і варіантів СПЗ ВС, аналіз журналів і логів виконання, а також добір сценаріїв для розширення покриття. Доцільним є також його застосування для поглиблення тестування безпеки, зокрема, під час підготування фазингу. Водночас у критичних і регульованих предметних областях такий підхід має бути застосований лише за умови контролю результатів, коли артефакти проходять формалізовані процедури перевірки, аналізу та аудиту з боку експерта в предметній області.

Перспективним є поєднання цифрового двійника з агентним штучним інтелектом в об'єднаному xiL-контурі безперервного тестування СПЗ. Тоді агент із функціями штучного інтелекту відбирає або, за потреби, формує релевантні сценарії, ініціює їх виконання на цифровому двійнику, узагальнює результати прогонів, готує

описи дефектів і підтримує актуальність тестового набору під час еволюції СПЗ ВС.

4. Авторський підхід до опрацювання проблем тестування СПЗ ВС

Зіставлення проаналізованих вище особливостей СПЗ ВС і процесу безперервного тестування СПЗ у SiL-конфігурації Embedded DevSecOps дозволяє уточнити проблеми ефективної організації цього процесу [13] на підтримку його додаткових функцій а)-в), зафіксованих у Вступі статті:

– ефективне керування артефактами тестування СПЗ, зокрема їх адміністрування й контрольоване повторне використання як у межах окремих циклів DevSecOps, так і між ними (P_1);

– забезпечення повноаспектного наскрізного простежування між самими артефактами тестування та між ними й іншими артефактами DevSecOps (P_2);

– адаптування перспективних процесів і технік динамічного тестування, регламентованих ДСТУ ISO/IEC/IEEE 29119-3, 29119-4, з урахуванням ускладнюючих особливостей СПЗ ВС (P_3);

– своєчасне ґрунтовне відстеження якості СПЗ ВС, насамперед кібер- і функціональної безпеки, прийнятної для подальшого тестування СПЗ на цільових платформах у конфігураціях PiL, HiL, ViL, DiL, її гарантування й засвідчення (P_4);

– ефективне розподілення й контролювання відповідальності й повноважень між різними командами в Embedded DevSecOps: конструювання, тестування, інформаційної безпеки, операційного персоналу (P_5);

– постійне (не)кількісне оцінювання рівня зрілості процесу безперервного тестування СПЗ і здобуття інформації щодо його вдосконалення (P_6).

Авторський підхід до опрацювання проблем P_1-P_6 поєднує:

а) конструктивну постановку спільної для P_1-P_6 *технічної задачі*, а саме: розроблення моделей і методів ефективного розгортання й використання безпечного конвеєра безперервного автоматизованого тестування СПЗ ВС у siL-конфігурації в Embedded DevSecOps для диверсифікованого конструювання мінливих СПЗ ВС;

б) засади розв'язання цієї задачі, підсумовані положеннями B_1–B_7;

в) першочергові кроки з реалізації підходу S_1.–S_7.

Запропоновані засади та кроки підсумовано в таблиці 3.

Таблиця 3.

Сутність авторського підходу до опрацювання проблем тестування СПЗ мінливих ВС

Засади підходу	Кроки реалізації засад
Організація конвеєра (pipeline) безперервного автоматизованого тестування СПЗ ВС у TDD-стилі [3] як підґрунтя для ефективного CI/CD конвеєра розгортання СПЗ в DevSecOps (B_1)	Побудова моделі властивостей (віртуальної) лінійки СПЗ мінливих ВС шляхом виокремлення в CI/CD конвеєрі конструювання СПЗ обов'язкових фрагментів, спільних для всіх СПЗ, і, відповідно, опціональних, властивих лише окремим СПЗ (S_1)
Формалізація обох конвеєрів у парадигмі лінійки програмних продуктів (Software Product line) згідно з ISO/IEC 26554 (для цільової Лінійки тестів мінливих СПЗ) і ДСТУ ISO/IEC 26550 (як (віртуальної) лінійки самих СПЗ) (B_2)	Побудова на підставі отриманої моделі властивостей технологічної моделі процесу автоматизованого тестування СПЗ як процесу розгортання й використання Лінійки їх тестів , а саме вкладених моделей конвеєрів тестування домену й застосунків та їх інфраструктури, згідно із засадами [13] і B_1, B_2 (S_2)
Декларативне моделювання процесів тестування домену та застосунків у парадигмі Pipeline as a Code [14] як часткових конвеєрів, вкладених до формованої Лінійки тестів, а інфраструктури тестування СПЗ – у	Налаштування традиційних технік тестування коду на рівнях від модульного до системного для СПЗ ВС та їх інтеграція з техніками поглибленого статичного аналізу [6] і фазингу [10,11] у модельованих вкладених кон-

Засади підходу	Кроки реалізації засад
парадигмі Infrastructure as a Code [15] (B_3)	веєрах тестування домену й застосунків (S_3)
Декларативне моделювання процесів розгортання та адміністрування інфраструктури Лінійки тестів у руслі підходів GitOps [16] і QAOps [5, 6] для адекватного розподілу відповідальностей і взаємної довіри різних команд в Embedded DevSecOps (B_4)	Розроблення методів ситуативного вибору оптимального інструмента, переважно з відкритим кодом, для основних видів статичного й динамічного тестування в модельованих конвеєрах рівня домену й застосунків (S_4) згідно з [17]
Застосування інструментальних засобів з відкритим кодом для декларативного конфігураційного керування автоматизованими конвеєрами Лінійки тестів та їх інфраструктурою (B_5)	Формування повної й ненадлишкової системи метрик якості запропонованої Лінійки тестів та розроблення процедур виявлення її неприйнятного зниження на їх підставі (S_5)
Забезпечення автоматизованого виявлення, у формованих конвеєрах Лінійки тестів, порушень відповідності тестованого СПЗ релевантним міжнародним і галузевим стандартам, насамперед щодо кібер- та інформаційної безпеки (B_6)	Аналіз ефективності інструментів з відкритим кодом для декларативного керування конвеєрами та інфраструктурою як кодом, де-факто стандартизованих підходами GitOps і QAOps, і визначення їх рамкового переліку для розгортання й використання запропонованої Лінійки тестів (S_6)
Безперервне відстеження якості запропонованої Лінійки тестів із вчасним виявленням її неприпустимого зниження та можливостей і способів удосконалення Лінійки (B_7)	Розроблення макетного зразка інструментального засобу вибору оптимального інструмента для певного виду тестування за допомогою методів кроку S_4, його інтеграція до процесів запропонованої Лінійки тестів разом із вибраними засобами з кроку S_6 та апробація отриманого рішення у вітчизняній організації-розробнику мінливих СПЗ ВС (S_7)

Реалізація й ситуативне уточнення кроків S_1.–S_7 є предметом подальших досліджень авторів.

Висновки

Запропоновано линійку тестів згідно зі стандартом ISO/IEC 26554 практиками QAOps для відокремленого тестування СПЗ ВС на рівнях від модульного до системного, де емуляцію замінено на використання тестових двійників (mocks, SiL) для модулів СПЗ і положеннями специфікації вимог до СПЗ.

Література

1. Beningo J. Embedded software design: A practical approach to architecture, processes, and coding techniques. Apress, 2022.
2. Lacamera D. Embedded systems architecture: Design and write software for embedded devices to build safe and connected systems. 2nd ed. Packt Publishing, 2023.
3. Grenning J.W. Test-Driven Development for Embedded C. Pragmatic Bookshelf, 2011.
4. Association for Standartization of Automation and Measuring Systems (ASAM). Generic Simulator Interface. Specification. Part 1, 2024. – 396 p.
5. Clokie K. A practical guide to Testing in DevOps – Leanpub, 2017. – 128 p.
6. Hornbeek M., Wakeman D. Continuous Testing, Quality, Security, and Feedback: Essential strategies and secure practices for DevOps, DevSecOps, and SRE transformations – Packt Publishing, Limited, 2024. – 420 p/
7. Salo O., Abrahamsson P. Agile methods in European embedded software development organisations: A survey on the actual use and usefulness of Extreme Programming and Scrum. IET Software. 2008. Vol. 2. No 1. P. 58–64.
8. Böhm W. et al. (Eds.): Model-Based Engineering of Collaborative Embedded Systems. ISBN 978-3-030-62135-3. Springer, Jan. 2021 – 358 p.
9. Nicolescu G., Mosterman P.J. (eds.). Model-Based Design for Embedded Systems. CRC Press, 2009.
10. Yun J., Lee I., Xu M., Kim T. Fuzzing of embedded systems: A survey. ACM

Computing Surveys. 2022. Vol. 55. No 7. Article 132. P. 1–33.
11. Eisele M., Ebert D., Huth C., Zeller A. Fuzzing embedded systems using debug interfaces. Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA ’23). 2023. P. 1031–1042.
12. Feng B., Mera A., Lu L. P²IM: Scalable and hardware-independent firmware testing via automatic peripheral interface modeling. 29th USENIX Security Symposium (USENIX Security 20). 2020. P. 1237–1254.
13. Ierofeiev I., Sinitsyn I., Slabospitska O. Embedded Software Testing Issues and Addressing Them with Software Product Lines Paradigm. CEUR Workshop Proceedings. 2024. Vol. 4053. P. 13–23.
14. Soni M. Hands-on Pipeline as YAML with Jenkins: A Beginner's Guide to Implement CI/CD Pipelines for Mobile, Hybrid, and Web Applications Using Jenkins BPB Publications, 2021 – 320 p.
15. Wang R.Infrastructure as Code, Patterns and Practices. With examples in Python and Terraform – Manning, 2022. – 400 p.
16. Salecha R. Practical GitOps : Infrastructure Management Using Terraform, AWS, and GitHub Actions – Apress L. P., 1st ed., 2022 –270 p.
17. Єрофеєв Ю.В., Слабоспицька О.О. Embedded DevSecOps у хмарі: засоби та перспективи оптимізації. Proc. of the XVIII Int. scientific and practical conf. «Information technologies and automation–2025». 2025 – P. 837.

References

1. Beningo J. Embedded software design: A practical approach to architecture, processes, and coding techniques. Apress, 2022.
2. Lacamera D. Embedded systems architecture: Design and write software for embedded devices to build safe and connected systems. 2nd ed. Packt Publishing, 2023.
3. Grenning J.W. Test-Driven Development for Embedded C. Pragmatic Bookshelf, 2011.
4. Association for Standartization of Automation and Measuring Systems (ASAM). Generic Simulator Interface. Specification. Part 1, 2024. – 396 p.
5. Clokie K. A practical guide to Testing in DevOps – Leanpub, 2017. – 128 p.
6. Hornbeek M., Wakeman D. Continuous Testing, Quality, Security, and Feedback: Essential strategies and secure practices for DevOps, DevSecOps, and SRE transformations – Packt Publishing, Limited, 2024. – 420 p/
7. Salo O., Abrahamsson P. Agile methods in European embedded software development organisations: A survey on the actual use and usefulness of Extreme Programming and Scrum. IET Software. 2008. Vol. 2. No 1. P. 58–64.
8. Böhm W. et al. (Eds.): Model-Based Engineering of Collaborative Embedded Systems. ISBN 978-3-030-62135-3. Springer, Jan. 2021 – 358 p.
9. Nicolescu G., Mosterman P.J. (eds.). Model-Based Design for Embedded Systems. CRC Press, 2009.
10. Yun J., Lee I., Xu M., Kim T. Fuzzing of embedded systems: A survey. ACM Computing Surveys. 2022. Vol. 55. No 7. Article 132. P. 1–33.
11. Eisele M., Ebert D., Huth C., Zeller A. Fuzzing embedded systems using debug interfaces. Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA ’23). 2023. P. 1031–1042.
12. Feng B., Mera A., Lu L. P²IM: Scalable and hardware-independent firmware testing via automatic peripheral interface modeling. 29th USENIX Security Symposium (USENIX Security 20). 2020. P. 1237–1254.
13. Ierofeiev I., Sinitsyn I., Slabospitska O. Embedded Software Testing Issues and Addressing Them with Software Product Lines Paradigm. CEUR Workshop Proceedings. 2024. Vol. 4053. P. 13–23.
14. Soni M. Hands-on Pipeline as YAML with Jenkins: A Beginner's Guide to Implement CI/CD Pipelines for Mobile, Hybrid, and Web Applications Using Jenkins BPB Publications, 2021 – 320 p.
15. Wang R.Infrastructure as Code, Patterns and Practices. With examples in Python and Terraform – Manning, 2022. – 400 p.
16. Salecha R. Practical GitOps : Infrastructure Management Using Terraform, AWS, and GitHub Actions – Apress L. P., 1st ed., 2022 –270 p.
17. Ierofeiev I., Slabospitska O. Cloude Embedded DevSecOps: Optimization Tools and Perspectives. Proc. of the XVIII Int. scientific and practical conf. «Information technologies and automation–2025». 2025. – P. 837.

Дата першого надходження до видання: 10.03.2026
Внутрішня рецензія отримана: 14.03.2026
Зовнішня рецензія отримана: 15.04.2026
Дата прийняття статті до друку: 19.03.2026
Дата публікації: 16.04.2026

Про авторів:

[1]*Єрофеєв Юрій Володимирович,*
аспірант ІПС НАНУ
Yerofieiev Yuriy,
Post-graduate student
http://orcid.org/0009-0006-8985-2729.

[2]*Сініцин Ігор Петрович,*
доктор технічних наук,
директор.
Sinitsyn Igor,
Ph.D (doctor, technical sciences),
director
http://orcid.org/0000-0002-4120-0784.

Місце робот авторів:

[1]Інститут програмних систем
НАН України
Institute of Software Systems
National Academy of Sciences of Ukraine
тел. +38-044-522-62-42
E-mail: www.iss.nas.gov.ua

УДК 004.056.5:004.7 https://doi.org/10.15407/pp2026.01.066

Ю.В. Костюк, П.М. Складанний, Д.Д. Гнатченко

РИЗИК-АДАПТИВНА АВТОРИЗАЦІЯ В ZERO TRUST ІЗ ДИНАМІЧНОЮ ДОВІРОЮ ТА ТОКЕНАМИ

У статті розв'язується задача ризик-адаптивної авторизації в архітектурі Zero Trust із використанням механізму динамічної оцінки довіри та адаптивного керування токенами доступу. Актуальність дослідження зумовлена зростанням кількості атак, пов'язаних із компрометацією облікових даних, перехопленням сесій та зловживанням привілеями в корпоративних інформаційних системах. Запропонований підхід базується на безперервному аналізі поведінкових характеристик користувача, параметрів пристрою, мережевого контексту та критичності ресурсу, що дозволяє формувати інтегральний показник ризику в реальному масштабі часу. На відміну від традиційних моделей із фіксованим часом життя токенів і статичними політиками доступу, розроблена модель передбачає динамічну зміну рівня довіри протягом усієї сесії взаємодії. Інтегральний показник ризику визначається як функція ймовірності реалізації загрози та потенційного впливу на актив, що забезпечує адаптивне коригування параметрів авторизації: обсягу привілеїв, часу дії токена, необхідності повторної автентифікації або примусової ротації криптографічних ключів. Реалізовано механізм скорочення або продовження життєвого циклу токена залежно від змін безпекового контексту, що мінімізує вікно можливого використання скомпрометованих облікових даних. Особливу увагу приділено забезпеченню балансу між рівнем безпеки, продуктивністю та зручністю користування. Запропонований підхід підвищує стійкість до атак типу session hijacking, credential stuffing та insider misuse і може бути інтегрований у сучасні системи управління доступом без значного збільшення обчислювальних витрат. Результати моделювання підтверджують ефективність застосування динамічної довіри як ключового елементу ризик-адаптивної авторизації.

Ключові слова: ризик-адаптивна авторизація, Zero Trust (нульова довіра), керування доступом, оцінювання довіри, час життя токена, безпека ідентичностей, керування доступом на основі політик

Y. Kostiuk, P. Skladannyi, D. Hnatchenko

RISK-ADAPTIVE AUTHORIZATION IN ZERO TRUST WITH DYNAMIC TRUST AND TOKENS

The article addresses the problem of risk-adaptive authorization in a Zero Trust architecture using a mechanism of dynamic trust assessment and adaptive access token management. The relevance of the study is driven by the increasing number of attacks related to credential compromise, session hijacking, and privilege misuse in corporate information systems. The proposed approach is based on continuous analysis of user behavioral characteristics, device parameters, network context, and resource criticality, enabling the formation of an integral risk indicator in real time. Unlike traditional models with fixed token lifetimes and static access policies, the developed model provides dynamic adjustment of the trust level throughout the entire interaction session. The integral risk indicator is defined as a function of threat likelihood and potential impact on the asset, ensuring adaptive adjustment of authorization parameters, including privilege scope, token lifetime, requirement for re-authentication, or enforced cryptographic key rotation. A mechanism for shortening or extending the token lifecycle depending on changes in the security context is implemented, thereby minimizing the window of opportunity for exploiting compromised credentials. Special attention is paid to maintaining a balance between security level, system performance, and usability. The proposed approach increases resilience against session hijacking, credential stuffing, and insider misuse attacks and can be integrated into modern access management systems without significant growth in computational overhead. Modeling results confirm the effectiveness of dynamic trust as a key element of risk-adaptive authorization.

Keywords: risk-adaptive authorization, Zero Trust, access management, trust evaluation, token lifetime, identity security, policy-based access control.

Вступ

Цифрова трансформація корпоративних інформаційних систем супроводжується зростанням ролі механізмів керування доступом, у межах яких авторизація виступає ключовим елементом забезпечення інформаційної безпеки [1]. Сучасні атаки дедалі частіше спрямовані не на подолання криптографічних механізмів, а на

компрометацію облікових записів, викрадення токенів доступу та зловживання привілеями, що призводить до тривалого несанкціонованого перебування зловмисника в системі [4, 8, 10]. За таких умов традиційні підходи до авторизації, засновані на статичних політиках і фіксованому часі життя сесій, виявляються недостатньо ефективними, оскільки не враховують динаміку ризиків і контекст виконання операцій.

Актуальність цієї проблеми посилюється впровадженням розподілених, хмарних та гібридних архітектур, у яких користувачі, сервіси та пристрої здійснюють доступ до ресурсів поза межами традиційного периметра безпеки [6-7]. У таких середовищах концепція Zero Trust розглядає кожен запит доступу як потенційно небезпечний і вимагає безперервної перевірки не лише ідентичності, а й рівня довіри, поведінки та контексту доступу [2, 12]. Проте на практиці реалізація Zero Trust часто обмежується посиленою автентифікацією, тоді як авторизація та керування токенами доступу залишаються статичними й слабо адаптованими до поточного стану безпеки.

У науковому та прикладному вимірах залишається відкритою задача побудови формалізованих моделей авторизації, здатних поєднувати оцінювання довіри, ризику та керування сесіями в єдиному контурі ухвалення рішень [1, 15]. Особливої уваги потребує проблема зменшення так званого «вікна атаки» — проміжку часу, протягом якого скомпрометований токен або сесія можуть бути використані зловмисником без виявлення та блокування [8-9, 17]. Розв'язання цієї задачі має безпосередній зв'язок із практичними завданнями підвищення стійкості систем керування доступом, мінімізації наслідків інцидентів безпеки та забезпечення безперервності бізнес-процесів.

У цьому контексті доцільним є розроблення ризик-адаптивних підходів до авторизації, які враховують динамічну зміну рівня довіри до користувача або сервісу, контекстні та поведінкові фактори, а також дозволяють керувати часом життя токенів доступу залежно від поточного ризику [5, 12-13, 15]. Такий підхід забезпечує зв'язок між фундаментальними науковими дослідженнями у сфері моделей доступу та практичними задачами впровадження Zero Trust-архітектур у корпоративних інформаційно-комунікаційних системах.

У роботі зроблено такі внески, що визначають її наукову новизну та практичну цінність. По-перше, формалізовано модель ризик-адаптивної авторизації в архітектурі Zero Trust як динамічного контуру ухвалення рішень [12-13], у якому враховуються суб'єкти доступу, активи, привілеї та контекст безпеки, а ключові параметри довіри й ризику оновлюються під час активної сесії. По-друге, запропоновано механізм $TL - TTL$ [15, 18], який пов'язує динамічну оцінку довіри та операційного ризику з адаптивним керуванням часом життя токенів доступу, забезпечуючи подієву реакцію на ризикові ситуації (поведінкові аномалії, зміни контексту, порушення цілісності пристрою) та скорочення потенційного «вікна атаки» у разі компрометації сесії. По-третє, введено систему метрик для кількісного оцінювання ефективності запропонованого підходу [8, 16], зокрема, метрику вікна атаки W, показники частоти додаткових перевірок доступу S та затримки ухвалення рішень L на рівнях PDP/PEP. По-четверте, виконано сценарне порівняльне оцінювання запропонованого методу з базовими моделями авторизації зі статичним та чутливісно-орієнтованим часом життя токенів [10, 12], що дозволило обґрунтувати переваги ризик-адаптивного керування сесіями в різних умовах доступу. По-п'яте, визначено умови застосування підходу та його обмеження [2, 7], пов'язані з якістю сигналів ризикових подій, необхідністю калібрування порогів і вагових коефіцієнтів, а також компромісом між посиленням безпеки та зручністю користувача в корпоративних середовищах.

Незважаючи на активний розвиток технологій керування ідентичностями та доступом, проблема ефективної авторизації в корпоративних інформаційних системах залишається частково невирішеною [1, 5]. Більшість наявних рішень зосереджені на посиленні автентифікації користувачів або розширенні політик доступу, тоді як процес авторизації та керування життєвим циклом токенів доступу часто реалізується за ста-

тичними правилами [10, 12-13]. У результаті системи виявляються недостатньо чутливими до змін контексту, поведінкових аномалій і поточного рівня ризику, що створює умови для тривалого зловживання скомпрометованими сесіями.

Аналіз наукових і практичних підходів показує, що в межах концепції Zero Trust відсутнє єдине формалізоване рішення, яке б інтегрувало оцінювання рівня довіри, ризику операції та керування параметрами сесії в узгоджений механізм ухвалення рішень [2, 10, 19]. Зокрема, залишаються невирішеними такі питання: яким чином кількісно враховувати динамічну зміну довіри до користувача або сервісу під час активної сесії [15, 18]; як поєднати цю оцінку з ризиками конкретної операції та чутливістю активу [4, 13]; як на основі отриманих показників адаптивно змінювати параметри авторизації без надмірного впливу на зручність користувачів і продуктивність системи.

Окремою проблемою є керування часом життя токенів доступу, який у більшості реалізацій визначається наперед і не змінюється у відповідь на ризикові події [10]. Такий підхід не дозволяє оперативно скорочувати «вікно атаки» у разі компрометації облікових даних, викрадення токенів або виявлення аномальної поведінки. Водночас відсутність формалізованих правил адаптивного керування токенами ускладнює оцінювання ефективності запропонованих рішень і їх порівняння з базовими моделями.

У зв'язку з цим постає науково-практичне завдання розроблення ризик-адаптивної моделі авторизації, яка забезпечує узгоджене використання динамічної оцінки довіри, контекстних і поведінкових ризиків та механізмів керування токенами доступу [2, 15, 18]. Така модель має бути формалізованою, придатною для реалізації в системах IAM/PAM та дозволяти кількісно оцінювати її вплив на зменшення потенційного вікна атаки й експлуатаційні характеристики авторизації в умовах Zero Trust.

Модель загроз охоплює атаки, пов'язані з компрометацією токенів доступу та зловживанням сесіями (token theft, session hijacking, повторне використання токена після первинної автентифікації), а також підвищення ризику через контекстні та поведінкові аномалії (нестандартна геолокація, зміна пристрою/мережі, нетипова частота запитів). Припускається наявність джерел телеметрії UEBA/SIEM/IdP/EDR/NAC і можливість анулювання/обмеження сесії протягом інтервалу Δt [8, 16, 20-21]. Поза межами розгляду залишаються сценарії повної компрометації IdP/PDP або відсутності достовірної телеметрії (наприклад, повний контроль зловмисника над endpoint без детектування).

Аналіз існуючих досліджень

У сучасних наукових дослідженнях значна увага приділяється розвитку систем керування ідентичностями та доступом (IAM) і впровадженню архітектури Zero Trust у корпоративних інформаційних системах. У роботах J. Glöckler, J. Sedlmeir, M. Frank та G. Fridgen узагальнено вимоги підприємств до IAM, зокрема, щодо керування життєвим циклом ідентичностей, узгодженості атрибутів, аудиту доступу та зменшення ризиків зловживання привілеями [1]. Автори переконливо показують, що сучасні корпоративні системи потребують більш гнучких і контекстно-орієнтованих механізмів доступу. Водночас запропонований у роботі аналіз має переважно концептуальний характер і не пропонує формалізованого механізму авторизації, який би безпосередньо пов'язував динамічні показники довіри та ризику з параметрами активної сесії й часом життя токенів доступу. У виправленнях і доповненнях до цього огляду [3] уточнюється методологічна коректність систематизації вимог, однак прикладна проблема адаптивного керування авторизаційними сесіями залишається поза межами розгляду.

Дослідження A. Aljohani зосереджене на практичних аспектах реалізації Zero Trust у сучасних корпоративних мережах, де кожен запит доступу розглядається як потенційно небезпечний [2]. Робота демонструє ефективність безперервної перевірки ідентичності та контексту, однак авторизація в більшості сценаріїв реалізується як статичне або напівстатичне рі-

шення на рівні політик доступу. Питання динамічного коригування параметрів сесії після надання доступу, зокрема, часу життя токенів у відповідь на зміну ризику, не отримує достатнього розвитку.

Окрему групу становлять роботи, присвячені формальним моделям доступу. Так M. U. Aftab та співавтори пропонують динамічну RBAC-модель із permission-based separation of duty, спрямовану на зменшення ризиків зловживання повноваженнями [4]. Запропонована модель суттєво підвищує контроль над призначенням ролей і дозволів, однак вона не враховує сценаріїв компрометації вже виданих токенів або активних сесій, що є типовими для сучасних атак на ідентичності.

У контексті хмарних і розподілених середовищ V. Yadav, M. K. Soni та A. Pratap розглядають захищене IAM у хмарних обчисленнях на основі Zero Trust [5]. Автори підкреслюють важливість урахування контексту доступу та чутливості ресурсів, проте запропоновані підходи зосереджені переважно на етапі ухвалення первинного рішення про доступ. Керування життєвим циклом токенів доступу впродовж активної сесії залишається статичним або прив'язаним до наперед визначених профілів ресурсів, що обмежує здатність системи оперативно реагувати на компрометацію токенів чи поведінкові аномалії.

Аналогічні обмеження простежуються у роботі H. Sivaraman, присвяченій Zero Trust IAM у multi-cloud середовищах [6]. Автор детально аналізує проблеми уніфікації політик ідентичностей і доступу між різними хмарними платформами та наголошує на необхідності безперервної оцінки доступу. Водночас у дослідженні не запропоновано формалізованого алгоритму, який би пов'язував результати такої оцінки з адаптивним керуванням параметрами авторизаційних сесій і токенів.

У роботі S. Ahmadi розглянуто застосування Zero Trust у хмарних мережах, а також визначено ключові виклики, пов'язані з масштабованістю, якістю телеметрії та балансом між безпекою і зручністю користувачів [7]. Автор зазначає, що надмірно жорсткі політики можуть негативно впливати на безперервність бізнес-процесів, однак у дослідженні не запропоновано кількісних моделей, які дозволяли б оптимізувати цей компроміс шляхом адаптивного керування часом життя сесій залежно від поточного ризику.

Важливим доповненням до проблематики Zero Trust є дослідження J. Lee та співавторів, у якому запропоновано метод ранжування аномальних активностей у корпоративних мережах [8]. Робота демонструє, як телеметричні дані можуть бути перетворені на числові оцінки аномальності та ризику. Проте результати такого ранжування розглядаються переважно як інструмент для SOC або моніторингу, а не як керуючий сигнал для механізмів авторизації та керування сесіями.

Проведений аналіз наукових публікацій показує, що, незважаючи на активний розвиток концепції Zero Trust і систем IAM, у більшості досліджень авторизація розглядається або як статичне рішення на момент запиту доступу, або як похідна від ролей, атрибутів чи чутливості активів. Питання інтеграції динамічної оцінки довіри, контекстних і поведінкових ризиків та керування параметрами активної сесії в єдиний формалізований контур ухвалення рішень залишається недостатньо вирішеним.

Зокрема, у наявних підходах відсутній узгоджений механізм адаптивного керування часом життя токенів доступу у відповідь на ризикові події, що безпосередньо пов'язано з проблемою мінімізації потенційного «вікна атаки» у разі компрометації сесії. Також бракує кількісних моделей і метрик, які дозволяли б порівнювати ефективність різних підходів до авторизації з точки зору балансу між рівнем безпеки, затримкою ухвалення рішень і зручністю користувачів. У сукупності ці невирішені питання зумовлюють необхідність розроблення ризик-адаптивної моделі авторизації в архітектурі Zero Trust, яка поєднує динамічну довіру, оцінювання ризику та адаптивне керування токенами доступу в одному формалізованому механізмі.

Метою статті є розробка формалізованої ризик-адаптивної моделі авторизації в архітектурі Zero Trust, що поєднує динамічну оцінку рівня довіри до користувачів і сервісів, контекстні та поведінкові фактори

ризику, а також адаптивне керування параметрами сесій і токенів доступу. Запропонований підхід спрямований на зменшення потенційного вікна атак у разі компрометації ідентичностей або токенів, підвищення точності рішень про доступ та забезпечення балансу між рівнем безпеки, продуктивністю системи й зручністю використання в корпоративних інформаційних системах.

Для формалізації проблеми ефективності авторизації доцільно ввести кількісну метрику, що безпосередньо характеризує наслідки компрометації сесій і токенів доступу [10, 12]. Нехай t_c – момент компрометації токена доступу або облікових даних, t_r – момент його відкликання, анулювання або завершення дії. Тоді потенційне вікно атаки визначається як: $W = t_r - t_c$ і характеризує проміжок часу, протягом якого зловмисник може несанкціоновано використовувати скомпрометований доступ [13, 19]. У традиційних моделях авторизації зі статичним часом життя токенів величина W практично еквівалентна заданому значенню TTL, оскільки параметри сесії не змінюються у відповідь на ризикові події [5, 10]. Натомість у ризик-адаптивному підході $TL - TTL$ величина W зменшується за рахунок подієвого коригування часу життя токена, коли виявлення аномальної поведінки, змін контексту або інших ризикових подій призводить до дострокового обмеження або відкликання сесії [12, 18-19]. Таким чином, задача авторизації в умовах Zero Trust формалізується як задача мінімізації потенційного вікна атаки W за умови збереження прийнятних експлуатаційних характеристик системи доступу.

Виклад основного матеріалу дослідження

Методологія дослідження ґрунтується на поєднанні формалізованих моделей керування доступом, ризик-орієнтованих підходів до авторизації та принципів архітектури Zero Trust [2, 10, 12-13, 19]. У межах роботи використано системний підхід до аналізу процесів авторизації в корпоративних інформаційних системах, що дозволяє розглядати керування доступом як динамічний контур ухвалення рішень, залежний від рівня довіри, контексту та поведінки суб'єктів доступу.

Основу дослідження становить формалізація ключових елементів системи Access Management, зокрема множин користувачів, активів і привілеїв, а також відношень доступу між ними. Авторизація розглядається як функція, що відображає атрибути суб'єкта, характеристики ресурсу та поточний контекст безпеки у рішення щодо дозволу, обмеження або заборони доступу. Такий підхід забезпечує можливість інтеграції рольових, атрибутивних і політик-орієнтованих моделей у єдину схему ухвалення рішень.

Для врахування динаміки ризиків у процесі доступу застосовано методи оцінювання довіри та ризику операцій [13, 19]. Рівень довіри до користувача або сервісу визначається на основі сукупності ідентифікаційних, контекстних, пристроєвих і поведінкових факторів та розглядається як змінна величина, що оновлюється впродовж активної сесії [15, 18]. Оцінка ризику операції формується з урахуванням чутливості активів, умов доступу та поточного стану безпеки, що дозволяє адаптувати рішення авторизації до реальних загроз.

Ключовим елементом методології є ризик-адаптивне керування токенами доступу, яке реалізується шляхом динамічного визначення часу їхнього життя залежно від рівня довіри, контексту доступу та критичності ресурсів [5-6, 12]. У дослідженні використано подієвий підхід, за якого виявлення ризикових подій (аномальної поведінки, змін контексту або порушень політик) призводить до коригування параметрів сесії та, за необхідності, ініціювання додаткових перевірок доступу. Це дозволяє мінімізувати потенційне вікно атак без надмірного зниження зручності використання системи.

Джерелом формування ризикових подій у запропонованій методології виступають різнорідні компоненти корпоративної інфраструктури безпеки, зокрема, системи аналізу поведінки користувачів і сутностей (UEBA), системи управління подіями та інцидентами безпеки (SIEM), сервіси керування ідентичностями та доступом (IdP/IAM), засоби захисту кінцевих точок

(EDR), а також механізми мережевого контролю доступу (NAC) [8, 13, 20-21]. Телеметричні дані, що надходять із цих джерел, агрегуються та нормалізуються з метою виявлення подій, які можуть свідчити про зростання ризику доступу, такі як аномальна поведінка, порушення цілісності пристрою, нетипові зміни контексту або спроби зловживання токенами доступу.

Для кожної зафіксованої події формується оцінка серйозності $Sev(RiskEvent)$, яка може визначатися на основі правил (rule-based підхід) або за допомогою моделей машинного навчання у вигляді нормалізованого ризикового скору [8, 16]. Оновлення рівня довіри TL здійснюється динамічно — безпосередньо під час обробки запиту доступу (per-request), періодично з фіксованим інтервалом Δt або асинхронно у відповідь на надходження ризикової події (on-event), залежно від вимог до чутливості та продуктивності системи [12-13, 15]. Узагальнено методологічний конвеєр ризик-адаптивної авторизації можна подати у вигляді послідовності: Telemetry → RiskEvent scoring → TL update → RiskOp evaluation → Decision & TTL update → PEP enforcement, що забезпечує замкнений контур ухвалення рішень, у якому сигнали безпеки безпосередньо впливають на параметри авторизації та керування сесіями в умовах Zero Trust.

Схема на рис. 1 ілюструє архітектуру ризик-адаптивної авторизації в парадигмі Zero Trust, у якій події безпеки з джерел телеметрії (UEBA, SIEM, IdP/IAM, EDR, NAC) агрегуються у формалізований об'єкт RiskEvent та оцінюються за рівнем серйозності. На рівні PDP здійснюється обчислення та оновлення показників динамічної довіри і операційного ризику, на основі яких формується рішення доступу та адаптивно коригується час життя токена (TTL). Рівень PEP забезпечує примусове виконання ухваленого рішення (permit, step-up, deny), а також керування токенами і сесіями (introspection, denylist, revoke, rotation) з одночасним формуванням аудиторських подій. Згенерований аудит повертається до SIEM, утворюючи замкнений контур зворотного зв'язку, що забезпечує безперервну переоцінку довіри та мінімізацію вікна компрометації сесії. Запропонована архітектура забезпечує перехід від статичної моделі авторизації до безперервного контекстно-орієнтованого контролю доступу, у якому параметри безпеки динамічно узгоджуються з поточним рівнем ризику. Такий підхід підвищує стійкість системи до компрометації облікових даних і перехоплення сесій без істотного зростання обчислювального навантаження та збереження прийнятного рівня зручності користування.

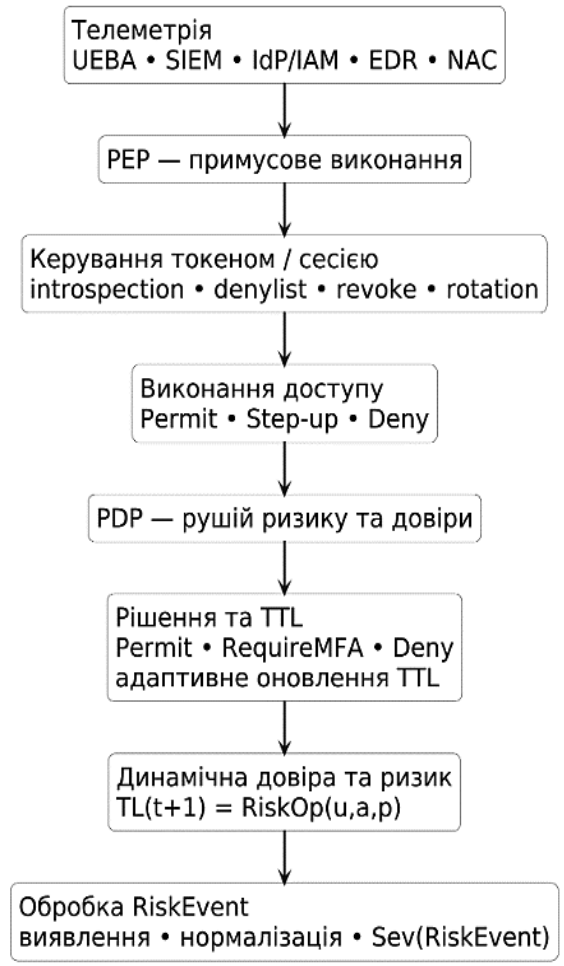

Рис. 1. Архітектура ризик-адаптивної авторизації в Zero Trust (PDP/PEP)

На рис. 2 подано граф подій та керуючих впливів у контурі PDP/PEP архітектури Zero Trust. Після входу користувача та видачі токена зміна контексту доступу (локація, пристрій) ініціює формування $RiskEvent$, для якого обчислюється рівень серйозності $Sev(RiskEvent)$. На основі цієї оцінки PDP ухвалює рішення доступу (*Permit / RequireMFA / Deny*), яке реалізується рівнем PEP через примусове вико-

нання, step-up аутентифікацію або відкликання/ротацію токена. Усі дії журналюються та передаються до SIEM, забезпечуючи кореляцію й збагачення подій, що підкреслює подієву, керовану та пояснювану природу ризик-адаптивної авторизації.

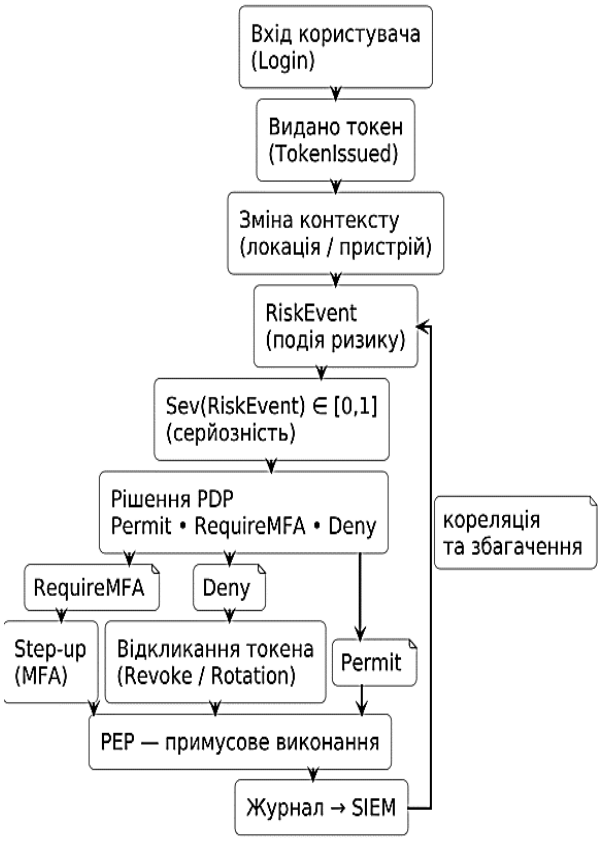

Рис. 2. Граф подій та керуючих впливів RiskEvent у контурі PDP/PEP (Zero Trust)

Оцінювання ефективності запропонованого підходу здійснюється за сценарним принципом із використанням порівняльного аналізу [10, 12-13, 19]. У межах дослідження розглядаються типові сценарії доступу, що включають нормальну роботу, контекстні та поведінкові аномалії, а також компрометацію токенів. Запропонований метод порівнюється з базовими моделями авторизації за показниками тривалості потенційного вікна атаки, частоти додаткових перевірок доступу та експлуатаційних характеристик ухвалення рішень. Такий підхід забезпечує об'єктивну оцінку переваг ризик-адаптивної авторизації в умовах Zero Trust.

Запропонований механізм може бути реалізований у токеноорієнтованих протоколах (OAuth 2.0 / OpenID Connect) через адаптивне керування TTL для access token та політику refresh token з подієвим скороченням строку дії [5-6, 16]. На практиці PDP виконує TL/RiskOp-оцінювання, а PEP застосовує рішення (Permit/RequireMFA/Deny) і параметри токена; ризикові події надходять через SIEM/UEBA або брокер подій, а анулювання може виконуватися через introspection/denylist/rotation залежно від обраної архітектури.

Формальна модель авторизації та довіри. Формалізація процесу авторизації в умовах Zero Trust передбачає опис суб'єктів доступу, ресурсів, привілеїв і контексту безпеки у вигляді взаємопов'язаних множин і функцій, що дозволяє кількісно враховувати довіру та ризик під час ухвалення рішень [13, 19]. На відміну від статичних моделей контролю доступу, запропонований підхід орієнтований на динамічне оновлення параметрів авторизації залежно від поведінки користувача та змін середовища.

Нехай множина користувачів і сервісних суб'єктів доступу корпоративної системи визначається як:

$$U = \{u_1, u_2, \dots, u_m\},$$

де u_j – окремий користувач або сервіс, що ініціює запит на доступ. Кожному суб'єкту відповідає набір атрибутів:

$$Attr(u_j) = \{Role_j, Dept_j, Clearance_j, Device_j, Location_j\},$$

які описують його функціональну роль, рівень повноважень, характеристики пристрою та контекст доступу [1, 13]. У рамках Zero Trust ці атрибути розглядаються як змінні величини, що можуть оновлюватися під час активної сесії.

Множина активів інформаційної системи задається як:

$$A = \{a_1, a_2, \dots, a_n\},$$

де a_i – окремий інформаційний ресурс, сервіс або об'єкт обробки даних [1]. Для кожного активу визначається вектор безпекових характеристик:

$$Attr(a_i) = \{Conf_i, Int_i, Avail_i, Sens_i, Owner_i\},$$

де $Conf_i, Int_i, Avail_i$ відображають вимоги до конфіденційності, цілісності та доступ-

ності, а $Sens_i$ характеризує чутливість активу. На основі цих параметрів формується оцінка ризику активу:

$$Risk(a_i) = f(Conf_i, Int_i, Avail_i, Threats_i, Vuln_i),$$

яка використовується для визначення суворості політик доступу та параметрів авторизації.

Множина дозволених операцій або привілеїв визначається як:

$$P = \{p_1, p_2, \dots, p_k\},$$

де p_k відповідає окремій дії над активом, наприклад, читанню, зміні або адмініструванню. Відношення доступу між користувачами, активами та привілеями описується множиною:

$$Access \subseteq U \times A \times P,$$

причому трійка $(u_j, a_i, p_k) \in Access$ означає, що суб'єкту u_j дозволено виконувати операцію p_k над активом a_i.

Для реалізації політик RBAC, ABAC і PBAC вводяться допоміжні відношення призначення ролей і привілеїв:

$$UA \subseteq U \times Role,\ PA \subseteq Role \times P,$$

У загальному випадку рішення про доступ формується як результат функції авторизації:

$$Decision = PDR(U, A, P, Policies, Context)$$

де $Policies$ – множина формалізованих правил доступу, $Context$ – сукупність параметрів середовища виконання запиту.

Ключовим елементом запропонованої моделі є динамічна оцінка рівня довіри до суб'єкта доступу [13, 19]. Рівень довіри $TL(u_j)$ визначається як зважена агрегація нормалізованих показників ідентичності, стану пристрою, контексту та поведінкових характеристик:

$$TL(u_j) = \sum_{l=1}^{L} w_l \cdot z_l(u_j),\ z_l \in [0,1],$$
$$\sum w_l = 1,$$

де $z_l(u_j)$ – окремі складові оцінки довіри (якість автентифікації, безпечність пристрою, стабільність поведінки), w_l – їхні вагові коефіцієнти. На відміну від статичних моделей, значення TL постійно оновлюється під час сесії відповідно до зафіксованих подій безпеки:

$$TL_{t+1} = clip(0, 1, TL_t - \Delta T(RiskEvent_t)),$$

де $RiskEvent_t$ відображає наявність аномалій або підозрілих дій у момент часу t. Оператор $clip\ (0, 1, \cdot)$ обмежує значення рівня довіри в діапазоні [0,1] і запобігає виходу TL за межі допустимих значень..

Оцінка ризику конкретної операції доступу визначається функцією

$$RiskOp(u_j, a_i, p_k) = \alpha \cdot Risk(a_i) + \beta \cdot \left(1 - TL(u_j)\right) + \gamma \cdot RiskContext,$$

де $RiskContext$ характеризує поточний контекст виконання операції (геолокація, час, тип мережі), а коефіцієнти α, β, γ задають внесок кожного компонента. Така формалізація дозволяє поєднати статичні характеристики активу з динамічною довірою до користувача.

Запропонована модель відрізняється від традиційних підходів тим, що рішення про доступ ухвалюється не лише на основі ролей або атрибутів, а з урахуванням поточного ризику операції та рівня довіри [12-13, 19]. Це створює основу для реалізації ризик-адаптивної авторизації, у межах якої подальше керування токенами доступу та параметрами сесії може бути безпосередньо пов'язане зі значеннями TL і $RiskOp$. Таким чином, формальна модель авторизації та довіри забезпечує математичне підґрунтя для побудови адаптивних Zero Trust-рішень, орієнтованих на мінімізацію вікна атак і підвищення стійкості систем керування доступом.

Запропонований ризик-адаптивний метод керування токенами доступу ($TL - TTL$). Запропонований метод керування токенами доступу ґрунтується на ідеї динамічного коригування параметрів сесії відповідно до поточного рівня довіри та ризику виконуваних операцій [6, 10, 19]. На відміну від традиційних підходів, у яких час життя токена визначається статично або залежить лише від чутливості ресурсу, у межах даного дослідження токен розглядається як адаптивний елемент авторизаційного контуру Zero Trust.

Нехай TTL – час життя токена доступу, що визначає максимальну тривалість дії авторизаційного рішення [12-13]. Запропоновано визначати початкове значення TTL як функцію динамічного рівня довіри до суб'єкта доступу, контекстних ризиків і чутливості активу:

$$TTL = f(TL, RiskContext, Sensitivity),$$

Для практичної реалізації ця залежність може бути задана у вигляді лінійної моделі з обмеженнями:

$$TTL = clip(TTL_{min}, TTL_{max},\ k_1 \cdot TL - k_2 \cdot RiskContext - k_3 \cdot Sensitivity),$$

де TTL_{min} та TTL_{max} – мінімальне і максимальне допустимі значення часу життя токена, $TL \in [0,1]$ – поточний рівень довіри, $RiskContext \in [0,1]$ – оцінка ризику контексту доступу, $Sensitivity \in [0,1]$ – чутливість активу, k_1, k_2, k_3 – коефіцієнти впливу відповідних факторів.

Особливістю запропонованого підходу є підтримка подієвого коригування часу життя токена впродовж активної сесії. У разі фіксації ризикових подій, таких як поведінкові аномалії, зміна геолокації або порушення політик безпеки, здійснюється штрафне зменшення часу життя токена [12-13]:

$$TTL_{t+1} = TTL_t - \Delta t - Penalty(RiskEvent_t),$$

де Δt – час, що минув з моменту попереднього оновлення, $Penalty(RiskEvent_t)$ – штрафна функція, пропорційна серйозності зафіксованої події:

$$Penalty(RiskEvent) = \lambda \cdot Sev(RiskEvent),\ Sev \in [0,1],$$

Отож, навіть за наявності чинного дозволу на доступ токен може бути достроково обмежений або відкликаний у разі зростання ризику, що істотно зменшує потенційне вікно атаки при компрометації сесії.

Запропонований метод інтегрується у стандартну архітектуру PDP/PEP і дозволяє реалізувати ризик-адаптивне керування сесіями без зміни базових механізмів автентифікації. Це забезпечує його сумісність із сучасними системами IAM/PAM та практичну придатність для впровадження в корпоративних інформаційних системах.

Модель ухвалення рішень авторизації. Для формалізації процесу авторизації з урахуванням ризик-адаптивного керування токенами доступу розглянемо модель ухвалення рішень, реалізовану на рівні Policy Decision Point (PDP) з подальшим застосуванням результатів на рівні Policy Enforcement Point (PEP).

Нехай запит доступу описується кортежем [13, 19]:

$$q = \langle u, a, p, ctx, t \rangle,$$

де u – суб'єкт доступу, a – актив, p – операція, ctx – поточний контекст, t – момент часу. На основі цього запиту PDP обчислює рівень довіри $TL(u)$та ризик операції $RiskOp(u, a, p)$.

Рішення авторизації формується як елемент множини [10, 12]:

$Decision \in \{Permit, RequireMFA, Deny\}$,

відповідно до порогових значень довіри та ризику:

$$Decision = \begin{cases} Deny, & TL < T_{deny} \text{ або } RiskOp > R_{max}, \\ RequireMFA, & T_{deny} \le TL < T_{mfa}, \\ Permit, & TL \ge T_{mfa}. \end{cases}$$

де T_{deny} та T_{mfa}– порогові значення рівня довіри, що визначають суворість політики доступу, R_{max} – максимально допустимий ризик операції.

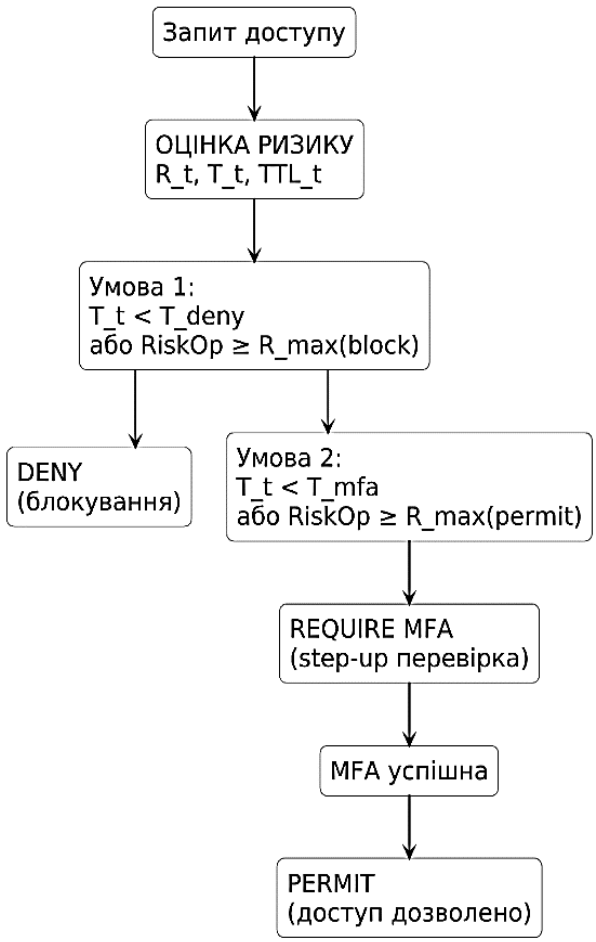

Рис. 3. Діаграма порогових станів і переходів ризик-адаптивної авторизації ($Permit/RequireMFA/Deny$)

На рис. 3 подано діаграму станів ухвалення рішення авторизації в архітектурі

Zero Trust, що реалізує формулу (18) через порогові значення рівня довіри TL та операційного ризику $RiskOp$.

Залежно від співвідношення з порогами $T_{deny}, T_{mfa}, R_{max}$ система переходить у стани $Permit$, $RequireMFA$ або $Deny$, забезпечуючи інтерпретовану та керовану логіку доступу з підтримкою step-up перевірок і блокування.

Параметри моделі (α, β, γ), пороги $(T_{deny}, T_{mfa}, R_{max})$ та коефіцієнти керування токеном (k_1, k_2, k_3, λ) налаштовуються відповідно до політики прийнятного ризику підприємства та історії інцидентів. Практично калібрування може виконуватися як задача мінімізації W за обмежень на експлуатаційні показники: частоту додаткових перевірок $S \leq S_{max}$ і затримку ухвалення рішень $L \leq L_{max}$ [10, 12-14]. Початкові значення параметрів встановлюються експертно (policy-driven), після чого уточнюються на основі журналів доступу та підтверджених інцидентів шляхом підбору, що зменшує W у сценаріях S2–S4 без надмірного зростання S у сценарії S1.

У разі ухвалення рішення Permit або RequireMFA PDP додатково визначає параметри сесії, зокрема оновлене значення часу життя токена TTL_{t+1}, відповідно до запропонованого ризик-адаптивного методу. Отримане рішення разом із параметрами токена передається до PEP, який забезпечує примусове виконання політики доступу, ініціює додаткову автентифікацію або блокує запит.

Таким чином, модель ухвалення рішень авторизації забезпечує тісний зв'язок між оцінкою довіри, ризику та керуванням токенами доступу [13, 15]. Вона слугує перехідною ланкою між формальною моделлю авторизації та практичною реалізацією алгоритму PDP/PEP, створюючи основу для подальшого алгоритмічного опису й експериментального оцінювання ефективності ризик-адаптивної авторизації в умовах Zero Trust.

Алгоритм ризик-адаптивної авторизації $TL - TTL$. Алгоритм ризик-адаптивної авторизації реалізує формалізований процес ухвалення рішень на рівні Policy Decision Point з подальшим примусовим виконанням на рівні Policy Enforcement Point [10, 12, 13]. Його метою є адаптація параметрів авторизації та часу життя токенів доступу до поточного рівня довіри та ризику операцій у межах архітектури Zero Trust.

Вхідними даними алгоритму є запит доступу $q = \langle u, a, p, ctx, t \rangle$, множина політик доступу, поточний рівень довіри $TL_t(u)$, а також параметри порогів і обмежень [13]. Вихідними даними є рішення авторизації та оновлені параметри токена доступу.

Алгоритм ризик-адаптивної авторизації $TL - TTL$ [10, 12-13, 18]:

1. Отримати запит доступу q та зібрати атрибути суб'єкта, активу й контексту.
2. Обчислити поточний рівень довіри $TL(u)$,.
3. Визначити ризик операції $RiskOp = (u, a, p)$.
4. Ухвалити попереднє рішення [11-12]:
   – якщо $TL < T_{deny}$ або $RiskOp > R_{max}$,, то Deny;
   – якщо $T_{deny} \leq TL < T_{mfa}$, то RequireMFA;
   – інакше – Permit.
5. У разі Permit або RequireMFA обчислити базове значення часу життя токена:

$$TTL^{\setminus *} = f(TL, RiskContext, Sensitivity),$$

Якщо зафіксовано ризикову подію, застосувати штрафне коригування:

$$TTL_{t+1} = TTL^{\setminus *} - Penalty(RiskEvent_t), \quad (20)$$

6. Обмежити TTL_{t+1} значеннями TTL_{min} і TTL_{max},.
7. Передати рішення та параметри токена до PEP для виконання.
8. Зареєструвати подію в системі моніторингу безпеки.

Запропонований алгоритм забезпечує безперервну адаптацію параметрів авторизації та дозволяє оперативно реагувати на зміну рівня ризику під час активної сесії.

Сценарії оцінювання та базові моделі. Оцінювання ефективності ризик-адаптивного підходу до авторизації виконується за сценарним методом із порівнянням із базовими моделями доступу [10-13]. Це

дозволяє проаналізувати роботу механізмів авторизації в типових умовах експлуатації як у штатному режимі, так і за підвищених ризиків, з урахуванням загроз компрометації ідентичностей, змін контексту доступу та аномальної поведінки користувачів.

Перший сценарій (S1) відображає нормальний режим доступу, за якого легітимний користувач працює в очікуваному контексті безпеки без фіксації ризикових подій; рівень довіри та параметри авторизації залишаються стабільними. Він використовується як базовий для оцінювання зручності доступу та відсутності надмірних обмежень.

Другий сценарій (S2) моделює контекстну аномалію, спричинену зміною геолокації, типу пристрою або мережевого середовища. У цьому випадку зростає контекстний ризик, що призводить до коригування рівня довіри та перевіряє здатність системи адаптивно реагувати без негайного блокування користувача.

Третій сценарій (S3) пов'язаний із поведінковими аномаліями, такими як нетипова активність або відхилення від звичних шаблонів доступу, і дозволяє оцінити ефективність динамічного зниження довіри та скорочення часу життя токенів у відповідь на ризикові події.

Четвертий сценарій (S4) моделює компрометацію токена під час активної сесії та зосереджується на аналізі вікна атаки, у межах якого зловмисник може зловживати доступом, демонструючи найбільш виразні переваги адаптивного керування часом життя токенів.

Для порівняльного аналізу застосовуються дві базові моделі авторизації: перша використовує статичний час життя токена, незмінний протягом сесії, а друга враховує лише чутливість активу, скорочуючи TTL для критичних ресурсів без урахування динамічних змін довіри та контексту. На відміну від них, запропонований метод поєднує оцінювання рівня довіри та операційного ризику з адаптивним керуванням параметрами токенів доступу.

Оцінювання всіх сценаріїв здійснюється за однакових умов і параметрів доступу, що забезпечує коректність порівняння результатів і дозволяє об'єктивно оцінити вплив запропонованого ризик-адаптивного підходу на безпеку та експлуатаційні характеристики системи авторизації.

Результати та порівняльний аналіз. Оцінювання ефективності запропонованої моделі здійснюється за сценарним підходом (S1–S4), визначеним у розділі методики, із фіксацією середніх значень показників за серією експериментальних запусків. Для кожного сценарію вимірювалися: потенційне вікно атаки W, частота step-up перевірок, затримка ухвалення рішень τ_{dec} та частка необґрунтованих блокувань. Кожен сценарій моделювався не менше, ніж у 30 незалежних ітераціях із фіксацією середніх значень показників.

Обчислювальна складність PDP-оцінювання визначається структурою моделі довіри. Обчислення інтегрального показника TL здійснюється як агрегація L ознак, що зумовлює лінійну складність $O(L)$. Операції обчислення RiskOp та оновлення TTL виконуються за фіксованої кількості параметрів і мають константну складність $O(1)$ [9, 13]. Основний внесок у затримку ухвалення рішення L формують операції отримання контекстних атрибутів і телеметрії (IdP, EDR, NAC), тоді як власне обчислення TL і TTL характеризується незначними накладними витратами.

Результати аналізу підтвердили, що запропонований ризик-адаптивний підхід із динамічним рівнем довіри та керуванням часом життя токенів суттєво підвищує безпеку порівняно з базовими моделями, з найбільшим ефектом у сценаріях компрометації токенів і появи поведінкових або контекстних аномалій.

У сценаріях нормальної роботи (S1) запропонований підхід не призводить до істотного погіршення експлуатаційних характеристик, забезпечуючи затримку PDP/PEP у межах допустимих значень та низьку частоту додаткових перевірок доступу. Водночас у сценаріях підвищеного ризику (S2–S4) динамічне керування часом життя токенів забезпечує суттєве скорочення потенційного вікна атаки – на 75–85 % відносно базової моделі A зі статичним TTL для контекстних і поведінкових аномалій, а в десятки разів у сценарії компроме-

тації токена (S4). Такий ефект досягається за рахунок контрольованого зростання частоти step-up перевірок та помірного збільшення затримки прийняття рішень, що узгоджується з принципами Zero Trust і не порушує безперервність бізнес-процесів. Варіація затримки ухвалення рішень у межах сценаріїв не перевищувала 5–8 %, що свідчить про стабільність роботи моделі за умов зміни контекстних параметрів.

У табл. 1 узагальнено результати порівняльного аналізу, які підтверджують, що запропонований підхід дозволяє зменшити потенційне вікно атак і підвищити адаптивність авторизації без істотного погіршення експлуатаційних характеристик.

Таблиця 1.

Порівняльні результати оцінювання моделей авторизації

Критерій	Базова модель A (статичний TTL токена)	Базова модель B (TTL за чутливістю активу)	Запропонований метод
Потенційне вікно атаки W	$\approx TTL$ (8 год)	↓ ~2×	↓ 10–90×
Реакція на ризикові події	Відсутня	Обмежена	Подієва, динамічна
Частота step-up перевірок	1–3 %	3–6 %	2–15 % (адапт.)
Ймовірність необґрунтованих блокувань	Висока	Середня	Низька
Затримка прийняття рішень τ_{dec}	8–10 мс	10–14 мс	12–25 мс

Показники, наведені в таблиці 1, отримані на основі сценарного моделювання з фіксованими параметрами TTL=8 год для моделі A та диференційованими значеннями TTL за рівнем чутливості активу для моделі B. Для запропонованого методу TTL змінювався адаптивно відповідно до рівня RiskOp і TL.

Як видно з рис. 4, у сценаріях S2–S4 спостерігається нелінійне скорочення W, що зумовлено раннім спрацюванням механізму адаптивного зменшення TTL. У сценарії S4 (компрометація токена) різниця між моделлю A та запропонованим методом є максимальною, що підтверджує ефективність реактивного коригування параметрів сесії.

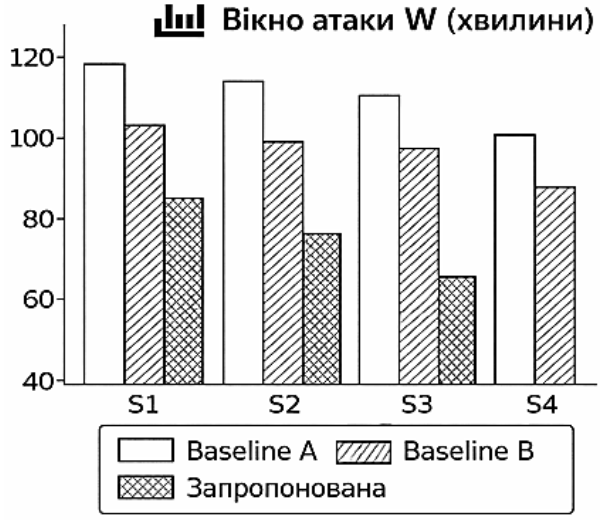

Рис. 4. Порівняльний графік вікна атаки W за сценаріями S1–S4 для Базової моделі A (статичний TTL токена), Базової моделі B (TTL за чутливістю активу) та запропонованого методу.

У випадку використання базової моделі A, що передбачає статичний час життя токена, зловмисник у разі компрометації зберігає можливість доступу до ресурсу протягом усього періоду дії сесії. Це формує значне потенційне вікно атаки, яке не скорочується навіть за наявності додаткових ознак ризику. На відміну від цього, у запропонованому методі час життя токена автоматично коригується у відповідь на зафіксовані ризикові події, що приводить до швидкого обмеження або припинення доступу. Таким чином тривалість потенційного зловживання скомпрометованими обліковими даними суттєво зменшується.

Порівняння з базовою моделлю B, у якій час життя токена визначається виключно чутливістю активу, показало, що такий підхід є недостатньо гнучким у динамічних середовищах. Хоча він дозволяє частково обмежити ризики для критичних ресурсів, відсутність урахування поточного рівня довіри та поведінкових характеристик користувача призводить або до надмірно жорстких обмежень, або до запізнілої реакції на

реальні загрози [11, 13, 18]. Включення динамічної оцінки довіри у запропонованому методі дозволяє більш точно адаптувати параметри авторизації до конкретної ситуації доступу, зменшуючи кількість необґрунтованих блокувань легітимних користувачів і водночас підвищуючи стійкість системи до атак.

Аналіз експлуатаційних характеристик показав, що ризик-адаптивне керування токенами не спричиняє істотного зростання кількості додаткових перевірок доступу: step-up authentication активується переважно у сценаріях підвищеного ризику та залишається мінімальною за нормальної поведінки користувачів [9, 12-14]. Затримки ухвалення рішень у точках реалізації політик не перевищують допустимих меж і не впливають на загальну продуктивність системи.

Отримані результати підтверджують, що ризик-адаптивна авторизація забезпечує перехід від статичної політики $\boldsymbol{TTL}$ до подієво-керованого механізму контролю доступу, в якому параметри сесії безперервно узгоджуються з поточним рівнем довіри та ризику. Це формує формалізований зв'язок між метрикою $\boldsymbol{W}$, динамічною оцінкою $\boldsymbol{TL}$ та механізмом керування токенами, що забезпечує контрольоване зменшення експозиції до атак без непропорційного зростання обчислювальних та експлуатаційних витрат.

Обговорення результатів

Отримані результати демонструють, що перехід від статичних моделей авторизації до ризик-адаптивного керування токенами суттєво підвищує стійкість систем керування доступом (Access Management) до загроз, пов'язаних із компрометацією ідентичностей і сесійних токенів. На відміну від базових підходів із незмінними параметрами доступу протягом усього життєвого циклу сесії, запропонований метод забезпечує зменшення потенційного вікна атаки завдяки динамічному скороченню часу життя токена у відповідь на ризикові події.

У таблиці 2 узагальнено вплив запропонованого підходу на ключові експлуатаційні характеристики системи доступу. Результати підтверджують, що підвищення рівня безпеки досягається без непропорційного зростання обчислювальних витрат і без порушення стабільності роботи сервісів.

Таблиця 2.

Вплив ризик-адаптивної авторизації на експлуатаційні характеристики

Параметр	Тенденція зміни	Практичний ефект
Рівень безпеки	Зростає	Підвищення стійкості до атак
Вікно атаки W	Зменшується	Скорочення часу потенційного зловживання
Частота перевірок доступу	Зростає локально (у S2–S4)	Контрольована адаптація до ризику
Затримка PDP/PEP	Незначне зростання	Придатність до реального використання

Важливо зазначити, що скорочення вікна атаки W має нелінійний характер і прямо залежить від чутливості системи до змін рівня довіри TL. У сценаріях із різкими поведінковими відхиленнями раннє спрацювання механізму адаптивного зменшення TTL забезпечує швидке обмеження сесії, що мінімізує часову експозицію до атак навіть за наявності дійсного токена доступу.

З точки зору системного аналізу запропонований підхід реалізує замкнутий керуючий контур, у якому телеметрія RiskEvent виступає вхідним сигналом, модель довіри TL – регулятором, а час життя токена TTL – керованою змінною. Така інтерпретація дозволяє розглядати процес авторизації як динамічну систему зі зворотним зв'язком, де стабільність визначається балансом між швидкістю реакції на ризик і допустимими коливаннями частоти step-up перевірок. Це створює підґрунтя для подальшої формалізації моделі засобами теорії керування та оптимізації параметрів політики доступу.

Включення динамічного рівня довіри як керуючого параметра авторизації дає змогу точніше відображати поточний стан безпеки та поведінкові характеристики користувача. На відміну від підходів, що враховують лише чутливість активів, запропонований метод забезпечує гнучке коригування доступу залежно від контексту, зменшуючи кількість необґрунтованих блокувань і водночас підвищуючи швидкість реакції на аномальні сценарії, що є критичним для динамічних корпоративних середовищ.

З експлуатаційної точки зору додаткові перевірки активуються переважно у сценаріях підвищеного ризику та не впливають на нормальний режим роботи. Затримки у PDP та PEP залишаються в допустимих межах, що підтверджує практичну придатність моделі для впровадження в системах реального часу. Тож, підхід забезпечує баланс між підвищенням безпеки та зручністю використання відповідно до принципів Zero Trust.

Практичне значення запропонованого підходу полягає у можливості інтеграції механізму адаптивного TTL у наявні IAM/IdP-рішення без зміни архітектури автентифікації. Модель може бути реалізована як надбудова на рівні PDP, що мінімізує витрати на впровадження та забезпечує масштабованість у хмарних і гібридних середовищах.

Отримані результати підтверджують гіпотезу про те, що інтеграція динамічної довіри в механізм керування токенами створює формалізований зв'язок між контекстом доступу, ризиком і тривалістю сесії, перетворюючи TTL із статичного параметра на керовану змінну безпеки.

На відміну від більшості існуючих Zero Trust реалізацій, де механізм авторизації залишається логічно відокремленим від керування криптографічними або сесійними параметрами, запропонований підхід інтегрує ризикову оцінку безпосередньо в механізм управління життєвим циклом токена. Це усуває розрив між політичним рішенням доступу та фактичною тривалістю дії сесії, формуючи єдину адаптивну модель контролю доступу. Така інтеграція забезпечує не лише реактивне, а й превентивне обмеження експозиції до атак.

Обмеження методу пов'язані з якістю та затримками телеметрії RiskEvent: хибнопозитивні події можуть призводити до надмірної активації step-up перевірок, тоді як хибнонегативні події можуть призводити до збереження надмірного TTL та збільшення часової експозиції до атак. Крім того, у розподілених середовищах критичним є коректний вибір часових параметрів (Δt) та калібрування порогів і ваг моделі довіри, оскільки їх некоректне налаштування може спричинити або надмірно жорстку, або занадто лояльну політику доступу.

Подальші дослідження можуть бути спрямовані на використання адаптивних методів машинного навчання для автоматичного калібрування порогів TL і RiskOp, а також на формалізацію метрики оптимального балансу між частотою step-up перевірок і скороченням вікна атаки W. Окремого аналізу потребує вплив затримок телеметрії у мультихмарних і розподілених архітектурах.

Висновки

У роботі запропоновано ризик-адаптивний підхід до авторизації в архітектурі Zero Trust, що базується на динамічній оцінці рівня довіри та керуванні часом життя токенів доступу. Розроблена модель дозволяє перейти від статичних механізмів контролю доступу до адаптивної авторизації, у якій параметри доступу коригуються залежно від поточного рівня ризику та поведінки користувача.

Результати аналізу показали, що застосування динамічного керування токенами приводить до зменшення потенційного вікна атак у сценаріях компрометації токенів і поведінкових аномалій. Порівняно з базовими моделями запропонований метод забезпечує більш точну реакцію на ризикові події, обмежує час зловживання доступом і водночас знижує ймовірність необґрунтованих відмов у доступі для легітимних користувачів. Частота ініціювання додаткових перевірок доступу зростає контрольовано та не призводить до істотного

погіршення експлуатаційних характеристик системи.

Практична значущість отриманих результатів полягає в можливості використання запропонованого підходу як механізму реалізації принципів Zero Trust у корпоративних інформаційних системах без радикальної перебудови існуючої IAM/PAM-інфраструктури. Метод може бути інтегрований у сучасні системи авторизації, що використовують токеноорієнтовані протоколи, та адаптований до різних профілів ризику підприємства.

Подальші дослідження доцільно спрямувати на розширення набору сценаріїв оцінювання з урахуванням складних багатокрокових атак, інтеграцію запропонованої моделі з системами UEBA та SIEM для автоматизованого формування ризикових подій, а також на кількісну оцінку ефективності підходу в реальних виробничих середовищах. Окремий інтерес становить дослідження методів автоматичного налаштування порогових значень рівня довіри та параметрів керування токенами з урахуванням політики прийнятного ризику організації.

Література

1. Glöckler J., Sedlmeir J., Frank M., Fridgen G. A systematic review of identity and access management requirements in enterprises and potential contributions of self-sovereign identity // Business & Information Systems Engineering. 2024. Vol. 66. P. 421–440. DOI: https://doi.org/10.1007/s12599-023-00830-x
2. Aljohani A. Zero-trust architecture: Implementing and evaluating security measures in modern enterprise networks // SHIFRA. 2023. P. 1–13. DOI: https://doi.org/10.70470/SHIFRA/2023/008
3. Glöckler J., Sedlmeir J., Frank M., Fridgen G. Publisher correction: A systematic review of identity and access management requirements in enterprises and potential contributions of self-sovereign identity // Business & Information Systems Engineering. 2023. DOI: https://doi.org/10.1007/s12599-023-00838-3
4. Aftab M. U., Qin Z., Hundera N. W., Ariyo O., Zakria Z., Son N. T., Dinh T. V. Permission-based separation of duty in dynamic role-based access control model // Symmetry. 2019. Vol. 11, no. 5. Art. 669. DOI: https://doi.org/10.3390/sym11050669
5. Yadav V., Soni M. K., Pratap A. Secured identity and access management for cloud computing using zero trust architecture // Cryptology and Network Security with Machine Learning (ICCNSML 2023). Lecture Notes in Networks and Systems. Vol. 918 / Eds. A. Chaturvedi et al. Springer, 2024. DOI: https://doi.org/10.1007/978-981-97-0641-9_47
6. Sivaraman H. Zero trust identity and access management (IAM) in multi-cloud environments // ESP Journal of Engineering & Technology Advancements. 2023. Vol. 3. DOI: https://doi.org/10.56472/25832646/JETA-V3I6P108
7. Ahmadi S. Zero trust architecture in cloud networks: Application, challenges and future opportunities // Journal of Engineering Research and Reports. 2024. Vol. 26, no. 2. P. 215–228. DOI: https://doi.org/10.9734/jerr/2024/v26i21083
8. Lee J., Tang F., Thet P. M., Yeoh D., Rybczynski M., Mon Divakaran D. SIERRA: Ranking anomalous activities in enterprise networks. arXiv preprint, 2022. DOI: https://doi.org/10.48550/arXiv.2203.16802
9. Kostiuk Y., Rzaieva S., Khorolska K., Mazur N., Korshun N. Architecture of the software system of confidential access to information resources of computer networks // Proceedings of the Workshop Cyber Security and Data Protection (CSDP 2025). Vol. 4042. CEUR-WS, 2025. P. 37–53.
10. Teerakanok S., Uehara T., Inomata A. Migrating to zero trust architecture: Reviews and challenges // Security and Communication Networks. 2021. Art. 9947347. DOI: https://doi.org/10.1155/2021/9947347
11. Костюк Ю., Бебешко Б., Крючкова Л., Литвинов В., Оксанич І., Складанний П., Хорольська К. Захист інформації та безпека обміну даними в безпроводових мобільних мережах з автентифікацією і протоколами обміну ключами // Кібербезпека: освіта, наука, техніка. 2024. № 1(25). С. 229–252. DOI: https://doi.org/10.28925/2663-4023.2024.25.229252
12. Phiayura P., Teerakanok S. A comprehensive framework for migrating to zero trust architecture // IEEE Access. 2023. Vol. 11. P. 19487–19511. DOI: https://doi.org/10.1109/ACCESS.2023.3248622
13. Syed N. F., Shah S. W., Shaghaghi A., Anwar A., Baig Z., Doss R. Zero trust architecture (ZTA): A comprehensive survey // IEEE Access. 2022. Vol. 10. P. 57143–57179. DOI: https://doi.org/10.1109/ACCESS.2022.3174679

14. Складанний П., Костюк Ю., Рзаєва С., Мазур Н. Паралельна обробка даних у розширюваних хеш-структурах та оцінка їх продуктивності // Кібербезпека: освіта, наука, техніка. 2025. № 3(31). С. 242–269. DOI: https://doi.org/10.28925/2663-4023.2025.31.1015
15. Amanlou S., Doss R., Li J. Implementing a dynamic and context-aware trust evaluation model for zero trust architecture (ZTA): A fuzzy logic approach // Proceedings of the 2025 International Wireless Communications and Mobile Computing (IWCMC). IEEE, 2025. P. 404–411. DOI: https://doi.org/10.1109/IWCMC65282.2025.11059668
16. Muhammad A. R., Sukarno P., Wardana A. A. Integrated security information and event management (SIEM) with intrusion detection system (IDS) for live analysis based on machine learning // Procedia Computer Science. 2023. Vol. 217. P. 1406–1415. DOI: https://doi.org/10.1016/j.procs.2022.12.269
17. Костюк Ю., Складанний П., Рзаєва С., Самойленко Ю., Коршун Н. Інтелектуальні системи керування та захисту в кіберфізичних і хмарних середовищах Smart Grid // Кібербезпека: освіта, наука, техніка. 2025. № 2(30). С. 125–156. DOI: https://doi.org/10.28925/2663-4023.2025.30.956
18. P. S. N., Pimpalkar A., Shelke N., Bahadur Saini D. K. J. Zero trust architectures empowered by AI: A paradigm shift in cloud and edge cybersecurity // Proceedings of the 3rd International Conference on Sustainable Computing and Data Communication Systems (ICSCDS 2025). IEEE, 2025. P. 328–335. DOI: https://doi.org/10.1109/ICSCDS65426.2025.11166875
19. He Y., Huang D., Chen L., Ni Y., Ma X. A survey on zero trust architecture: Challenges and future trends // Wireless Communications and Mobile Computing. 2022. Art. 6476274.
20. Цирканюк Д., Соколов В. Методика розслідування інцидентів інформаційної безпеки // Кібербезпека: освіта, наука, техніка. 2024. № 2(26). С. 140–154. DOI: https://doi.org/10.28925/2663-4023.2024.26.675
21. Kostiuk, Y., Skladannyi, P., Sokolov, V., Rzaieva S., Khorolska, K. Machine learning methods for detecting intrusions based on network traffic analysis. Proceedings of the Cybersecurity // Providing in Information and Telecommunication Systems II (CPITS-II 2025), October 26, 2025, Kyiv, Ukraine, Vol-4145, P. 72-94. ISSN 1613-0073

Дата першого надходження до видання: 12.02.2026
Внутрішня рецензія отримана: 19.02.2026
Зовнішня рецензія отримана: 01.03.2026
Дата прийняття статті до друку: 19.03.2026
Дата публікації: 16.04.2026

Про авторів:

[1] Костюк Юлія Володимирівна,
PhD in Computer Science
[1] Kostiuk Julia,
PhD in computer science
http://orcid.org/0000-0001-5423-0985

[1] Складанний Павло Миколайович,
к.т.н., доцент
[2] Skladannyi Pavlo,
Ph.D (technical sciences), associate professor
http://orcid.org/0000-0002-9457-7454.

[2] Гнатченко Дмитро Дмитрович,
PhD in Computer Science
[1] Hnatchenko Dmytro,
PhD in computer science
http://orcid.org/0000-0002-7775-6039.

Місце роботи авторів:

[1] Київський столичний університет
імені Бориса Грінченка
[1] Borys Grinchenko
Kyiv Metropolitan University
тел. +38-044-272-19-02
E-mail: kubg@kubg.edu.ua
Сайт: https://kubg.edu.ua/

[2] Державний торговельно-економічний
Університет
[2] State University of Trade and Economics
тел. +38-044-531-49-84
E-mail: knute@knute.edu.ua
Сайт: https://knute.edu.ua/

УДК 004.94 https://doi.org/10.15407/pp2026.01.082

О.В. Захарова, Л.О. Спекторовська

ВИКОРИСТАННЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ПРОЦЕСУ АДМІНІСТРУВАННЯ КОСМЕТОЛОГІЧНИХ ПОСЛУГ

Робота присвячена вирішенню проблеми покращення якості надання косметологічних послуг у випадку стрімкого масштабування прикладної системи. Це супроводжується створенням великої кількості нових ролей та видів послуг, суттєвим розширенням клієнтської бази й комунікаційної мережі та, відповідно, значним збільшенням обсягів інформації, що потребує обробки. Метою дослідження є вироблення підходів, що дозволили б підвищити ефективність адміністрування косметологічних послуг шляхом автоматизації обробки вхідних повідомлень та їхньої багатокритеріальної категоризації. Як критерії для категоризації виділені: тип повідомлення, пріоритет, фах спеціаліста, що з ним пов'язаний, вид послуги.
В роботі також виконано огляд існуючих підходів з урахуванням постановки прикладної задачі, що дозволив дійти висновку про доцільність використання комбінації попередньої обробки текстових даних, методів витягнення ознак із класичними моделями машинного навчання для досягнення поставленої мети.
Ключові слова: машинне навчання, класифікація текстів, маршрутизація заявок, системи на основі правил, гібрідні системи, трансформери, категоризація повідомлень, прогнозування, навчальні дані

O.Zakharova, L. Spektorovska

USING MACHINE LEARNING METHODS TO IMPROVE THE EFFICIENCY OF THE COSMETOLOGICAL SERVICES ADMINISTRATION PROCESS

The article is devoted to solving the problem of improving the quality of cosmetic service provision during the rapid scaling of an applied system. This process is accompanied by the creation of a large number of new roles and types of services, a significant expansion of the client base and communication network. Accordingly, it also increases significantly the volume of information that requires processing. The aim of the study is to develop approaches that would increase the efficiency of administering cosmetic services through the automation of incoming message processing and their multi-criteria categorization. The criteria identified for categorization are: message type, priority, the specialist, and the type of service.
The paper also includes a review of existing approaches, taking into account the formulation of the applied task. This allows to conclude: to achieve the stated objective it is advisable to use a combination of text data preprocessing, feature extraction methods, and classical machine learning models.
Keywords: machine learning, text classification, rule-based systems, gibrid systems, transformers, message categorization, routing of requests, prediction, training data

Вступ

Вимоги та забезпечення ефективності будь-якої системи у сфері надання послуг насамперед обумовлюються розміром цієї системи. І сфера косметологічних послуг не є винятком. Якщо це косметологічний кабінет з мінімальною кількістю ролей, що забезпечують виконання базових функцій, то автоматизованого робочого місця зі стандартними можливостями обліку клієнтів, ведення їх запису, контролю розкладу завдань та надання пояснювальної (описової) інформації з доступом до веб є цілком достатнім. Зокрема, мінімальна кількість ролей в системі не передбачає реалізації складної автоматичної маршрутизації завдань. Але в процесі масштабування різко зростає на лише набір ролей та функцій системи, а й обсяг бази клієнтів і мережа комунікацій, включаючи джерела надходження інформації. Велику кількість різнотипних клієнтських заявок на отримання послуг потрібно «на льоту» класифікувати і розподіляти між ролями системи. Заявки можуть надходити до адміністраторів клі-

ніки з різних джерел: телефоном, з чату сайту клініки, електронною поштою тощо. Фактично це текстові повідомлення довільної форми, природною мовою різних форматів.

Це критично збільшує навантаження на адміністраторів косметологічної клініки, суттєво збільшує трудомісткість процесу адміністрування. Ручна обробка великої кількості різноманітних заявок породжує ризики помилкового розподілу задач та негативно впливає на ефективність роботи клініки в цілому. Тому задача автоматизації обліку та класифікації заявок на косметологічні послуги набуває актуальності під час масштабування системи надання послуг. А її вирішення потребує залучення сучасних технологій для виявлення семантик в природномовних контентах та подальшої динамічної семантичної класифікації текстових повідомлень, що надходять з різних джерел у різних форматах.

Постановка задачі

Задача полягає у динамічному зборі та обробці текстових повідомлень, що надходять з різних джерел. Джерелами вхідних повідомлень можуть бути: листи електронної пошти, зафіксовані письмово оператором call -центру/адміністратором в електронному журналі звернення по телефону, повідомлення з месенжерів Viber або WhatsApp, повідомлення з чату сайту клініки. Результатом має бути визначення категорії повідомлення за різними критеріями (рис.1), а саме:

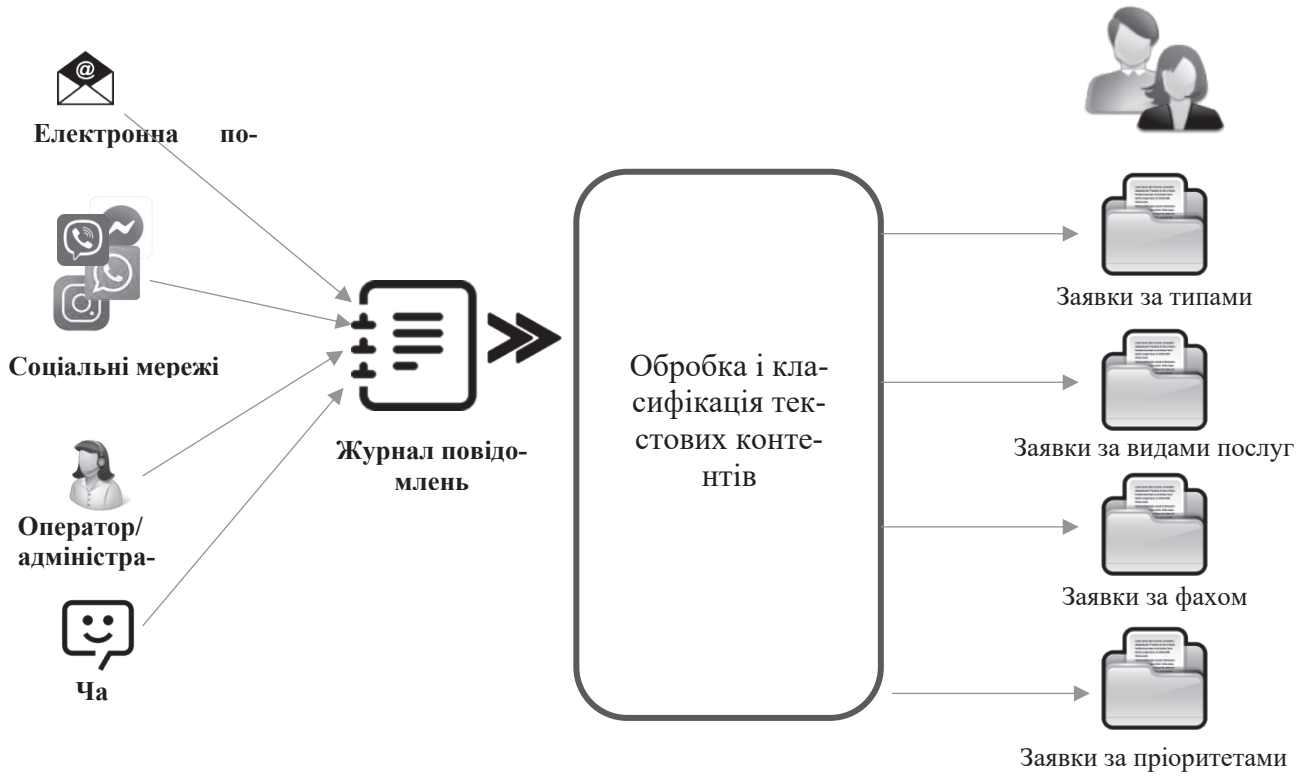

Рис. 1. Постановка задачі

– *Тип повідомлення* (запис на первинну консультацію, запис на косметологічну процедуру, відгук/скарга, технічне питання, адміністративне питання, питання/уточнення, запит на покупку косметологічних препаратів, спам тощо).

– *Пріоритет* (термінова заявка/звичайна).

– *Фах спеціаліста*, якому спрямована, можливо пряме призначення за прізвищем спеціаліста.

– *Вид послуг* (дерматологія, ін'єкційна косметологія, масаж, трихологія тощо).

Ведення журналу повідомлень має забезпечувати його динамічне поповнення

новими даними з різних джерел вхідної інформації, оновлення у разі зміни характеристик і стану заявок та видалення застарілих.

Як базові характеристики повідомлення можна виділити: номер (id); дата надходження; вид джерела надходження (чат, оператор, viber, whatsapp, дані про автора (електронна пошта, телефон, viber, whatsapp тощо), Instagram, messenger, електронна пошта); контент; дата останнього оновлення; статус заявки (нова, класіфікована, закрита, виконана); категорія заявки; вид послуг, якого вона стосується; фах і прізвище спеціаліста, якому спрямована; пріоритет.

Основна мета реалізації функції семантичного аналізу тексту в цій задачі полягає в класифікації текстових повідомлень на фіксовану кількість наперед визначених категорій, де одне повідомлення (в загальному випадку) може потрапити до кількох категорій, до однієї категорії, або не потрапити до жодної.

Тобто результатом обробки і класифікації текстових контентів має бути визначення характеристик класифікації (тип заявки; вид послуг, яких стосується; фах і прізвище спеціаліста, якому призначена; пріоритет заявки) на основі аналізу виявлених у контенті семантичних елементів. Загальна схема процесу представлена на рис.2.

Рис. 2. Загальна схема процесу обробки повідомлень

Огляд існуючих підходів

Головне завдання, що потребує вирішення в сформульованій вище задачі, це автоматична класифікація текстів. Слід зазначити, що автоматична класифікація тексту є однією з базових задач обробки природної мови. Умовно можно виділити три групи систем автоматичної класифікації текстів [1]: системи на основі правил, системи на основі машинного навчання та гібридні системи.

Системи на основі правил. Системи на основі правил [2] є одним з найпростіших підходів до класифікації. Для категоризації тексту вони використовують набір мовних правил, розроблених заздалегідь вручну. Кожне правило складається з шаблону та визначення категорії, що відповідає цьому шаблону. Тобто вони наказують системі класифікувати текст у певну категорію на основі його змісту, використовуючи в шаблоні семантично релевантні текстові елементи. Слід зазначити,

що за одним шаблоном в загальному випадку текстовий елемент може потрапляти до більш ніж однієї категорії.

Наприклад, можна визначити правило, що, якщо текст повідомлення містить слова «підліток» та «акне», то повідомлення належить до категорії видів послуг *Підліткова дерматологія*, а до категорії типів повідомлень *Спам* віднести тексти, що включають одне з наступних слів: «акція», «знижка», «розпродаж», «виграти», «не пропустіть» тощо. Повідомлення з назвою косметологічного препарату може бути віднесено як до категорії *Запит на покупку косметологічного препарату*, так і до «Питання/уточнення» чи «Відгук/скарга». Найпоширенішими формами правил, що використовуються в системах класифікації текстів, є регулярні правила (шаблони), правила на ключові слова, дерева рішень, словники тощо.

Досить відомими є такі реалізації систем на основі правил, як спам-фільтри [3], системи класифікації заявок (Helpdesk / Service desk routing), системи класифікації новин за словниками, системи виявлення тональностей за правилами (Vader [4], словники SentiWordNet [5]), платформи Rasa rules [6] та Dialogflow rules [7] – класифікація намірів у чат-ботах, які використовують як правила на ключові слова, так і шаблони й прості деревоподібні правила тощо.

Перевагою систем на основі правил є їхня простота і зрозумілість, до недоліків можна віднести їхню трудомісткість (вимагають багато часу для ручного створення правил, ретельного вивчення предметної області й тестування) та складність у масштабуванні (додавання нових правил може змінити результати вже існуючих). Також системи на основі правил є складними в обслуговуванні та масштабуванні.

Системи на основі машинного навчання. Задача класифікації є однією з класичних задач машинного навчання (ML). Для автоматичної класифікації текстової інформації системи на основі ML використовують різноманітні алгоритми та моделі. Вони зазвичай навчаються на зразках розмічених текстів (навчальна вибірка). Це дозволяє під час навчання зрозуміти певні закономірності у вхідних текстах і в подальшому правильно класифікувати нові зразки.

Серед найбільш використовуваних класичних моделей машинного навчання для класифікації текстів варто виділити наступні.

Multinomial Naive Bayes (MNB) [8] алгоритм належить до родини наївних Баєсівських моделей. Це клас моделей, заснований на теоремі Баєса та припущенні умовної незалежності ознак від заданої мітки класу. Незалежність ознак передбачає, що у встановленні належності елемента певної категорії, яка визначається множиною ознак, кожна ознака елемента розглядається незалежно від інших. Такий підхід є ефективним для вирішення задач, де дискримінативні слова домінують у приналежності до класу. Мультиноміальна модель, як правило, використовується саме для класифікації текстів, де для кожної цільової категорії y, що визначається n ознаками, розподіл визначається параметризованим вектором $\theta_y = (\theta_{y_1}, \dots, \theta_{y_n})$, де θ_{y_i} – ймовірність ознаки i в екземплярі класу y, що обчислюється підрахунком відносної частоти появи ознаки i в класі y в тренувальному наборі даних.

Слід зазначити, що припущення про незалежність ознак не завжди є вірним, і в таких випадках метод може бути не достатньо ефективним. Але, попри це, наївна баєсівська модель часто забезпечує доволі конкурентоспроможну продуктивність у задачах класифікації коротких текстів.

Linear SVM (Support Vector Machine) [9] - лінійний класифікатор, що базується на принципі мінімізації структурного ризику [10] з теорії обчислювального навчання. Ідея полягає у знаходженні гіпотези h, для якої можна гарантувати найменшу істинну похибку. Істинна похибка гіпотези h - це ймовірність того, що h дасть помилку на не- побаченому та випадково вибраному тестовому прикладі. Верхня межа може бути використана для зв'язку істинної похибки гіпотези h з похибкою цієї гіпотези на

навчальному наборі даних та складністю простору гіпотез, що містить h, H (що вимірюється розмірністю Vapnik–Chervonenkis (VC) [11]). SVM знаходить таку гіпотезу *h*, яка мінімізує (наближено) цю межу істинної похибки шляхом ефективного та результативного контролю VC-розмірності простору H.

Якщо набір тренувальних даних представити як множину точок $(x_i, y_j), i = \overline{1,n}, j = \overline{1,p}$, де x_i – текстовий елемент, що підлягає класифікації, а y_j – визначає належність x_i до певного класу j, і може приймати одне з двох значень: 1 – належить, -1 – не належить. То мета SVM полягає у розділенні всієї множини точок xi, для яких $y_j = 1$, від тих точок, для яких $y_j = -1$, гіперплощиною (межу) з максимальним «зазором» і забезпечити мінімальну похибку класифікації. Результативна оцінка визначається до цієї межі.

SVM є дуже універсальними навчальними системами, які у своїй базовій формі вивчають лінійну порогову функцію. Але у разі певного вдосконалення (простим «підключенням» відповідної функції ядра), можуть бути використані й для навчання поліноміальних класифікаторів, мереж радіальних базових функцій (RBF) та тришарових сигмоподібних нейронних мереж.

Logistic Regression [12] ще один метод лінійної класифікації, що моделює ймовірність належності текст до категорії на основі ознак типу Bag-of-Words або TF-IDF. Загалом лінійна класифікація стала одним із найперспективніших методів навчання для великих розріджених даних із величезною кількістю екземплярів і ознак. Логістична регресія, як і SVM, оперує даними як множиною точок $(x_i, y_j), i = \overline{1,n}, j = \overline{1,p}$, де x_i – текстовий елемент, що підлягає класифікації, а y_j – визначає належність x_i до певного класу j, і може приймати одне з двох значень: 1 – належить, -1 – не належить. Обидва методи вирішують ту саму задачу оптимізації, але використовують різні функції втрат. На відміну від SVM, результатом методу логістичної регресії є ймовірність того, що x_i належить класу *j*.

Серед нейронних моделей (глибокого навчання) для класифікації текстів найбільш використовуваними на сьогодні є:

TextCNN [13], як і більшість моделей глибокого навчання, що працюють з природним текстом, розглядає текст як послідовність векторних представлень слів (word embeddings). Ідея полягає в тому, що слова проєктуються з розрідженого кодування 1-з-N (де N – розмір словника) на векторний простір нижчої розмірності через прихований шар. Ці вектори слів по суті є витяжкою ознак, що кодують семантичні ознаки слів у їхніх вимірах. У таких щільних представленнях семантично близькі слова також є близькими (за евклідовою або косинусною відстанню) й у векторному просторі нижчої розмірності.

CNN (згорткові нейронні мережі) використовують шари зі згортковими фільтрами, які застосовуються до локальних ознак. Згорткові фільтри ковзають по послідовності векторів слів та “виявляють” локальні шаблони: характерні n-грамні шаблони, ключові фрази, шаблони емоційної лексики, типові для певної категорії слова.

Модель TextCNN є досить ефективною і швидкою в навчанні й успішно працює на коротких текстах (запити, відгуки, спам тощо), але, слід зазначити, що якість отриманого результату напряму залежить від якості векторного представлення слів.

RNN/LSTM/GRU. Рекурентні нейронні мережі (RNN) [14] – клас нейромереж, що був спеціально розроблений для роботи з такими послідовностями даних (sequence data) як текст. На відміну від класичних моделей, RNN обробляють текст покроково (слово за словом) і мають прихований стан, що дозволяє переносити інформацію з попередніх кроків. Фактично наступний крок (прихований стан на кроці *t*) є нелінійною функцією, що враховує векторні представлення слів на даному кроці, попередній крок (прихований стан на кроці *t-1*) і деякі параметри моделі.

LSTM є модифікацією RNN, що була створена для вирішення проблеми довгих залежностей, яка існує в RNN.

LSTM вводить комірки пам'яті (memory cell) та гейти (gates), які контролюють: що саме треба запам'ятати, що забути, а що передати далі в наступний стан. GRU є спрощеним варіантом LSTM, який має менше параметрів і швидше навчається.

Також слід виділити групу трансформерних моделей, які зараз є найпопулярнішими і активно використовуються в системах обробки природньої мови (NLP). Вони є також класом нейромережевих архітектур, але, на відміну від вище розглянутих, базуються не на рекурентних зв'язках, а на механізмі самоуваги, що є їхньою ключовою інновацією. Механізм самоуваги дозволяє моделі фіксувати контекстуальні зв'язки між усіма токенами в послідовності.

Механізм самоуваги дозволяє моделі визначати, на які слова в реченні потрібно звернути увагу, щоб краще інтерпретувати поточне слово. Ця властивість особливо важлива для розуміння неоднозначних фраз, довгострокових залежностей та полісемії.

Трансформерна модель була вперше запропонована 2017 року і швидко стала основою більшості сучасних моделей NLP, включаючи *BERT* та *RoBERTa***,** що є найвикористовуванішими серед моделей цієї групи.

BERT (Bidirectional Encoder Representations from Transformers) [15] – найбільш поширена модель для класифікації текстів, що спочатку навчається на великих мовних корпусах з використанням самоконтрольованих цілей, а потім налаштовується для виконання конкретних завдань, зокрема, класифікації тексту. Точне налаштування зазвичай вимагає додавання класифікаційної структури (наприклад, онтології) після трансформаторного кодера та навчання моделі на позначених прикладах.

BERT усуває обмеження односпрямованості, коли мовна модель попередньо навчається зліва направо, використовуючи «моделі маскованої мови» (MLM) як мету попереднього навчання. Ідея полягає в тому, що модель маскованої мови випадковим чином маскує деякі токени з вхідних даних, а метою є прогнозування оригінального ідентифікатора словника замаскованого слова лише на основі його контексту. MLM дозволяє представленню об'єднати лівий та правий контексти, що дозволяє попередньо навчити глибокий двонаправлений трансформатор. На додаток до моделі маскованої мови використовується також завдання «передбачення наступного речення», що разом з MLM (спільно) попередньо навчає представлення текстових пар.

Порівняно з методами на основі частотних векторів, класифікатори на основі BERT мають змогу краще узагальнювати синоніми та різні формулювання одного й того ж запиту, забезпечуючи високу якість отриманого результату.

Модель RoBERTa [16] є покращеним варіантом навчання моделі BERT. Внесені до BERT модифікації включають: довше навчання моделі, з більшими пакетами, на більшій кількості даних; видалення цілі прогнозування наступного речення; навчання на довших послідовностях; та динамічну зміну шаблону маскування, що застосовується до навчальних даних.

Окрім цього, RoBERTa збирає новий набір даних (CC-NEWS) досить великого розміру порівняно з іншими приватно використовуваними наборами даних, що дозволяє краще контролювати вплив розміру навчального набору на результат. Дане покращення моделі показало, що використання більшої кількості даних на етапі попереднього навчання моделі значно покращує продуктивність вирішення задачі.

XLM-R (XLM-RoBERTa) [17] – це багатомовна трансформерна модель типу encoder-only, побудована на архітектурі RoBERTa і дозволяє отримувати контекстні представлення тексту більш, ніж 100 мовами. Модель є розвитком одразу трьох ідей, а саме: підходу до двоспрямованого кодування контексту BERT, ідеї оптимізованого навчання, що реалізована в RoBERTa, та технології «cross-lingual language modeling» (XLM).

XLM-R навчається за раніше згаданою схемою MLM. Її головною відмінністю є використання дуже великого багатомов-

ного корпусу для попереднього навчання моделі. Це визначає головний напрямок її застосування – вирішення багатомовних задач, де XLM-R показує високу якість результату. Прикладами таких задач може бути: багатомовна класифікація тексту, аналіз тональності для різних мов, обробка тексту з міжмовними переходами (приміром, попереднє навчання моделі англійською мовою, а працює українською).

Слід зазначити, що моделі BERT, RoBERTa та XLM-R досягають найвищої точності, коли межі класів залежать від семантичного контексту, а не від ключових слів. Їхніми основними недоліками є досить високі, особливо порівняно з класичними ML моделями, обчислювальні вимоги та довший час навчання й логічного висновку.

Відомі на сьогодні *Великі Мовні Моделі (LLM)* поки залишаються досить дорогим рішенням для класифікації текстів із доволі непередбачуваним результатом, хоча в цілому, непогано працюють для складних категорій і мультимовних вхідних текстів.

Гібрідні підходи. Гібрідні методи класифікації текстів [1] поєднують два або більше різні типи методів, моделей/представлень або наборів ознак, для досягнення кращої якості, швидкості або продуктивності. Наприклад, це може бути комбінація в одній моделі частотного методу побудови вектора лексичних ознак TF-IDF, що дозволяє ефективно виявляти ключові слова в тексті, а також - нейромережевих технік embeddings [13], які гарно розуміють контекст та виявляють синонімію в тексті. Інший приклад - поєднання системи на основі правил і машинного навчання. Тоді частина класів формується за допомогою правил, а решта - методами машинного навчання. Існують також інші варіанти побудови гібрідних моделей, що пропонують комбінацію різноманітних методів в одній моделі. Таким чином гібрідні моделі є потужним інструментом класифікації, який може бути оптимізований та налаштований до вимог конкретної задачі чи прикладної системи, дозволяючи досягти високих показників точності та ефективності і не лише для вирішення завдань класифікації. Однак, з іншого боку, наслідком поєднання різних методів може бути суттєве підвищення складності розробки, підтримки та масштабування самої системи. Складність гібрідних систем досі залишається їхнім недоліком, який не можна недооцінювати.

Опис процесу категорізації повідомлень

З огляду на поставлену задачу, перш за все, задачу категорізації заявок на послуги, найбільш прийнятним видається комбінація методу витягнення властивостей з одним із класичних методів машинного навчання (навчання з вчителем або кероване машинне навчання), як, наприклад, згадані вище Naive Bayes, Logistic Regression або Linear SVM. Це швидкі та прості в розгортанні надійні базові моделі, які досягають високої продуктивності, якщо класи добре розділені словником термінів, що розглядаються як ключові слова. Доцільність використання саме класичних моделей для вирішення поставленої задачі підтверджується фактом їхнього застосування у багатьох реальних промислових задачах, зокрема, для маршрутизації заявок клієнтів для служби підтримки.

Методи керованого машинного навчання вивчають співставлення вхідного необробленого тексту з мітками (що відомі також як цільові змінні). Тобто алгоритм контрольованої класифікації навчається на певному наборі вхідних необроблених текстів для прогнозування категорії.

Серед методів витягнення ознак із тексту (feature extractor) найпоширенішими є TF-IDF та Bag of Words (відомий також як CountVectorizer). Головною метою цих методів є перетворення текстових даних (рядків) на вектор числових ознак, що може бути поданий на вхід моделі машинного навчання. Зазвичай це і є першим кроком у поетапному вирішенні задачі класифікації засобами класичного ML.

Обидва названі методі є простими способами представлення текстових даних як числових ознак на основі частотного аналізу тексту. Модель Bag of Words (BoW) [2], часто перекладається як «мішок слів»,

передбачає створення словника відомих слів у корпусі, а потім створення вектора для кожного документа, який містить підрахунок частоти появи кожного слова.

TF-IDF [18] є ще одним способом представлення тексту як числових ознак. Модель TF-IDF відрізняється від BoW тим, що враховує частоту слів у документі, а також обернену частоту документа. Тобто, TF-IDF має вищу ймовірність знаходження ключових слів, ніж BoW. Розглянемо даний метод трохи детальніше.

TF (Term Frequency) – це частота слова в тексті, тоді як IDF (Inverse Document Frequency) навпаки визначає, наскільки рідко дане слово зустрічається у колекції текстових повідомлень. Здебільшого TF визначається як кількість входжень терміна t у текстове повідомлення d - TF(t,d), або ця оцінка може бути нормалізованою і враховувати загальну кількість слів у повідомленні:

$$\mathrm{TF(t,d)} = \frac{tf(t,d)}{|d|}$$

Інверсна оцінка («рідкість» терміна у колекції всіх поданих на аналіз текстових повідомлень) обчислюється відповідно:

$$\mathrm{IDF}(t,d) = log\left(\frac{N}{df(t)}\right),$$

де N – кількість повідомлень у колекції, $df(t)$ – кількість повідомлень, що містять слово t.

Тоді результуюча оцінка обчислюється як добуток оцінок, наведених вище:

$$\mathrm{TFIDF}(t,d) = TF(t,d) * IDF(t)$$

Фактично найвищу вагу отримують слова, що часто зустрічаються в конкретному повідомленні, але не дуже часто в решті повідомлень. Тобто вони є характерними саме для цієї заявки і тому мають більший вплив.

Як і будь-яка інша задача керованого машинного навчання, задача класифікації тексту включає два етапи: навчання та прогнозування. Перший етап полягає у навчанні моделі на певному тренувальному наборі релевантних розмічених текстових даних. Після цього навчена модель може бути використана для прогнозування міток (категорій) для нових та невидимих даних.

Також слід зазначити, що значну роль у підвищенні ефективності обробки текстів природної мови відіграє попередня

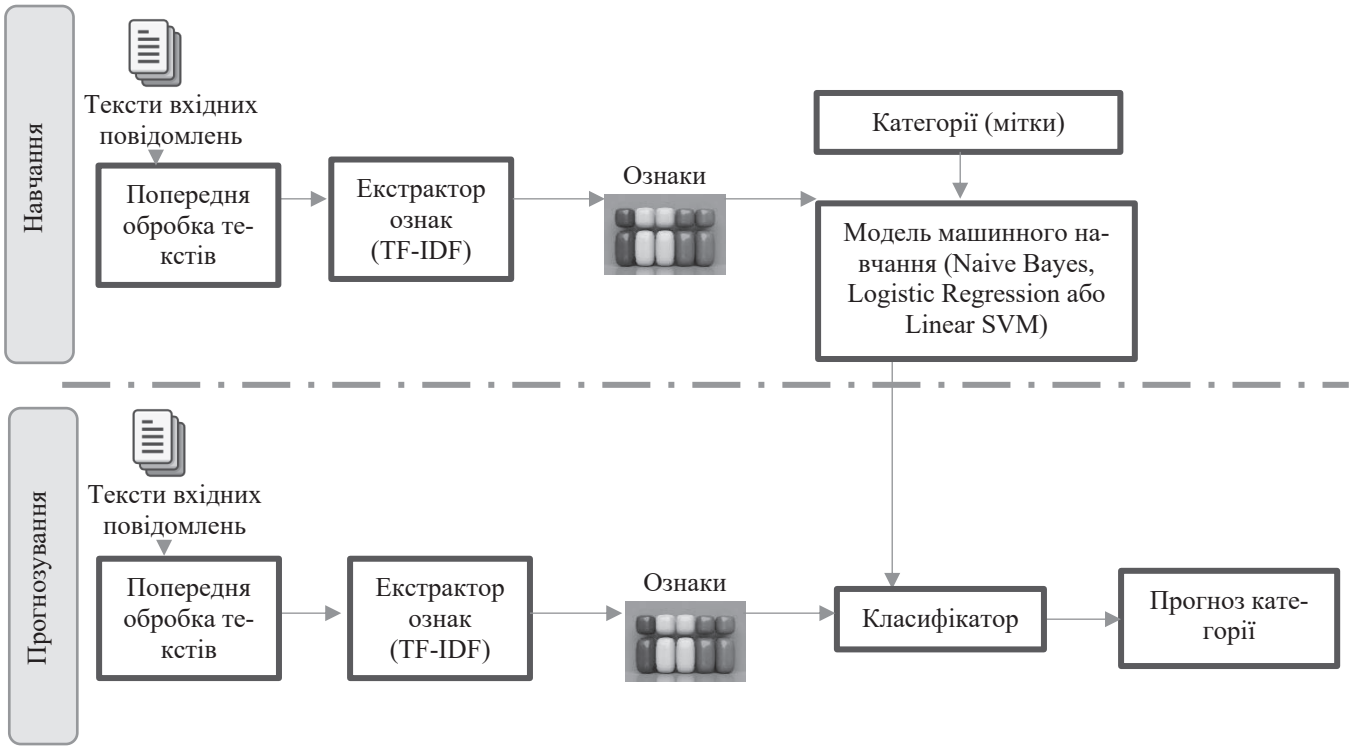

Рис. 3. Категорізація повідомлень на основі ML підходу

підготовка текстових даних для аналізу. Текстові дані неструктуровані, часто містять багато шуму. Це можуть бути орфографічні помилки, граматичні помилки, нестандартне форматування тощо. Попередня підготовка дозволяє очистити цей шум та полегшити подальший аналіз/обробку тексту. Набір кроків попередньої обробки тексту може відрізнятися, але зазвичай він включає такі завдання, як: токенізація, видалення стоп-слів, стеммінг та лематизація. Ці кроки допомагають зменшити розмір текстових даних, підвищити точність завдань обробки природної мови, зокрема, класифікації тексту.

Загальна схема вирішення задачі категорізації повідомлень наведена на рис.3.

Висновки

Метою даного дослідження є підвищення ефективності надання косметологічних послуг за рахунок автоматичної класифікації та маршрутизації заявок на послуги, а також повідомлень клієнтів. Аналіз найбільш використовуваних на сьогодні підходів до класифікації текстів, з урахуванням саме вимог поставленої задачі, дозволив дійти висновку про доцільність застосування для досягнення поставленої мети саме моделей класичного машинного навчання з попередньою обробкою текстів та їх представленням у вигляді числових векторів TF-IDF.

Напрямками подальших досліджень є:

- деталізація процесу класифікації з вибором конкретної ML моделі;
- формування словника термінів та вибір засобів його формалізації для ефективного визначення результуючих категорій.

Література

1. *Дубовик А. В., Волинець Є. А.* Автоматична класифікація текстів. Наукові записки НаУКМА. Комп'ютерні науки. Том 8 (2025). С. 102-107. DOI: 10.18523/2617-3808.2025.8.102-107. – https://nrpcomp.ukma.edu.ua/article/view/344850/332233
2. *Moez All.* Understanding Text Classification in Python. 2022. – https://www.datacamp.com/tutorial/text-classification-python
3. SpamAssassin configuration file. https://spamassassin.apache.org/full/3.4.x/doc/Mail_SpamAssassin_Conf.html
4. *Hutto, C. J., & Gilbert, E.* VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Proceedings of ICWSM. 2014. – https://ojs.aaai.org/index.php/ICWSM/article/view/14550/14399
5. *Esuli, A., & Sebastiani, F.* SentiWordNet: A Publicly Available Lexical Resource for Opinion Mining. Proceedings of LREC 2006. – http://www.lrec-conf.org/proceedings/lrec2006/pdf/384_pdf.pdf
6. Rasa Open Source Documentation. 2025 – https://legacy-docs-oss.rasa.com/docs/rasa/rules/
7. GoogleCloud Guide. – https://docs.cloud.google.com/dialogflow/es/docs/intents-overview
8. *Shuo Xu, Yan Li, Zheng Wang*. Bayesian Multinomial Naïve Bayes Classifier to Text Classification. Institute of Scientific and Technical Information of China. № 15. 2015. – https://www.researchgate.net/publication/317173563_Bayesian_Multinomial_Naive_Bayes_Classifier_to_Text_Classification/link/59fa7e88aca272026f6f98e4/download?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6InB1YmxpY2F0aW9uIiwicGFnZSI6InB1YmxpY2F0aW9uIn19
9. *C. Cortes and V. Vapnik.* Support-vector networks. Machine Learning. November 1995. – P. 273–297
10. *Thorsten Joachims.* Text Categorization with Support Vector Machines: Learning with Many Relevant Features. Support Vector Learning. Conference paper. 2005. – P. 137-142. https://www.cs.cornell.edu/people/tj/publications/joachims_98a.pdf
11. *J. Kivinen, M. Warmuth, and P. Auer.* The perceptron algorithm vs. winnow: Linear vs. logarithmic mistake bounds when few input variables are relevant. In Conference on Computational Learning Theory, 1995.
12. *Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., & Lin, C.-J.* LIBLINEAR: A Library for Large Linear Classification. 2008. – https://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf
13. *Kim, Y.* Convolutional Neural Networks for Sentence Classification. 2014. – https://arxiv.org/pdf/1408.5882

14. *Hochreiter, S., & Schmidhuber, J.* Long Short-Term Memory. Neural Computation. 1997. – https://www.bioinf.jku.at/publications/older/2604.pdf
15. *Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K.* BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2019. – https://arxiv.org/pdf/1810.04805
16. *Liu, Y., et al.* RoBERTa: A Robustly Optimized BERT Pretraining Approach. 2019. – https://arxiv.org/pdf/1907.11692
17. *Conneau, A., et al.* Unsupervised Cross-lingual Representation Learning at Scale. 2020. – https://arxiv.org/pdf/1911.02116
18. *Salton, G., & Buckley, C.* Term-weighting approaches in automatic text retrieval. 1988. – https://dl.acm.org/doi/pdf/10.1145/53990.54006

References

1. *Dubrovik A. V., Volynec J. A.* Automatic text classification. Proceedings of NaUKMA. Computer Science. Volume 8 (2025). P. 102-107. DOI: 10.18523/2617-3808.2025.8.102-107. – https://nrpcomp.ukma.edu.ua/article/view/344850/332233
2. *Moez All.* Understanding Text Classification in Python. 2022. – https://www.datacamp.com/tutorial/text-classification-python
3. SpamAssassin configuration file. https://spamassassin.apache.org/full/3.4.x/doc/Mail_SpamAssassin_Conf.html
4. *Hutto, C. J., & Gilbert, E.* VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Proceedings of ICWSM. 2014. – https://ojs.aaai.org/index.php/ICWSM/article/view/14550/14399
5. *Esuli, A., & Sebastiani, F.* SentiWordNet: A Publicly Available Lexical Resource for Opinion Mining. Proceedings of LREC 2006. – http://www.lrec-conf.org/proceedings/lrec2006/pdf/384_pdf.pdf
6. Rasa Open Source Documentation. 2025 – https://legacy-docs-oss.rasa.com/docs/rasa/rules/
7. GoogleCloud Guide. – https://docs.cloud.google.com/dialogflow/es/docs/intents-overview
8. *Shuo Xu, Yan Li, Zheng Wang*. Bayesian Multinomial Naïve Bayes Classifier to Text Classification. Institute of Scientific and Technical Information of China. № 15. 2015. – https://www.researchgate.net/publication/317173563_Bayesian_Multinomial_Naive_Bayes_Classifier_to_Text_Classification/link/59fa7e88aca272026f6f98e4/download?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6InB1YmxpY2F0aW9uIiwicGFnZSI6InB1YmxpY2F0aW9uIn19
9. *C. Cortes and V. Vapnik.* Support-vector networks. Machine Learning. November 1995. – P. 273–297
10. *Thorsten Joachims.* Text Categorization with Support Vector Machines: Learning with Many Relevant Features. Support Vector Learning. Conference paper. 2005. – P. 137-142. https://www.cs.cornell.edu/people/tj/publications/joachims_98a.pdf
11. *J. Kivinen, M. Warmuth, and P. Auer.* The perceptron algorithm vs. winnow: Linear vs. logarithmic mistake bounds when few input variables are relevant. In Conference on Computational Learning Theory, 1995.
12. *Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., & Lin, C.-J.* LIBLINEAR: A Library for Large Linear Classification. 2008. – https://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf
13. *Kim, Y.* Convolutional Neural Networks for Sentence Classification. 2014. – https://arxiv.org/pdf/1408.5882
14. *Hochreiter, S., & Schmidhuber, J.* Long Short-Term Memory. Neural Computation. 1997. – https://www.bioinf.jku.at/publications/older/2604.pdf
15. *Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K.* BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2019. – https://arxiv.org/pdf/1810.04805
16. *Liu, Y., et al.* RoBERTa: A Robustly Optimized BERT Pretraining Approach. 2019. – https://arxiv.org/pdf/1907.11692
17. *Conneau, A., et al.* Unsupervised Cross-lingual Representation Learning at Scale. 2020. – https://arxiv.org/pdf/1911.02116
18. Salton, G., & Buckley, C. Term-weighting approaches in automatic text retrieval. (1988). https://dl.acm.org/doi/pdf/10.1145/53990.54006

Дата першого надходження до видання: 24.02.2026
Внутрішня рецензія отримана: 03.03.2026
Зовнішня рецензія отримана: 05.03.2026
Дата прийняття статті до друку: 19.03.2026
Дата публікації: 16.04.2026

Про авторів:

[1]Захарова Ольга Вікторівна,
кандидат технічних наук,
старший науковий співробітник
[1] Zakharova Olga,
Ph.D (technical sciences), senior scientist
http://orcid.org/0000-0002-9579-2973.

[2] Спекторовська Лада Олександрівна,
Студент бакалаврата
[2] Spektorovska Lada,
Bachelor student
http://orcid.org/0009-0007-7173-0149

Місце роботи авторів:

[1] Інститут програмних систем
НАН України,
проспект Академіка Глушкова, 40
[1] Institute of Software Systems.
National Academy of Sciences of Ukraine
Тел.: +380(68)5947560
E-mail: ozakharova68@gmail.

[2] Національний технічний університет
«Київський політехнічний інститут
імені Сікорського»
[2] National Technical University of Ukraine
“Igor Sikorsky Kyiv Polytechnic Institute”
Тел.: +380(68)3051221
E-mail: spektorovskalada@gmail.com.

УДК 004.052.32, 004.43 https://doi.org/10.15407/pp2026.01.093

О.В. Ніконов

АНАЛІЗ ЗАСОБІВ АВТОМАТИЗОВАНОГО МАШИННОГО НАВЧАННЯ ДЛЯ ЗАСТОСУВАННЯ У МАРКЕТИНГУ

У статті досліджено проблему автоматизації діяльності IT-експертів з машинного навчання за допомогою сучасних фреймворків AutoML (AutoSklearn та TPOT). Метою роботи є подолання фундаментального протиріччя між високою ресурсомісткістю ручного створення предиктивних конвеєрів та необхідністю будувати точні моделі в умовах обмеженої кількості даних, коли компанії володіють лише базовою транзакційною інформацією. Запропонований підхід формалізує використання алгоритмів AutoML для розв'язання задачі прогнозування відтоку клієнтів у роздрібній торгівлі, замінюючи ручні процеси обробки даних автоматизованими рішеннями. Підхід реалізовано шляхом генерації набору предметно-орієнтованих ознак на базі RFM-моделі та валідовано методами історичної симуляції на транзакційному датасеті «Online Retail». Експериментальні результати демонструють, що системи AutoML здатні ефективно працювати із «сирими» даними: AutoSklearn забезпечує стабільну зважену F1-міру на рівні 0.78 та ROC AUC 0.792 вже за 5 хвилин роботи. Робота має практичне значення для розробки ресурсоефективних предиктивних систем, мінімізації впливу людського фактора та пришвидшення розгортання моделей на підприємствах із базовим рівнем збору даних.
Ключові слова: автоматизоване машинне навчання, предиктивна аналітика, прогнозування відтоку клієнтів, роздрібна торгівля, інженерія ознак.

O.V. Nikonov

ANALYSIS OF AUTOMATED MACHINE LEARNING TOOLS FOR APPLICATION IN MARKETING

The article investigates the problem of automating the activities of IT experts in machine learning using modern AutoML frameworks (AutoSklearn and TPOT). The aim of the work is to overcome the fundamental contradiction between the high resource intensity of manual creation of predictive pipelines and the need to build accurate models in conditions of limited data, when companies have only basic transactional information. The proposed approach formalizes the use of AutoML algorithms to solve the problem of predicting customer churn in retail, replacing manual data processing processes with automated solutions. The approach is implemented by generating a set of subject-oriented features based on the RFM model and validated by historical simulation methods on the transactional dataset "Online Retail". Experimental results demonstrate that AutoML systems are able to work effectively with "raw" data: AutoSklearn provides a stable weighted F1-measure at the level of 0.78 and ROC AUC 0.792 in just 5 minutes of work. The work has practical significance for developing resource-efficient predictive systems, minimizing the impact of the human factor, and accelerating the deployment of models at enterprises with a basic level of data collection.
Keywords: automated machine learning, predictive analytics, customer churn prediction, retail, feature engineering.

Вступ

Сучасний конвеєр машинного навчання (ML pipeline) є доволі складним багатоетапним IT-процесом. Традиційне створення предиктивних моделей вимагає глибокої експертизи в програмуванні та статистиці для відбору ознак, вибору алгоритмів і тонкого налаштування гіперпараметрів [1, 2]. На практиці IT-фахівці та інженери з даних витрачають від 60% до 80% свого робочого часу саме на рутинні завдання з підготовки масивів інформації, що робить розробку тривалою та дорогою [3, 4]. Вирішенням цієї технологічної проблеми є впровадження засобів автоматизованого машинного навчання (AutoML). Вони зводять до мінімуму ручне втручання, самостійно тестуючи методи очищення даних та вибираючи оптимальні архітектури моделей (наприклад, ансамблі дерев рішень чи нейронні мережі). Це дозволяє суттєво скоротити час розробки та делегувати створення систем експертам без глибоких навичок кодування [5, 6, 7].

Завдяки здатності ефективно обробляти гігантські масиви даних та автоматизувати процеси, ці IT-рішення активно впроваджуються у маркетинг для переходу до гіперперсоналізації та підвищення рентабельності інвестицій [8, 9, 10, 11]. Критичним стратегічним викликом у цій сфері є проблема відтоку клієнтів. Залучення нових споживачів коштує компаніям у 5-7 разів дорожче, ніж утримання існуючих, а глобальні показники відтоку у висококонкурентних галузях сягають 15-25% щорічно [12, 13]. Зниження рівня відтоку лише на 5% здатне підвищити прибутковість бізнесу на 25-125%, тоді як загальні втрати від переходу клієнтів до конкурентів вимірюються трильйонами доларів [12].

Попри загальну ефективність AutoML, стандартні універсальні системи часто зосереджуються на базовій обробці табличних даних і не здатні самостійно конструювати складні предметно-орієнтовані ознаки [1, 3, 14]. З огляду на це, ідеальним і надзвичайно складним середовищем для перевірки таких IT-рішень є прогнозування відтоку клієнтів у роздрібній торгівлі [15]. Використання спеціалізованих платформ AutoML дозволяє автоматично генерувати специфічні маркетингові метрики: показники поведінки RFM (давність, частота, сума покупок), індикатори довічної цінності (CLV) та оцінки поведінкової залученості [1, 12, 16]. Завдяки здатності алгоритмів знаходити приховані нелінійні зв'язки, компанії можуть виявляти вразливих клієнтів із безпрецедентною точністю, формувати списки ризику для цільових кампаній і ефективно замінювати ручну попередню обробку даних надійною автоматизацією.

1. Аналіз існуючих проблем

Хоча дослідження у сфері автоматизованого машинного навчання (AutoML) та прогнозування відтоку клієнтів демонструють значний науковий прогрес, існує низка суттєвих недоліків та розривів між теорією і практикою, які обмежують їхнє застосування. Однією з проблем є те, що багато досліджень не розголошують дані, на яких тренувалися їхні моделі, оскільки вони переважно є приватними, що робить практично неможливим відтворення результатів та їхнє об'єктивне порівняння між різними науковими роботами. Це помітно в [1, 2, 13], про дану проблему також зазначено в [16].

Ті ж набори даних, які знаходяться у відкритому доступі, зазвичай надаються вже у попередньо обробленому або агрегованому вигляді. Це створює додаткові перешкоди, оскільки не дозволяє дослідникам виводити специфічні та нові ознаки безпосередньо на основі необробленої інформації [16]. Крім того, такий формат даних викликає обґрунтовані питання до коректності методології тестування, оскільки багато моделей оцінюються за допомогою простого випадкового розділення даних (random split), що є некоректним для ринків, які постійно змінюються. Замість цього кращою практикою є тестування на даних поза вибіркою з урахуванням часу (out-of-sample testing або історична симуляція), яке краще відображає реальні умови прогнозування, але рідше застосовується через попередню агрегацію [15].

Іншим вагомим недоліком є галузевий дисбаланс у фокусі наукових робіт. Більшість досліджень AutoML у сфері прогнозування відтоку клієнтів віддає значну перевагу таким індустріям, як фінанси (банкінг) [1, 3, 12], телекомунікації [2, 3, 14] та сектор онлайн-ігор [16]. Водночас набагато менше уваги приділяється сектору роздрібної торгівлі (retail), детально розглянутий лише в [15] та поверхнево у [3], де відносини з клієнтами часто є неконтрактними, що робить завдання прогнозування відтоку та життєвої цінності клієнта ще складнішим і може вимагати специфічних підходів.

Остання вагома проблема пов'язана із розбіжністю між обсягом даних, які використовуються в дослідницьких експериментах, та тими, якими реально володіє бізнес. У наукових роботах тестування моделей часто відбувається на дуже багатих та різноманітних категоріях даних, що включають демографічні показники, детальну інформацію про сесії, графи соціальних зв'язків та історію звернень до служби підтримки [16]. Проте на практиці багато компаній, зокрема, в роздрібній торгівлі, можуть мати лише дуже обмежений набір даних, який зводиться виключно до базової інформації про транзакції (сума, дата, кількість товарів та ідентифікатор клієнта), від якого і потрібно відштовхуватися при побудові прогностичних систем [15]. Це створює значний розрив між теоретичними досягненнями багатокритеріальних моделей AutoML та їхньою практичною цінністю для компаній із базовим рівнем збору даних.

Зважаючи на вищезазначені прогалини, виникає гостра необхідність у дослідженні, яке б практично оцінило здатність сучасних систем AutoML автоматизувати роботу ML-інженерів в умовах обмежених даних у роздрібній торгівлі. Тому метою даної статті є оцінка ефективності передових фреймворків автоматизованого машинного навчання (на прикладі AutoSklearn та TPOT) у вирішенні задачі прогнозування відтоку клієнтів. Для досягнення цієї мети буде виконано наступні кроки: підготовку відкритого транзакційного датасету, генерацію предметно-орієнтованих маркетингових ознак, тестування моделей AutoML на різних часових лімітах із використанням методу історичної симуляції (out-of-sample testing), а також порівняння результатів роботи автоматизованих конвеєрів із ручною попередньою обробкою.

2. Підготовка даних

У даному дослідженні використовується загальнодоступний транзакційний набір даних «Online Retail» (зокрема версія з репозиторію машинного навчання UCI [17]), який охоплює історію покупок клієнтів британського інтернет-магазину унікальних подарунків за дворічний період з 2009 по 2011 рік. Початковий масив даних налічує 1 067 371 рядок та 8 колонок із такою базовою інформацією: номер рахунку, унікальний код і назва товару, кількість куплених одиниць, дата та час транзакції, ціна товару у фунтах стерлінгів, унікальний ідентифікатор (ID) клієнта та країна транзакції.

Для підвищення точності моделювання масив даних пройшов комплексну попередню обробку. Процес очищення розпочався з видалення дублікатів та транзакцій без унікального ідентифікатора клієнта, що уможливило коректну агрегацію індивідуальної історії покупок. Згодом було вилучено скасовані замовлення (із префіксом «C») та нетипові технічні товари (наприклад, «ADJUST» чи «BANK CHARGES»), аби запобігти викривленню розрахунків доходу й поведінкових метрик. Вибірку географічно звузили до транзакцій із Великої Британії (понад 90% датасету), оскільки використання вітчизняних даних наразі неактуальне через суттєву зміну патернів споживання внаслідок війни. На фінальному етапі для стабілізації роботи алгоритмів було відсіяно статистичні викиди вище 99-го перцентиля, а всі валідні транзакції згруповано по днях, що дозволило сформувати чітку структуру щоденної активності кожного покупця.

Навіть за умови використання засобів AutoML, суттєво важливою залишається обробка та агрегація даних перед їх подачею в модель. Автоматизація не замінює необхідності у технічних навичках для

очищення масиву від дублікатів та аномалій, а також у знаннях предметної області для відсіювання нерелевантних записів чи врахування змін у поведінці. Якісний результат моделювання безпосередньо залежить від експертної підготовки вхідної вибірки, яка є фундаментом для будь-якого алгоритму.

3. Визначення відтоку

З технічної точки зору прогнозування відтоку клієнтів найчастіше формулюється як класична задача бінарної класифікації, де алгоритм машинного навчання має передбачити один із двох можливих станів: клієнт продовжить взаємодію з компанією або ж припинить її [3, 15]. Для коректної побудови та тренування такої прогностичної моделі життєвий цикл клієнта штучно розділяють на специфічні часові проміжки. Спочатку визначається вікно спостереження (observation window), протягом якого система агрегує історичні дані про транзакції, поведінку та будь-які взаємодії споживача з брендом для формування вхідних ознак [3, 12, 15]. За цим періодом слідує вікно прогнозування, яке також називають вікном маркування (labeling window або evaluation window). Саме в цьому вікні фіксується фактичний цільовий статус клієнта, а саме чи відбувся відтік. Часове та логічне розділення цих вікон є важливим для уникнення витоку даних, коли інформація з майбутнього могла б випадково потрапити у тренувальний набір і спотворити результати моделювання. Тривалість вікна маркування зазвичай залежить від специфіки конкретного бізнесу.

Саме поняття «відтоку клієнтів» є доволі розмитим і суб'єктивним, адже його визначення здатне суттєво відрізнятися залежно від типу послуг, галузі або стратегічних потреб конкретної компанії [12, 16]. У науковій літературі та комерційній практиці не існує єдиного універсального стандарту того, що саме вважати втратою клієнта. Наприклад, у банкінгу відтоком вважають зниження транзакцій на 30% або статус «неактивності» [16], тоді як в інших сферах критерієм є відсутність дій протягом 90 днів чи медіанного інтервалу між покупками [15]. Отже, формулювання задачі прогнозування майже завжди базується на евристичних правилах, які встановлюються експертами галузі або керівництвом для вирішення дуже специфічних поточних проблем бізнесу.

Через різноманітність цих визначень та характеристик різних ринків, нормальні показники відтоку суттєво коливаються. Наприклад, щорічно телекомунікаційний сектор втрачає 20–40% абонентів [2], банківський — 15–25% [8], а за іншими даними ці рівні становлять 20,4% та 26,5% відповідно [3]. Окремої уваги заслуговує сфера електронної та роздрібної торгівлі, де відносини між магазином та покупцем переважно не мають жодного контрактного характеру. У такому середовищі споживачі можуть дуже легко змінювати платформи та бренди, тому рівень відтоку тут традиційно є найвищим. Зокрема, дослідження на базі даних онлайн-магазину роздрібної торгівлі фіксують частку відтоку на рівні 32,1% [3].

Спираючись на проаналізовану інформацію, для подальшого дослідження було вирішено сфокусуватися лише на активних клієнтах, які зробили не менше 4 покупок упродовж усього часового періоду. Розмір оглядового вікна встановлений на рівні 12 місяців у зв’язку з тим, що специфіка даного датасету не передбачає дуже частих покупок (середня кількість днів між покупками становить 112 днів, а медіана – 82 дні). Вікно прогнозування визначене на рівні 6 місяців. Це дало можливість отримати датасет, у якому 22.4% клієнтів зазнали відтоку, що відповідає очікуванням з огляду на дані попередніх досліджень.

4. Підготовка ознак

Побудова власної прогностичної моделі відтоку клієнтів спирається на транзакційні дані для генерації набору поведінкових метрик, концептуально подібних до тих, що були розроблені у [15], головною відмінністю між якими виступає розмір оглядового вікна (1 рік замість 2 місяців). Основу набору ознак складають базові змінні: загальна кількість транзакцій, сумарні витрати клієнта, середній розмір чека за обраний період спостереження (який може

гнучко налаштовуватися залежно від бізнес-циклу), а також кількість днів, що минула з моменту останньої покупки. Цей вибір є обґрунтованим, оскільки такі метрики фактично є адаптацією класичної моделі RFM (давність, частота, грошова цінність), яка, згідно з численними дослідженнями, слугує галузевим стандартом для моделювання та прогнозування поведінки споживачів у багатьох індустріях [16]. Окрім цього, також включено метрику загальної тривалості активності клієнта, що допоможе алгоритмам оцінювати його загальну лояльність. Важливість цього кроку підтверджується ефективністю розширених моделей LRFM (до яких додається тривалість відносин, Length), які активно застосовуються в інших наукових працях для суттєвого покращення розпізнавання стабільних користувачів [16].

Крім статичних показників, важливим етапом є генерація розширених динамічних ознак, які фіксують відсоткові зміни в купівельній активності. Наслідуючи логіку [15], створено змінні для вимірювання відносного зростання або падіння кількості покупок та сум витрат порівняно з попередніми періодами, адже саме спад активності є найяскравішим раннім індикатором майбутнього відтоку. Прогностична цінність таких трендових індикаторів надійно підкріплена й іншими сучасними роботами. Наприклад, спеціалізовані системи автоматизованого машинного навчання, такі як Marketing-AutoM3L, за замовчуванням генерують подібні оцінки поведінкової залученості, обчислюючи швидкість і відносні тренди активності клієнтів для завчасного виявлення ризиків [3]. Такий же принцип доводить свою дієвість і у дослідженнях фінансового сектору, де аналіз спадних чи зростаючих трендів (нахилів) транзакційної активності користувачів дозволяв алгоритмам значно ефективніше та точніше прогнозувати відтік, ніж використання лише статичної демографічної інформації [12].

5. Масштабування й вибір ознак

Саме на цьому етапі порівнюється ефективність засобів автоматизованого машинного навчання із ручним підходом. За основу ручного підходу взято дослідження [15], де для боротьби зі зміщеним розподілом значень, який часто виникає через екстремальні покупки клієнтів, застосовується логарифмічна трансформація. Після цього відбувається масштабування ознак за допомогою стандартизації Z-score, що приводить всі показники до єдиного діапазону. Щоб усунути проблему високої кореляції між згенерованими метриками, використовується метод ручного групування: виявляються ознаки із високою кореляцією, після чого їхні значення усереднюються, об'єднуються в єдиний показник. Це дозволяє зменшити розмірність даних і значно спростити роботу прогностичних алгоритмів.

На противагу ручному процесу, передові системи автоматизованого машинного навчання, такі як AutoSklearn, пропонують широкий спектр вбудованих математичних методів для попередньої обробки. AutoSklearn повністю автоматизує масштабування, самостійно тестуючи та обираючи найоптимальніший метод серед стандартного, мінімаксного або робастного масштабування для конкретного набору даних. Щодо обробки сильно корельованих ознак, AutoSklearn пропонує прямий алгоритмічний аналог - метод агломерації ознак [18]. Цей алгоритм автоматично виконує кластеризацію змінних і об'єднує схожі ознаки, що по суті автоматизує процес ручного усереднення корельованих метрик. Крім того, AutoSklearn використовує потужні методи зменшення розмірності, такі як метод головних компонент (PCA) та швидкий аналіз незалежних компонент (Fast ICA), що дозволяє ефективно усувати надлишковість та шум без жодного ручного втручання експерта [18].

Інша потужна система, TPOT, підходить до масштабування та відбору ознак через призму генетичного програмування, будуючи гнучкі еволюційні конвеєри перетворень. Для масштабування TPOT автоматично інтегрує стандартні та робастні нормалізатори (StandardScaler, RobustScaler) безпосередньо у свої пайплайни, якщо еволюційний алгоритм підтверджує, що це підвищує загальну точність моделі [19]. Однак, на відміну від прямого групування та усереднення корельованих ознак, TPOT за-

стосовує рандомізований метод головних компонент (RandomizedPCA), рекурсивне виключення ознак (RFE) та фільтрацію за порогом дисперсії (Variance Threshold) [19]. Це означає, що замість злиття схожих метрик в одну, TPOT комбінує їх у нові нелінійні компоненти (головні компоненти) або відкидає ті з них, які еволюційний алгоритм визнає найменш корисними чи надто корельованими для кінцевого прогнозу.

6. Перебіг експерименту та його результати

У ході дослідження відбувалося тренування багатьох моделей машинного навчання, для яких було використано різні методи підготовки ознак та різні засоби AutoML для вибору й оптимізації моделей. Оскільки процес автоматизованого машинного навчання ресурсомісткий, для AutoSklearn і TPOT було надано різні проміжки часу з метою перевірити як часові рамки впливають на навчання моделей AutoML.

Експеримент полягає у проведенні порівняльного тестування AutoSklearn та TPOT у задачі прогнозування відтоку клієнтів. Оскільки процес автоматизованого машинного навчання ресурсомісткий, тренування було розподілене за різними часовими лімітами (5, 15 та 30 хвилин) і також за трьома підходами до підготовки даних: повністю ручне масштабування та групування ознак, лише ручне масштабування та використання повністю «сирих» маркетингових ознак.

Під час розробки та тренування моделей машинного навчання замість стандартного випадкового перемішування даних було застосовано підхід історичної симуляції, відомий як out-of-sample historical testing [15]. Цей метод передбачає навчання алгоритму на вибірці історичних даних з минулого та подальше тестування його прогнозів на абсолютно нових даних із майбутнього періоду. Важливість такого підходу полягає в тому, що реальні ринки та поведінка споживачів постійно змінюються у часі. Прогностичні моделі поводяться зовсім інакше в умовах простого випадкового розділення даних порівняно з реальним прогнозуванням майбутніх подій. Завдяки історичній симуляції поза вибіркою можна отримати максимально реалістичну оцінку того, як саме модель працювала б, якби її запустили в комерційну експлуатацію в реальному часі.

Для забезпечення додаткової надійності та стабільності алгоритмів на етапі навчання також застосовано метод п'ятиблочної перехресної перевірки, тобто 5-fold cross-validation, подібно до [1]. Цей підхід розділяє тренувальний масив даних на п'ять рівних частин, де кожна з них по черзі виступає в ролі валідаційного набору, тоді як інші чотири використовуються для навчання моделі. Використання перехресної перевірки є важливим, оскільки ефективно допомагає пом'якшити проблему перенавчання (overfitting) [16]. Оцінюючи модель на різних незалежних підмножинах даних, алгоритм узагальнює знайдені закономірності, а не просто заучує специфіку тренувальної вибірки.

Для оцінки прогнозів обрано комплекс метрик, що враховують значний класовий дисбаланс (22.4% відтоку). Оскільки стандартна точність (accuracy) у таких умовах може бути оманливою [16], основна увага приділена ROC AUC. Вона об'єктивно оцінює здатність моделі розрізняти класи незалежно від порогу відсічення [15]. Додатково використано weighted F1 [14] — гармонійне середнє між точністю та повнотою, зважене за кількістю представників кожного класу. Для оцінки бізнес-ефективності застосовано top-decile lift. Цей показник демонструє перевагу моделі над випадковим вгадуванням серед 10% найбільш ризикових клієнтів, що дозволяє оптимізувати витрати на їхнє утримання [15, 16].

Отримані результати, наведені в таблиці, демонструють різну поведінку двох фреймворків за умов обмеженого часу. AutoSklearn показує високу стабільність і точність прогнозів усіх часових проміжків (Weighted F1 на рівні 0.75–0.78, ROC AUC 0.783–0.792) незалежно від того, як були підготовлені дані. Така ефективність навіть за встановлених часових обмежень може бути зумовлена його архітектурою: система використовує байєсівську оптимізацію на базі випадкових лісів (SMAC),

яка суттєво пришвидшується завдяки метанавчанню. Фреймворк аналізує метаознаки поточного набору даних, використовує досвід попередніх успішних оптимізацій для «теплого старту», а наприкінці автоматично поєднує найкращі знайдені моделі в ефективний ансамбль.

На противагу цьому, продуктивність TPOT суттєво залежить від наданого йому обчислювального часу. За умови ліміту в 5 хвилин на «сирих» даних його результати є менш показовими (Accuracy лише 0.647, Weighted F1 падає до 0.66), проте зі збільшенням часу до 15 та 30 хвилин він різко стабілізує роботу, досягаючи рівня AutoSklearn, іноді й перевершуючи показники з невеликим відривом (Accuracy зростає до 0.800, а Weighted F1 до 0.78). Ця динаміка відповідає закладеним у TPOT алгоритмам генетичного програмування, де система ітеративно будує, мутує та схрещує деревоподібні конвеєри перетворень. Мутація пайплайнів із подальшою багатоцільовою Парето-оптимізацією є вкрай ресурсомістким процесом, що вимагає значного часу для еволюції моделі та збіжності до оптимального рішення. За обмеженого часу ручне масштабування дещо допомогло TPOT знайти рішення швидше (Accuracy 0.710 на 5 хвилинах), але не розкрило його повного потенціалу.

Ще одним показовим висновком з експерименту є вплив ручної попередньої обробки порівняно з автоматизованою. Подача «сирих» (немасштабованих і незгрупованих) даних може несуттєво погіршити деякі результати (падіння ROC AUC з 0.799 до 0.792), а за достатнього часу тренування дозволила обом фреймворкам досягти найвищих показників Weighted F1 (0.78) у порівнянні з ручною обробкою (0.71–0.75). Це підтверджує тезу про те, що сучасні засоби AutoML здатні на рівні з людиною автоматизувати процеси нормалізації та боротьби з мультиколінеарністю. AutoSklearn самостійно тестує різні методи масштабування та використовує алгоритм агломерації ознак або аналіз головних компонент (PCA) для усунення надлишковості та шуму. Своєю чергою, TPOT інтегрує нормалізатори безпосередньо у свої еволюційні пайплайни, а замість ручного злиття схожих метрик комбінує їх у нові нелінійні компоненти або алгоритмічно відкидає через ре-

AutoML tool	AutoSklearn				TPOT			
Testing on features scaled and grouped manually	Accuracy	ROC AUC	Weighted F1	Top-Decile Lift	Accuracy	ROC AUC	Weighted F1	Top-Decile Lift
5m	0.785	0.783	0.75	2.390	0.710	0.736	0.73	2.148
15m	0.791	0.794	0.75	2.685	0.786	0.787	0.74	2.497
30m	0.786	0.794	0.74	2.470	0.785	0.799	0.74	2.551
Testing on features scaled manually	Accuracy	ROC AUC	Weighted F1	Top-Decile Lift	Accuracy	ROC AUC	Weighted F1	Top-Decile Lift
5m	0.789	0.790	0.75	2.578	0.711	0.785	0.73	2.470
15m	0.784	0.796	0.72	2.578	0.726	0.789	0.75	2.524
30m	0.779	0.784	0.69	2.685	0.781	0.792	0.71	2.470
Testing on raw features	Accuracy	ROC AUC	Weighted F1	Top-Decile Lift	Accuracy	ROC AUC	Weighted F1	Top-Decile Lift
5m	0.798	0.792	0.78	2.658	0.647	0.745	0.66	2.739
15m	0.787	0.778	0.78	2.658	0.793	0.791	0.78	2.524
30m	0.787	0.769	0.78	2.685	0.800	0.787	0.78	2.497

курсивне виключення ознак (RFE). Таким чином, ручне логарифмування, стандартизація Z-score та логічне усереднення залежних змінних виявляються зайвими та певною мірою обмежувальними кроками.

У контексті бізнес-аналітики, оцінювати ці моделі варто через призму профільних метрик. Через значний дисбаланс класів базова точність (Accuracy) може бути зміщеною та оманливою. Значно важливішими є стабільні показники ROC AUC та weighted F1, які успішно тримаються на рівні близько 0.78–0.79. Водночас метрика Top-Decile Lift, яка досягає значень 2.390–2.739, оозначає, що навчені моделі AutoML дозволяють виявляти клієнтів із найвищим ризиком відтоку (у верхніх 10% клієнтської бази) у приблизно два з половиною рази ефективніше, ніж якби маркетологи обирали аудиторію навмання, що демонструє суттєву комерційну цінність побудованих конвеєрів. Завдяки цьому бізнес може максимально раціонально оптимізувати маркетинговий бюджет, спрямовуючи пропозиції з утримання виключно на тих споживачів, які цього справді потребують.

Результати підтверджують, що для швидкого отримання надійних прогнозів на непідготовлених даних добре підходить AutoSklearn, тоді як TPOT потребує значних обчислювальних витрат і часу. Однак важливо розуміти, що високі результати обох систем стали можливими також завдяки тому, що до їхнього запуску було проведено якісну інженерію доменних ознак: вхідні дані вже містили розраховані маркетингові RFM-метрики, тривалість активності рахунків та індикатори поведінкової залученості клієнтів. Засоби AutoML беруть на себе математичну рутину обробки та відбору ознак, але формування правильного маркетингового контексту та створення релевантних змінних все ще лежить в основі успішного вирішення проблеми.

Висновки

У статті запропоновано та практично оцінено ефективність впровадження засобів автоматизованого машинного навчання (AutoML) для мінімізації ручного втручання технічних спеціалістів у процес побудови предиктивних моделей на базі обмежених наборів даних. Проведені експерименти підтвердили основну гіпотезу дослідження, згідно з якою сучасні платформи (AutoSklearn та TPOT) здатні успішно та на рівні з людиною автоматизувати процеси нормалізації, масштабування та боротьби з мультиколінеарністю без втрати якості кінцевого прогнозу. У межах роботи формалізовано задачу прогнозування відтоку клієнтів у неконтрактному середовищі роздрібної торгівлі, де бізнес оперує виключно базовою транзакційною історією. Валідація алгоритмів здійснювалася методами історичної симуляції (out-of-sample testing) та 5-блочної перехресної перевірки з використанням 11 розроблених поведінкових метрик.

Експериментальна оцінка продемонструвала високу ефективність запропонованого підходу: подача «сирих» (немасштабованих і незгрупованих) даних у системи AutoML дозволила досягти найвищих показників Weighted F1 (0.78) порівняно з їхньою попередньою ручною обробкою (0.71–0.75). Аналіз часових лімітів виявив, що AutoSklearn здатен генерувати стабільні конвеєри з ROC AUC на рівні 0.792 вже за 5 хвилин обчислень завдяки механізмам метанавчання, тоді як архітектура TPOT потребувала від 15 до 30 хвилин для збіжності до аналогічних результатів. Побудовані моделі забезпечили показник Top-Decile Lift до 2.739, що дозволяє виявляти ризикових клієнтів у 2,5 раза ефективніше за випадковий вибір. Отримані результати підтверджують доцільність використання фреймворків AutoML для розгортання прогностичних IT-систем в умовах дефіциту розширених даних та нестачі часу технічних експертів, водночас визначаючи ключову роль якісної генерації доменних ознак перед етапом автоматизації.

References

1. S. Dao, T. Dong, S. Chen, Automated Customer Churn Prediction in Banking: A Domain-Aware AutoML Approach (2026).
2. M. Mandić, G. Kraljević, Churn prediction model improvement using automated machine learning with social network parameters,

Revue d'Intelligence Artificielle 36 (3) (2022) 373–379. doi: 10.18280/ria.360304
3. Y. Tian, W. Shao, Z. Deng, Marketing-AutoM3L: domain-aware automated machine learning for financial customer analytics, Frontiers in Artificial Intelligence 9 (2026) 1726900. doi: 10.3389/frai.2026.1726900
4. M.-A. Zöller, M.F. Huber, Benchmark and Survey of Automated Machine Learning Frameworks, Journal of Artificial Intelligence Research 70 (2019) 409–472.
5. D. Luo, C. Feng, Y. Nong, Y. Shen, AutoM3L: An Automated Multimodal Machine Learning Framework with Large Language Models, in: Proceedings of the 32nd ACM International Conference on Multimedia (2024).
6. P. Trirat, W. Jeong, S.J. Hwang, AutoML-Agent: A Multi-Agent LLM Framework for Full-Pipeline AutoML, arXiv preprint arXiv:2410.02958 (2024).
7. X. He, K. Zhao, X. Chu, AutoML: A Survey of the State-of-the-Art, arXiv preprint arXiv:1908.00709 (2019).
8. A. Mari, The Rise of Machine Learning in Marketing: Goal, Process, and Benefit of AI-Driven Marketing (2019).
9. D. Herhausen, S.F. Bernritter, E.W.T. Ngai, A. Kumar, D. Delen, Machine learning in marketing: Recent progress and future research directions, Journal of Business Research 170 (2024) 114254. doi: 10.1016/j.jbusres.2023.114254
10. M.S. Kasem, M. Hamada, I. Taj-Eddin, Customer profiling, segmentation, and sales prediction using AI in direct marketing, Neural Computing and Applications 36 (2024) 4995–5005. doi: 10.1007/s00521-023-09339-6
11. B. Gao, Y. Wang, H. Xie, Y. Hu, Y. Hu, Artificial Intelligence in Advertising: Advancements, Challenges, and Ethical Considerations in Targeting, Personalization, Content Creation, and Ad Optimization, Sage Open 13 (4) (2023).
12. E. Kaya, X. Dong, Y. Suhara, et al., Behavioral attributes and financial churn prediction, EPJ Data Science 7 (2018) 41. doi: 10.1140/epjds/s13688-018-0165-5
13. M. Mandić, G. Kraljević, Two-Layer Architecture of Telco Churn Auto-ML (2020).
14. M.V.C. Aragão, A.G. Afonso, R.C. Ferraz, et al., A practical evaluation of AutoML tools for binary, multiclass, and multilabel classification, Scientific Reports 15 (2025) 17682. doi: 10.1038/s41598-025-02149-x
15. S. Akhmetbek, Forecasting Customer Future Behavior in Retail Business Using Machine Learning Models, Scientific Journal of Astana IT University (2022).
16. A. Manzoor, M.A. Qureshi, E. Kidney, L. Luca, A Review on Machine Learning Methods for Customer Churn Prediction and Recommendations for Business Practitioners, IEEE Access 12 (2024) 70434–70463.
17. D. Chen, Online Retail II (Version 1) [Data set], UCI Machine Learning Repository (2019).
18. M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, F. Hutter, Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning, arXiv preprint arXiv:2007.04074 (2020).
19. P. Ribeiro, et al., TPOT2: A New Graph-Based Implementation of the Tree-Based Pipeline Optimization Tool for Automated Machine Learning, in: S. Winkler et al. (Eds.), Genetic Programming Theory and Practice XX, Springer, Singapore (2024). doi: 10.1007/978-981-99-8413-8_1

Дата першого надходження до видання: 07.03.2026
Внутрішня рецензія отримана: 14.03.2026
Зовнішня рецензія отримана: 14.03.2026
Дата прийняття статті до друку: 19.03.2026
Дата публікації: 16.04.2026

Про автора:

[1]Ніконов Олександр Володимирович,
аспірант
Nikonov Olexandr,
Post-graduate student
https://orcid.org/0009-0009-4743-4854.

Місце роботи автора:

[1] Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
E-mail: nikonov.sanynikonov@gmail.com
Сайт: https://ist.kpi.ua

УДК 519.61:004.272 https://doi.org/10.15407/pp2026.01.102

Г. І. Малашонок, С. С. Сухарський

БЛОКОВО-РЕКУРСИВНИЙ ПІДХІД ДО УНІТАРНОГО РОЗКЛАДУ МАТРИЦЬ

Сингулярне розкладання матриці є одним з основних інструментів чисельної лінійної алгебри та широко застосовується у задачах обробки даних, оптимізації, комп'ютерного зору та машинного навчання. Класичний підхід до його обчислення базується на алгоритмі Хаусхолдера, який дозволяє звести довільну матрицю до тридіагональної або бідіагональної форми з подальшим обчисленням сингулярних значень. Однак зі зростанням розмірності матриць виникають суттєві труднощі з ефективним розпаралелюванням таких алгоритмів, що обмежує їхню продуктивність на сучасних високопродуктивних обчислювальних системах.

Одним із перспективних підходів до подолання цих обмежень є використання блоково-рекурсивних алгоритмів. Такі алгоритми дозволяють розділяти початкову задачу на незалежні блокові підзадачі, що можуть виконуватися паралельно, забезпечуючи високий ступінь масштабованості на багатопроцесорних системах та кластерах із розподіленою пам'яттю.

У даній роботі запропоновано новий алгоритм ортогонального (унітарного) блоково-рекурсивного розкладу симетричних матриць, який базується на використанні QR- та QP-розкладів. Алгоритм дозволяє побудувати ортогональні (унітарні) множники та отримати стрічкову матрицю, що зменшує обчислювальну складність подальших етапів спектрального розкладу.

Отримано оцінку обчислювальної складності алгоритму та показано, що її асимптотичний порядок визначається складністю множення матриць. Експериментальні дослідження продуктивності проведено у середовищі паралельних обчислень із використанням прискорення матричних операцій на графічному процесорі. Отримані результати демонструють ефективність запропонованого алгоритму при зростанні розміру задачі та підтверджують узгодженість експериментальної складності з теоретичними оцінками. Значення похибки реконструкції матриці знаходяться на рівні машинної точності для чисел подвійної точності, що свідчить про чисельну стабільність запропонованого підходу.

Ключові слова: унітарний розклад матриці, блоково-рекурсивний алгоритм, паралельні обчислення, графічний процесор

G. I. Malaschonok, S. S. Sukharskyi

A BLOCK-RECURSIVE APPROACH TO UNITARY MATRIX DECOMPOSITION

Singular value decomposition is one of the fundamental tools of numerical linear algebra and is widely used in data processing, optimization, computer vision, and machine learning. The classical approach to its computation is based on the Householder algorithm, which reduces an arbitrary matrix to a tridiagonal or bidiagonal form followed by the computation of singular values. However, as the matrix size increases, significant difficulties arise in efficiently parallelizing such algorithms, which limits their performance on modern high-performance computing systems.

One promising approach to overcoming these limitations is the use of block-recursive algorithms. Such algorithms allow the original problem to be decomposed into independent block subproblems that can be processed in parallel, providing a high degree of scalability on multiprocessor systems and distributed-memory clusters.

This paper proposes a new block-recursive algorithm for the orthogonal (or unitary) decomposition of symmetric matrices based on QR and QP decompositions. The algorithm constructs orthogonal (unitary) factors and produces a band matrix, reducing the computational complexity of subsequent stages of spectral decomposition.

A complexity analysis is presented showing that the asymptotic order of the algorithm is determined by the complexity of matrix multiplication.Experimental performance studies were carried out in a parallel computing environment using GPU acceleration for matrix operations. The obtained results demonstrate the efficiency of the proposed algorithm as the problem size increases and confirm the agreement between the experimental complexity and the theoretical estimates. The reconstruction error of the matrix remains at the level of machine precision for double-precision floating-point numbers, which indicates the numerical stability of the proposed approach.

Keywords: unitary matrix decomposition, block-recursive algorithm, parallel computing, graphics processing unit

Вступ

Задача ортогонального (унітарного) розкладу симетричної матриці до тридіагональної або стрічкової форми є однією з базових задач чисельної лінійної алгебри. Вона виникає у багатьох прикладних задачах, зокрема, при знаходженні власних значень матриць, розв'язуванні задач спектрального аналізу, моделюванні фізичних процесів та обробці великих масивів даних. У більшості сучасних алгоритмів обчислення власних значень попереднє перетворення матриці до тридіагонального або стрічкового вигляду є необхідним етапом, що значною мірою визначає загальну обчислювальну складність задачі. Зі зростанням розмірів задач та розвитком високопродуктивних обчислювальних систем особливого значення набуває розробка алгоритмів, які дозволяють ефективно використовувати паралельні архітектури. Традиційні методи розкладу матриць, хоча й забезпечують високу чисельну стабільність, часто мають обмежені можливості для паралелізації. У зв'язку з цим актуальною є задача розробки нових алгоритмічних підходів, що поєднують чисельну стійкість класичних методів із високою ефективністю паралельного виконання. Одним із перспективних напрямів розв'язання цієї задачі є використання блоково-рекурсивних алгоритмів, які дозволяють природно організувати паралельні обчислення та ефективно використовувати сучасні високопродуктивні обчислювальні системи.

Задача тридіагоналізації матриці тісно пов'язана з класичними методами QR-розкладу, які широко застосовуються у чисельній лінійній алгебрі завдяки своїй чисельній стійкості. Основні підходи до побудови QR-розкладу — обертання Гівенса, ортогоналізація Грама–Шмідта та відбиття Хаусхолдера — детально описані у класичних роботах Голуба і Ван Лоана [1], а також Трефезена і Бау [2]. Одним із важливих напрямів розвитку алгоритмів чисельної лінійної алгебри стало використання методів швидкого множення матриць. У роботі Штрассена показано можливість зниження складності множення матриць до $\boldsymbol{O}\left(\boldsymbol{n}^{\log_2 7}\right)$ [3], а подальші дослідження Копперсміта і Віноградa дозволили зменшити асимптотичну оцінку до $\boldsymbol{O}(\boldsymbol{n}^{2.38})$ [4]. Ці результати створили передумови для розробки ефективних блоково-рекурсивних алгоритмів у задачах чисельної лінійної алгебри.

Однією з ранніх робіт у цьому напрямі є дослідження Шонхаге [5], у якому розглянуто швидке виконання унітарних перетворень великих матриць. У роботі показано, що такі перетворення можуть бути організовані таким чином, що їхня асимптотична складність має той самий порядок, що і складність множення матриць. Подальший розвиток цієї ідеї привів до створення рекурсивних алгоритмів факторизації матриць. Зокрема, у роботах [6, 7, 8] розглядалось застосування блоково-рекурсивних алгоритмів до задач унітарного розкладу матриць, пов'язаних із задачею QR-розкладу.

Важливим результатом цього напрямку стала робота Деммела та співавторів [9], у якій доведено, що швидкі алгоритми множення матриць можуть бути використані для розв'язання стандартних задач чисельної лінійної алгебри (SVD, LU, QR) із тією самою асимптотичною складністю, зберігаючи водночас чисельну стійкість. Автори показали, що навіть у разі використання так званих слабо стійких алгоритмів множення матриць забезпечується зворотна стійкість відповідних матричних розкладів.

Подальші дослідження показали, що рекурсивні матричні алгоритми можуть ефективно масштабуватися у паралельних обчислювальних системах із динамічним розподілом обчислювальних ресурсів [6]. Для реалізації таких алгоритмів розроблено середовище виконання DAP, яке забезпечує ефективне розпаралелювання рекурсивних алгоритмів у системах із розподіленою пам'яттю [10, 11]. Це створює передумови для розробки нових блоково-рекурсивних алгоритмів.

Метою даної роботи є розробка та дослідження ефективного алгоритму ортогонального (унітарного) розкладу симетричних матриць до стрічкової форми на основі блоково-рекурсивних QR та QP пере-

творень. Для досягнення цієї мети у роботі запропоновано алгоритм Q3RP, що базується на послідовному застосуванні QR та QP перетворень. Також проведено аналіз обчислювальної складності алгоритму та експериментальне дослідження його продуктивності у паралельному середовищі виконання DAP із використанням прискорення матричних операцій на графічному процесорі.

Ортогональний розклад симетричної матриці

Ортогональний (унітарний) розклад симетричної матриці до тридіагональної форми є одним із ключових етапів у багатьох алгоритмах знаходження власних значень. Класичні методи такого розкладу, зокрема, метод відбиттів Хаусхолдера, мають складність порядку $\boldsymbol{O(n^3)}$ і реалізуються переважно як послідовні алгоритми, що обмежує їхню ефективність у паралельних обчислювальних системах [1, 2]. У зв'язку з цим значний інтерес становлять блоково-рекурсивні алгоритми, які дозволяють краще використовувати паралельну архітектуру сучасних обчислювальних платформ.

QR-розклад. Одним із таких підходів є блоково-рекурсивні алгоритми, що базуються на QR-розкладі матриці. QR-розклад представляє довільну матрицю $\boldsymbol{A}$ у вигляді добутку ортогональної матриці Q та верхньої трикутної матриці $\boldsymbol{R}$:

$$\boldsymbol{A = Q^T R}, \quad \boldsymbol{Q^T Q = Q Q^T = I}, \tag{1}$$

де $\boldsymbol{Q}$ — ортогональна матриця, а $\boldsymbol{R}$ — верхня трикутна матриця.

Одним із поширених способів побудови такого розкладу є метод обертань Ґівенса, який послідовно занулює піддіагональні елементи матриці за допомогою ортогональних матриць повороту. Для двох елементів вектора $\boldsymbol{(a, b)^T}$ відповідна матриця повороту має вигляд

$$\boldsymbol{g(\alpha, \gamma)} = \begin{pmatrix} \boldsymbol{\alpha} & \boldsymbol{\gamma} \\ \boldsymbol{-\gamma} & \boldsymbol{\alpha} \end{pmatrix}, \quad \boldsymbol{\alpha^2 + \gamma^2 = 1}, \tag{2}$$

що забезпечує ортогональність перетворення. Узагальненням такого перетворення для матриці є матриця Ґівенса

$$\boldsymbol{G(i, j)} = \mathrm{diag}(\boldsymbol{I, g(\alpha, \gamma), I}), \tag{3}$$

яка відрізняється від одиничної лише у двох рядках. Завдяки цьому відповідні перетворення є локальними та придатними для паралельної реалізації. Для щільної матриці розміру $\boldsymbol{n \times n}$ обчислювальна складність такого алгоритму становить приблизно $\boldsymbol{2n^3}$ арифметичних операцій.

Подальший розвиток цієї ідеї привів до появи блоково-рекурсивних алгоритмів QR-розкладу. У роботі [6] запропоновано алгоритм, у якому матриця

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix} \tag{4}$$

розбивається на чотири блоки однакового розміру. Алгоритм складається з трьох основних етапів.

На першому етапі виконується QR-розклад блоку $\boldsymbol{C}$

$$\boldsymbol{Q_1 C = C^U}, \tag{5}$$

у результаті чого формується верхньотрикутна матриця $\boldsymbol{C^U}$ та ортогональне перетворення $\boldsymbol{Q_1}$. Отримане перетворення застосовується до всієї матриці.

Другим етапом є спеціальна процедура, яка називається QP-розкладом – Рис. 1. Вона використовується для занулення елементів області, утвореної нижньою трикутною частиною блоку $\boldsymbol{A}$ та верхньою трикутною частиною блоку $\boldsymbol{C^U}$. Ця область має форму так званого «паралелограму». У процесі QP-розкладу будується ортогональне перетворення $\boldsymbol{Q_2}$, яке перетворює матрицю

$$\boldsymbol{P} = \begin{pmatrix} \boldsymbol{A} \\ \boldsymbol{C^U} \end{pmatrix} \tag{6}$$

до верхньотрикутного вигляду.

У результаті застосування перетворення $\boldsymbol{Q_2}$ матриця $\boldsymbol{M_1}$ набуває вигляду

$$\boldsymbol{M_2 = Q_2} \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C^U} & \boldsymbol{D_1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A^U} & \boldsymbol{B_1} \\ \boldsymbol{0} & \boldsymbol{D_2} \end{pmatrix}, \tag{7}$$

де $\boldsymbol{A^U}$ — верхньотрикутний блок, $\boldsymbol{B_1}$ — оновлений правий блок, а $\boldsymbol{D_2}$ — модифікований нижній блок матриці.

На третьому етапі виконується QR-розклад оновленого блоку $\boldsymbol{D_2}$, що приводить до загального розкладу

$$\boldsymbol{M} = \boldsymbol{Q}\,\boldsymbol{R}, \tag{8}$$

де

$$\boldsymbol{Q} = \mathrm{diag}(\boldsymbol{I}, \boldsymbol{Q_3})\,\boldsymbol{Q_2}\,\mathrm{diag}(\boldsymbol{I}, \boldsymbol{Q_1}). \tag{9}$$

QP-розклад. QP-розклад будується за блоково-рекурсивним принципом. Ми шукаємо розклад матриці $\boldsymbol{P}$:

$$\boldsymbol{P} = \begin{pmatrix} \boldsymbol{A} \\ \boldsymbol{C^U} \end{pmatrix}$$

$$Q_2 P = P^U, \quad P = Q_2^T P^U, \quad Q_2^{-1} = Q_2^T. \tag{10}$$

Матриця P є лівою половиною матриці M_1. Ми розбили кожен із двох блоків, що утворюють матрицю P (розміру $2n \times n$), на чотири рівні частини. У результаті отримано 8 блоків, включаючи один нульовий блок і два верхньотрикутні блоки f^U та h^U. Необхідно занулити всі елементи матриці P, розташовані між верхньою та нижньою діагоналями, включно із нижньою діагоналлю. Нас цікавить блокова процедура. Оскільки n є парним, паралелограм, утворений цими діагоналями, можна розділити на 4 частини за допомогою двох середніх ліній. У результаті отримуємо 4 однакові паралелограми. Для розкладу кожного з них процедура QP викликається рекурсивно 4 рази. Обробка виконується у такому порядку: спочатку нижній лівий паралелограм (P_{ll}), потім одночасно зануляються верхній лівий (P_{ul}) та нижній правий (P_{lr}), і на завершення виконується розклад паралелограма у верхньому правому куті (P_{ur}). Відповідні ортогональні матриці розміру $n \times n$ позначаються Q_{ll}, Q_{ul}, Q_{lr}таQ_{ur}.

$$P = (AC^U) = \begin{pmatrix} a & c \\ b & d \\ f^U & g \\ 0 & h^U \end{pmatrix},$$

$$P^U = (A^U 0) = \begin{pmatrix} a^U & c_1 \\ 0 & d^U \\ 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

$$P_{ll} = Q_{ll}\begin{pmatrix} b & d \\ f^U & g \end{pmatrix} = \begin{pmatrix} b^U & d_1 \\ 0 & g_1 \end{pmatrix},$$

$$P_{ul} = Q_{ul}\begin{pmatrix} a & c \\ b^U & d_1 \end{pmatrix} = \begin{pmatrix} a^U & c_1 \\ 0 & d_2 \end{pmatrix},$$

$$P_{lr} = Q_{lr}\begin{pmatrix} 0 & g_1 \\ 0 & h^U \end{pmatrix} = \begin{pmatrix} 0 & g^U \\ 0 & 0 \end{pmatrix},$$

$$P_{ur} = Q_{ur}\begin{pmatrix} 0 & d_2 \\ 0 & g^U \end{pmatrix} = \begin{pmatrix} 0 & d^U \\ 0 & 0 \end{pmatrix}. \tag{11}$$

У результаті отримуємо матриці Q_2 та P^U:

$$Q_2 = \overline{Q_{ur}}\,\overline{Q_2}\,\overline{Q_{ll}}, \quad Q_2 P = P^U, \tag{12}$$

де

$$\overline{Q_{ur}} = \mathrm{diag}\left(I_{\frac{n}{2}}, Q_{ur}, I_{\frac{n}{2}}\right),$$
$$\overline{Q_2} = \mathrm{diag}(Q_{ul}, Q_{lr}),$$
$$\overline{Q_{ll}} = \mathrm{diag}\left(I_{\frac{n}{2}}, Q_{ll}, I_{\frac{n}{2}}\right). \tag{13}$$

Схему блоково-рекурсивного QR-розкладу представлено на Рис. 1.

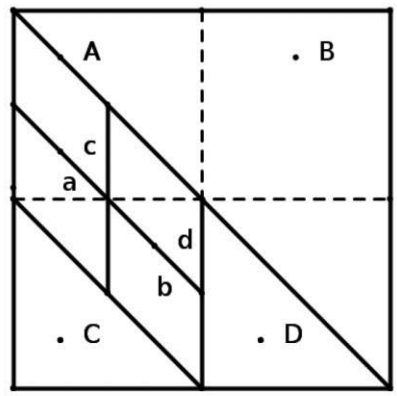

Рис. 1. Схема блоково-рекурсивного QR-розкладу

Складність QP-розкладу можна оцінити, підрахувавши кількість множень матричних блоків. У процесі виконання алгоритму необхідно виконати 28 множень блоків розміру $\boldsymbol{n/2} \times \boldsymbol{n/2}$: 8 множень використовуються у процесі обчислення матриць $\boldsymbol{P_{ll}}$ та $\boldsymbol{P_{ul}}$, а ще 20 — при побудові матриці $\boldsymbol{Q_2}$. Тому загальна кількість операцій задовольняє рекурентному співвідношенню

$$\boldsymbol{C_{qp}(2n) = 4C_{qp}(n) + 28M(n/2)}, \tag{14}$$

де $\boldsymbol{M(n)}$ — складність множення матриць розміру $\boldsymbol{n \times n}$.

Нехай складність множення двох матриць розміру $\boldsymbol{n \times n}$ дорівнює

$$\boldsymbol{M(n) = \gamma n^{\beta}},$$

(15)

а $\boldsymbol{n = 2^k}$. Тоді рекурентне співвідношення набуває вигляду

$$\boldsymbol{C_{qp}(2^{k+1}) = 4C_{qp}(2^k) + 28M(2^{k-1})}.$$

(16)

Розгортаючи рекурсію, отримуємо

$$\boldsymbol{C_{qp}(2^{k+1}) = 4^k C_{qp}(2) + 28\gamma \sum_{i=0}^{k-1} 4^{k-i-1} 2^{i\beta}}.$$

(17)

Для розв'язання рекурентного співвідношення врахуємо базовий випадок

$$\boldsymbol{C_{qp}(2) = 6}.$$

(18)

Тоді отримуємо явний вираз для складності QP-розкладу

$$\boldsymbol{C_{qp}(n) = \frac{28\gamma(n^{\beta} - n^2)}{2^{\beta}(2^{\beta} - 4)} + 6n^2}.$$

(19)

Обмежуючись старшим ступенем за n, отримуємо асимптотичну оцінку

$$\boldsymbol{C_{qp}(n) = \varphi(\beta)\gamma n^{\beta}, \quad \varphi(\beta) = \frac{28}{2^{\beta}(2^{\beta} - 4)}}.$$

$$\boldsymbol{\varphi(3) = \frac{7}{8}, \quad \varphi(\log_2 7) = \frac{4}{3}}.$$

(20)

Складність блоково-рекурсивного QR-розкладу визначається з урахуванням того, що алгоритм містить два рекурсивні виклики QR-процедури, одну процедуру QP-розкладу та кілька операцій множення матриць. Якщо позначити складність QP-розкладу як

$$\boldsymbol{C_{qp}(n) = \alpha n^{\beta}},$$

(21)

а складність множення матриць як (15), то кількість операцій QR-розкладу задовольняє рекурентному співвідношенню

$$\boldsymbol{C_{qr}(n) = 2C_{qr}(n/2) + C_{qp}(n/2) + 8M(n/2)}$$

(22)

Нехай $\boldsymbol{n = 2^k}$. Тоді

$$\begin{aligned}\boldsymbol{C_{qr}(2^k)} &= \boldsymbol{2C_{qr}(2^{k-1}) + C_{qp}(2^{k-1}) + 8M(2^{k-1})} \\ &= \boldsymbol{2^k C_{qr}(2^0) + \sum_{i=0}^{k} 2^{k-i} C_{qp}(2^{i-1}) + 8\sum_{i=0}^{k} 2^{k-i} M(2^{i-1})} \\ &= \boldsymbol{\gamma(\phi + 8)\sum_{i=0}^{k} 2^{k-i}(2^{\beta(i-1)})} \\ &= \boldsymbol{\gamma(\phi + 8)\frac{2^{\beta}n^{\beta} - 2n}{2^{\beta}(2^{\beta} - 2)}}\end{aligned}$$

(23)

Обмежуючись старшим ступенем за $\boldsymbol{n}$, отримуємо:

$$\begin{aligned}&\boldsymbol{C_{qr}(n) = \rho(\beta)\gamma n^{\beta},\ \rho(\beta) = (\phi + 8)/(2^{\beta} - 2)} \\ &\boldsymbol{\rho(3) = 71/48,\ \rho(\log_2 7) = 14/9}\end{aligned}$$

(24)

Алгоритм Q3RP. Описані вище властивості блоково-рекурсивних QR та QP розкладів дозволяють використовувати їх як базові операції для побудови більш складних алгоритмів матричних перетворень. Одним із них є алгоритм, який ми будемо називати Q3RP, призначений для ортогонального розкладу симетричної матриці до стрічкової форми.

Основна ідея алгоритму полягає в поетапному зануленні цілих блокових шарів симетричної матриці за допомогою послідовності QR та QP розкладів. На відміну від класичних алгоритмів, де елементи занулюються послідовними перетвореннями окремих рядків або стовпців, у цьому алгоритмі занулення виконується для цілих блоків матриці. Це дозволяє відокремлювати та незалежно обчислювати відповідні підзадачі, що забезпечує високий рівень паралелізму та робить алгоритм ефективним для обробки великих матриць на кластерах з розподіленою пам’яттю.

Нехай задано симетричну матрицю $\boldsymbol{M}$ розміру $\boldsymbol{n \times n}$. На першому етапі матриця розбивається на чотири великі блоки

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix}, \tag{25}$$

де кожен блок має розмір $\boldsymbol{n/2} \times \boldsymbol{n/2}$. Далі кожен із цих блоків додатково ділиться на чотири підблоки розміру $\boldsymbol{n/4} \times \boldsymbol{n/4}$. Таким чином, матриця розбивається на 16 рівних блоків.

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix} = \begin{pmatrix} \boldsymbol{a_{1,1}} & \boldsymbol{a_{1,2}} & \boldsymbol{b_{1,1}} & \boldsymbol{b_{1,2}} \\ \boldsymbol{a_{2,1}} & \boldsymbol{a_{2,2}} & \boldsymbol{b_{2,1}} & \boldsymbol{b_{2,2}} \\ \boldsymbol{c_{1,1}} & \boldsymbol{c_{1,2}} & \boldsymbol{d_{1,1}} & \boldsymbol{d_{1,2}} \\ \boldsymbol{c_{2,1}} & \boldsymbol{c_{2,2}} & \boldsymbol{d_{2,1}} & \boldsymbol{d_{2,2}} \end{pmatrix} \tag{26}$$

Схематично це представлено на Рис. 2.

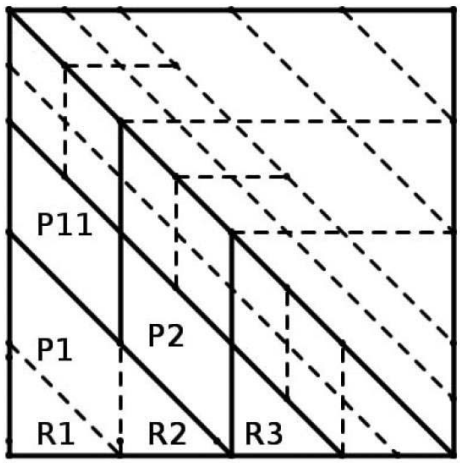

Рис. 2. Схема розбиття симетричної матриці на блоки для алгоритму Q3RP

Першим кроком алгоритму є виконання QR-розкладу блоку $\boldsymbol{c_{2,1}}$. У результаті цей блок набуває верхньотрикутної форми, а отримана ортогональна матриця $\boldsymbol{Q_1}$ задає відповідне ортогональне перетворення, яке застосовується до інших блоків матриці. Завдяки симетричності матриці таке перетворення одночасно впливає і на симетричний блок $\boldsymbol{b_{1,2}}$. Після цього виконується QP-розклад, який дозволяє усунути решту елементів у відповідному верхньотрикутному блоці. Добуток отриманих ортогональних матриць утворює узагальнену матрицю повороту $\boldsymbol{Q_3}$. Поточна матриця домножується на $\boldsymbol{Q_3^T}$ зліва та на $\boldsymbol{Q_3}$ справа, що забезпечує збереження її симетричності. На наступному етапі аналогічно обробляються блоки $\boldsymbol{c_{1,1}^U}$ та симетричний до нього блок $\left(\boldsymbol{b_{1,1}}\right)^{\boldsymbol{U}}$ за допомогою QP-розкладу. Далі послідовно застосовуються QR- та QP-розклади для блоків $\boldsymbol{c''_{2,2}}$ та $\boldsymbol{c'''_{1,2}}$. Кожен із цих кроків породжує нові ортогональні перетворення, які комбінуються у відповідну матрицю повороту для поточного шару.

У результаті послідовного застосування цих перетворень усі блоки матриці $\boldsymbol{C}$ зануляються, що приводить до повного занулення нижнього лівого блоку матриці разом із відповідними симетричними елементами верхнього правого блоку.

Завершальним етапом процедури є QR-розклад блоку $\boldsymbol{d_{2,1}^{(5)}}$. Отримана матриця повороту $\boldsymbol{Q_8}$ знову застосовується до матриці зліва та справа, що завершує формування стрічкової структури. У результаті отримуємо представлення початкової матриці у вигляді

$$\boldsymbol{S} = \boldsymbol{Q^T M Q}, \tag{27}$$

де $\boldsymbol{Q} = \boldsymbol{Q_1 Q_2 \cdots Q_8}$ є добутком ортогональних матриць, отриманих на кожному кроці алгоритму, а $\boldsymbol{S}$ — результуюча стрічкова матриця. Матриця $\boldsymbol{S}$ має спеціальну структуру: усі елементи, що знаходяться поза межами діагональної смуги шириною $\boldsymbol{p} = \frac{\boldsymbol{n}}{\boldsymbol{4}}$, дорівнюють нулю. Ширина цієї смуги визначається розміром блоків, на які розбивається початкова матриця. Отримана стрічкова форма є проміжним результатом, який надалі може бути перетворений до тридіагонального вигляду за допомогою спеціалізованих алгоритмів звуження смуги, зокрема, алгоритму Ribbon, який буде представлено в наступній статті.

Складність Q3RP. Нехай задано симетричну матрицю $\boldsymbol{M}$ розміру $\boldsymbol{2^k \times 2^k}$. Для оцінки складності алгоритму використаємо складності QR та QP розкладів, описані у попередньому розділі, та припустимо, що складність множення матриць задається функцією

$$\boldsymbol{M(n) = \gamma n^{\beta}}. \tag{28}$$

Аналіз послідовності операцій алгоритму Q3RP показує, що для матриці розміру $\boldsymbol{n}$ необхідно виконати:

- три QR-розклади для матриць розміру $\boldsymbol{n/4}$;

- три QP-розклади для матриць розміру $\boldsymbol{n/4}$;
- шість множень матриць розміру $\boldsymbol{n/2}$;
- двадцять вісім множень матриць розміру $\boldsymbol{n/4}$.

Оскільки множення матриць розміру $\boldsymbol{n/2}$ можна звести до множень блоків розміру $\boldsymbol{n/4}$, усі матричні множення можна представити як множення блоків одного розміру $\boldsymbol{n/4}$. Загальна кількість таких множень становить 50. Тож загальну складність алгоритму можна записати у вигляді

$$\begin{aligned} \boldsymbol{C_n} &= \boldsymbol{3C_{qr}(n/4) + 3C_{qp}(n/4)} \\ &\quad \boldsymbol{+ 50M(n/4)} \\ &\quad \boldsymbol{= (3\phi + 3\rho + 50)\gamma\left(n^{\beta}\right)/4^{\beta}} \end{aligned} \tag{29}$$

тобто:

$$\boldsymbol{C_n = \tau\gamma n^{\beta}}$$

$$\boldsymbol{\tau(\beta) = (3\phi + 3\rho + 50)/4^{\beta}}$$

$$\boldsymbol{\phi + \rho = \left(28\left(2^{\beta} - 1\right) + 8\left(2^{\beta}\left(2^{\beta} - 4\right)\right)\right) / \left(\left(2^{\beta} - 2\right)\left(2^{\beta}\left(2^{\beta} - 4\right)\right)\right)}$$

$$\boldsymbol{\phi + \rho = \left(8 \cdot 2^{2\beta} + 4 \cdot 2^{\beta} - 28\right) / \left(\left(2^{\beta} - 2\right)\left(2^{\beta}\left(2^{\beta} - 4\right)\right)\right)}$$

$$\boldsymbol{\tau(\beta) = \left(50 \cdot 2^{3\beta} + 412 \cdot 2^{\beta} - 84 - 276 \cdot 2^{2\beta}\right) / \left(\left(2^{\beta} - 2\right)2^{3\beta}\left(2^{\beta} - 4\right)\right)}$$

$$\boldsymbol{\tau(3) = 0.91, \quad \tau(\log_2 7) = 1.25}$$

(30)

Отже, асимптотична складність алгоритму Q3RP має той самий порядок, що і складність множення матриць. Використання швидких алгоритмів множення матриць, зокрема, алгоритму Штрассена або його узагальнень, дозволяє зменшити асимптотичний показник степеня $\boldsymbol{\beta}$ і, відповідно, покращити теоретичну оцінку складності алгоритму Q3RP. Для порівняння класичний алгоритм розкладу симетричної матриці до тридіагональної форми на основі відбиттів Хаусхолдера має складність приблизно $\frac{4}{3}\boldsymbol{n^3}$ арифметичних операцій. Однак цей метод реалізується переважно як послідовний алгоритм і не має природної блоково-рекурсивної структури, що обмежує можливості його ефективної паралельної реалізації. На відміну від нього, алгоритм Q3RP використовує блоково-рекурсивні QR та QP перетворення, що дозволяє виконувати більшість обчислень у вигляді операцій над матричними блоками. Така структура алгоритму добре узгоджується з паралельними обчислювальними середовищами та дозволяє ефективно використовувати високопродуктивні обчислювальні системи.

Експерименти

Програмна реалізація алгоритму виконана у середовищі блоково-рекурсивних паралельних обчислень DAP, яке є децентралізованою системою керування паралельними задачами [10, 11]. Система реалізована мовою Java з використанням MPI для організації взаємодії між обчислювальними вузлами. Для запуску алгоритму формується обчислювальний граф, вершинами якого є обчислювальні блоки (так звані *дропи*), а ребрами — зв'язки передачі даних між ними. Така структура дозволяє природно реалізувати рекурсивні матричні алгоритми у вигляді набору незалежних підзадач, які можуть виконуватися паралельно. У реалізації алгоритму Q3RP основні обчислювально затратні операції — QR-розклади, QP-розклади та множення матриць — додатково прискорюються за рахунок використання графічного процесора (GPU). Використання GPU дозволяє ефективно виконувати великі блокові матричні операції, що є ключовими для блоково-рекурсивної структури алгоритму. Експерименти проводилися на одній обчислювальній ноді з такими характеристиками: CPU: Intel Core i5-12600K; RAM: 32 GB DDR4; GPU: NVIDIA RTX 3070 (8 GB VRAM). Усі обчислення виконувалися у подвійній точності (double precision). Розмір листової вершини у середовищі DAP становив 128, що визначає глибину рекурсії алгоритму. Вхідні матриці генерувалися як щільні матриці зі щільністю 50 % ненульових елементів. Для кожного розміру матриці час виконання визначався як середнє значення за серією запусків.

Таблиця 1.

Час виконання та точність алгоритму Q3RP

N	Час виконання, мс	Абсолютна похибка
256	98	1.60e−15
512	121	2.34e−15
1024	354	2.71e−15
2048	1863	3.38e−15
4096	9617	4.36e−15

Отримані результати наведені в Таблиці 1 показують, що алгоритм Q3RP демонструє стабільну продуктивність при збільшенні розміру матриці.

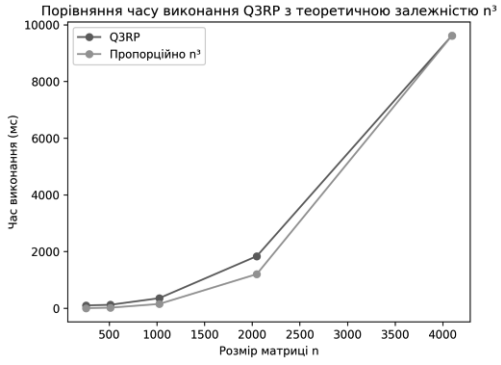

Рис. 3. Час виконання алгоритму Q3RP

З графіка на Рис. 3 видно, що зі збільшенням розміру задачі час виконання алгоритму зростає приблизно кубічно, а це відповідає теоретичній оцінці складності алгоритму.

Крім продуктивності важливою характеристикою алгоритму є його чисельна стабільність. Для оцінки точності було виміряно абсолютну похибку реконструкції матриці після виконання розкладу. Похибка визначалася як норма різниці між початковою матрицею та матрицею, відновленою після застосування відповідних ортогональних перетворень. Результати показують, що абсолютна похибка знаходиться на рівні 10^{-15}, що відповідає машинній точності для чисел типу double. Це свідчить про високу чисельну стабільність алгоритму Q3RP.

Отримані експериментальні результати підтверджують ефективність запропонованого алгоритму Q3RP для ортогонального розкладу симетричних матриць до стрічкової форми. Блоково-рекурсивна структура алгоритму дозволяє виконувати більшість обчислень у вигляді великих матричних операцій, що добре узгоджується з архітектурою сучасних обчислювальних систем.

Використання прискорення на GPU для виконання QR- та QP-розкладів і операцій множення матриць дозволяє суттєво зменшити час виконання алгоритму при зростанні розміру задачі. Водночас алгоритм зберігає високу точність обчислень, що підтверджується малими значеннями похибки реконструкції.

Висновки

У роботі описано блоково-рекурсивні алгоритми ортогонального (унітарного) розкладу матриць та отримано оцінки їхньої обчислювальної складності. Показано, що кількість арифметичних операцій визначається алгоритмом множення матриць, який використовується у відповідних блокових обчисленнях. Водночас кількість операцій у запропонованих алгоритмах у будь-якому випадку менша, ніж у двох операціях множення матриць того самого розміру. Отже, для матриць великого розміру ефективними є алгоритми, що використовують швидкі методи множення матриць.

Перевагою запропонованих алгоритмів у суперкомп’ютерних обчисленнях є їхня блокова структура. Класичні алгоритми, зокрема, алгоритм Хаусхолдера, не дозволяють природно відокремлювати незалежні блокові підзадачі, тому розпаралелювання задач великого розміру пов’язане зі значними труднощами.

Запропоновані блокові алгоритми усувають ці обмеження. Вони дають змогу виокремлювати незалежні блокові підзадачі та забезпечують паралельний і конвеєрний характер обчислювального процесу, що є особливо важливим для виконання обчислень на кластерах із розподіленою пам’яттю.

Література

1. Golub G. H., Van Loan C. F. Matrix Computations. Baltimore: Johns Hopkins University Press, 2013. 756 p.

2. Trefethen L. N., Bau D. Numerical Linear Algebra. Philadelphia: SIAM, 1997.
3. Strassen V. Gaussian elimination is not optimal. Numerische Mathematik. 1969. Vol. 13. No. 4. P. 354–356.
4. Coppersmith D., Winograd S. Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation. 1990. Vol. 9. No. 3. P. 251–280.
5. Schönhage A. Unitäre Transformationen großer Matrizen. Numerische Mathematik. 1973. Vol. 20. P. 409–417.
6. Malaschonok G. Recursive matrix algorithms, distributed dynamic control, scaling, stability. Computer Science and Information Technologies (CSIT). Yerevan, 2019. P. 112–115. doi:10.1109/CSITechnol.2019.8895255.
7. Malaschonok G., Ivaskevich A. Quick recursive QR decomposition. Proceedings of the Conference on Mathematical Foundations of Informatics (MFOI-2020). Kyiv, 2020. P. 1–8.
8. Першута П. В. Алгоритм зведення симетричної матриці до тридіагональної форми із застосуванням QR та QP-розкладів: магістерська робота. Київ, 2024. 55 с.
9. Demmel J., Dumitriu I., Holtz O., Kleinberg R. Fast Linear Algebra is Stable. Numerische Mathematik. 2007. Vol. 108. P. 59–91.
10. Сідько А. А. Середовище виконання для блоково-рекурсивних матричних алгоритмів на суперкомп'ютері з розподіленою пам'яттю: дис. … д-ра філософії. Київ, 2024.
11. Малашонок Г. І., Сідько А. А. Розподілені обчислення: ДАП-технологія розпаралелювання рекурсивних алгоритмів. Наукові записки НаУКМА. 2018. № 1. С. 25–32.

References

1. Golub, G.H. and Van Loan, C.F., 2013. Matrix Computations. Baltimore: Johns Hopkins University Press.
2. Trefethen, L.N. and Bau, D., 1997. Numerical Linear Algebra. Philadelphia: SIAM.
3. Strassen, V., 1969. Gaussian elimination is not optimal. Numerische Mathematik, 13(4), pp.354–356.
4. Coppersmith, D. and Winograd, S., 1990. Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation, 9(3), pp.251–280.
5. Schönhage, A., 1973. Unitäre Transformationen großer Matrizen. Numerische Mathematik, 20, pp.409–417.
6. Malaschonok, G., 2019. Recursive matrix algorithms, distributed dynamic control, scaling, stability. In: Computer Science and Information Technologies (CSIT). Yerevan, pp.112–115. https://doi.org/10.1109/CSITechnol.2019.8895255
7. Malaschonok, G. and Ivaskevich, A., 2020. Quick recursive QR decomposition. In: Proceedings of the Conference on Mathematical Foundations of Informatics (MFOI-2020). Kyiv, pp.1–8.
8. Pershuta, P.V., 2024. Algorithm for reducing a symmetric matrix to tridiagonal form using QR and QP decompositions (Master's thesis). Kyiv (in Ukrainian).
9. Demmel, J., Dumitriu, I., Holtz, O. and Kleinberg, R., 2007. Fast linear algebra is stable. Numerische Mathematik, 108, pp.59–91.
10. Sidko, A.A., 2024. Execution environment for block-recursive matrix algorithms on a distributed-memory supercomputer (PhD dissertation). Kyiv (in Ukrainian).
11. Malashonok, H.I. and Sidko, A.A., 2018. Distributed computing: DAP-technology for parallelization of recursive algorithms. Naukovi zapysky NaUKMA, 1, pp.25–32 (in Ukrainian).

Дата першого надходження до видання: 13.03.2026
Внутрішня рецензія отримана: 16.03.2026
Зовнішня рецензія отримана: 17.03.2026
Дата прийняття статті до друку: 19.03.2026
Дата публікації: 16.04.2026

Про авторів:

[1]Малашонок Геннадій Іванович,
Доктор фізико-математичних наук, професор
[1] Malaschonok Gennadiy,
Ph.D (doctor, physical and mathematical sciences), professor
https://orcid.org/0000-0002-9698-6374

[2]Сухарський Сергій Сергійович,
PhD студент
[2] Sukharskyi Sergiy,
Post-graduate student
https://orcid.org/0000-0002-5873-984X

Місце роботи авторів:

[1]Національний університет
«Києво-Могилянська академія»
[1] National University
“Kyiv–Mohyla Academy”
malaschonok@gmail.com

[2]Інститут Програмних Систем
НАН України
[2] Institute of Software Systems.
National Academy of Sciences of Ukraine
serhii.sukharskyi@gmail.com

ВИМОГИ ДО ОФОРМЛЕННЯ СТАТЕЙ

1. Загальні положення

У журналі "Проблеми програмування" публікуються наукові матеріали, які раніше не публікувалися в інших виданнях.

Мова статті: українська, англійська. 16.07.2020 р. набули чинності положення Закону України «Про забезпечення функціонування української мови як державної». Відповідно до статті 22 «Державна мова у сфері науки» у наукових виданнях не повинно бути вміщено матеріалів іншими мовами, окрім державної, англійської та мов ЄС.

Обсяг статті - від 6 до 16 сторінок формату А4. Документ зберігається у форматі doc або docx.

Назва файлу включає транслітерацію прізвища автора (авторів), наприклад, “Petrenko.doc”.

Стаття надається без нумерації сторінок.

Для передачі до редакції тексту статті, ділової переписки та правки при коректурі автори користуються електронною поштою редакції: alengoro@isofts.kiev.ua, alengoro2022@gmail.com. Для надійності інформаційного обміну в умовах можливих відключень електрики – прохання надсилати матеріали на обидві електронні пошти одночасно. Телефон: +380 (96) 418 3082.

2. Оформлення файлу з текстом статті

При підготовці файлу використовуються: стиль нормальний (звичайний) або normal; шрифт Times New Roman, розмір шрифту 12 пт.; міжрядковий інтервал – 1,0; абзацний відступ -1,25 см; вирівнювання – по ширині. У тексті не допускається вирівнювання пропусками; розстановка переносів – автоматична. Формат паперу А4, розміри полів документа – 20 мм. Текст статті після зазначення авторів, назви і анотацій (двома мовами) має бути оформлений **у 2 колонки,** ширина яких – 7,86 см, а пробіл між ними – 1,27 см.

3. Послідовність розміщення та оформлення матеріалу статті

1. ***Верхній колонтитул:*** назва рубрики відповідно до переліку, прийнятому редакцією журналу *(пропонується авторами, остаточно уточняється редакцією).*

УДК *(зліва під рискою верхнього колонтитулу)**:*** індекс за універсальною десятковою класифікацією. ***DOI:*** в тому ж рядку правіше *(заповнюється редакцією)*.

Автори: ініціали та прізвища авторів, курсив *(світлий)*.

Заголовок 1 ***(назва статті)***: не містить абревіатур та строго відповідає змісту статті. Шрифт 15 пт, напівжирний, регістр верхній, вирівнювання по центру.

Анотація: 1800-2100 знаків враховуючи пробіли, не має бути абревіатур. Шрифт 10 пт, звичайний.

Ключові слова: не більше 10 слів, не містить абревіатур, подаються в називному відмінку, розділені комами. Шрифт 10 пт, звичайний.

УВАГА!

Автори, заголовок статті, анотація і ключові слова зазначаються **ДВІЧІ:** українською і англійською мовами. Спочатку мовою статті, потім іншою мовою.

2. ***Нижній колонтитул*** (тільки для першої сторінки) включає стандартну інформацію Copyright: перший рядок – прізвища авторів, рік; другий рядок – номер ISSN, назва журналу, рік, номер випуску.
3. **Заголовок 2** ***(назва розділу)***: шрифт 14 пт, напівжирний; абзац із центральним вирівнюванням, без переносів. Заголовки нижчого рівня *(пункти і т.п.)* у

самостійний абзац не виділяються і проходять першим реченням текстового абзацу, шрифт 12 пт, напівжирний.

4. **Формули** створюються в редакторі Microsoft Equation 3.0 або MathType. Формули, на які є посилання в тексті, повинні мати наскрізну нумерацію. Номер формули друкується в круглих дужках біля краю правого поля. Розмір основного шрифту редактора формул – 12 пт. Розміри символів у формулах: звичайний – 12 пт, великий індекс – 9 пт, дрібний індекс – 7 пт, великий символ – 18 пт, дрібний символ – 11 пт. Не допускається масштабування формульних об'єктів. Великі формули мають бути розбиті на декілька рядків.

Наприклад:

$$\frac{\partial T}{\partial t}+\frac{u}{a\cos\varphi}\frac{\partial T}{\partial \lambda}+\frac{v}{a}\frac{\partial T}{\partial \varphi}+w\frac{\partial T}{\partial z}=\frac{\delta}{c_V\rho}, \quad (1)$$

де λ – довгота, φ – широта, z – висота над рівнем моря, $V=(u,v,w)$, a – радіус Землі, ω – швидкість добового обертання Землі, $F_f=(F_\lambda, F_\varphi, F_z)$.

5. **Рисунки** мають бути створені вбудованим редактором Word Picture або експортовані з прикладних програм Windows у графічних форматах (bmp, pcx, gif, jpg або tif). Рисунки розташовуються по центру. Нумерація рисунків здійснюється відповідно до порядку згадування у тексті. Нумеровані підписи розміщуються під рисунком з позначенням "Рис. ", далі вказується номер рисунка і текст підпису.
6. **Таблиці** мають бути підготовлені стандартним вбудованим в Word інструментарієм "Таблиця". Таблиці нумеруються за порядком згадування. На номер таблиці мають бути посилання в тексті. Номер таблиці вказується в окремому рядку з вирівнюванням по правій стороні (наприклад: "Таблиця 1"). Назви таблиць розміщуються над таблицею з вирівнюванням по центру. Мінімальний розмір шрифту в таблицях – 11 пт.

При посиланні на формулу, рисунок, таблицю або літературне джерело, використовуйте наступні позначення відповідно: (1), (1, 2); Рис.1, Рис.1, 2; Табл.1., Табл.1, 2; [1], [1, 2].

7. **Основний текст статті** має такі необхідні елементи:

– постановка проблеми в загальному вигляді і її зв'язок з важливими науковими або практичними завданнями;

– аналіз останніх досліджень і публікацій, у яких розпочато рішення даної проблеми і на які спирається автор, виділення невирішених раніше частин загальної проблеми, яким присвячується дана стаття;

– формулювання цілей статті (постановка задачі);

– виклад основного матеріалу дослідження з повним обґрунтуванням отриманих наукових результатів;

– висновки з даного дослідження і перспективи подальших розробок у даному напрямку;

– подяка (за наявності такої).

Застосований у статті маркований список (наведений вище) має наступні параметри: маркер має відступ 1,25 см, текст для першого рядка – 1,9 см. Аналогічні відступи слід підтримувати і для нумерованого списку.

8. **Література**: єдиний нумерований список джерел згідно ДСТУ 8302:2015 від 01.07.2016 р., шрифт 11 пт, відступ: спеціальний, навислий, 0,63 см.
9. **References:** література англійською мовою подається як список використовуваних джерел згідно *Harvard Style*. Джерела з заголовками на латиниці наводяться без

перекладу. Для літератури джерел на мовах, що не використовують латинський алфавіт, необхідно забезпечити переведення назв джерел і вказати після них у дужках мову оригіналу. Прізвища та ініціали авторів, слід транслітерувати за правилами як для закордонного паспорта.

Література, що надана другою мовою не враховується при підрахунку кількості сторінок статті. У випадках, коли список джерел включає джерела тільки однією мовою, він подається один раз.

10. ***Дата надходження статті*** позначається редакцією цифрами окремим рядком після слова «Одержано:»/”Received:”.

11. ***Дата надходження внутрішньої рецензії*** позначається редакцією цифрами окремим рядком після слів «Внутрішня рецензія отримана»/”Internal review received:”.

12. ***Дата надходження зовнішньої рецензії*** позначається редакцією цифрами окремим рядком після слів «Зовнішня рецензія отримана:»/” External review received:”.

Відомості про рецензентів конкретної статті не розголошуються для підвищення об’єктивності рецензування.

13. ***Дані про авторів*****:** мають починатися рядком “*Про авторів:*”, напівжирний курсив. Далі вказуються для кожного з авторів ПІБ повністю, вчений ступінь, наукове звання, посада, обов’язково номер ORCID (сайт ORCID http://orcid.org/). Особисті телефони та електронні пошти авторів вказуються тут тільки, якщо автор хоче, щоб вони були опубліковані в журналі.

Перелік авторів подається під номерами (надстроковим шрифтом), що відповідають нумерації місць роботи, де вони працюють (надстроковим шрифтом).

14. ***Дані про місце роботи авторів*****:** починаються рядком “*Місце роботи авторів:*”, напівжирний курсив. Далі вказуються місце роботи, телефон, електронна пошта, веб-сайт.

15. Обов’язково вказати мобільний телефон і e-mail відповідального виконавця для роботи з редактором при підготовці статті до друку. Ця інформація не публікується і призначена виключно для контакту редактора з авторами.

    Для полегшення підготовки статей, що задовольняють вищенаведеним вимогам, редакція журналу розробила файл шаблону статті “shablon.dot”, який можуть використовувати автори.