УДК 004.43:004.8
DOI: 10.31891/2219-9365-2021-67-1-9

КРИВИЙ В. М., ЯШИНА О. М.,
РАДЕЛЬЧУК Г. І., ЛИСЕНКО С. М.
Хмельницький національний університет

ПОРІВНЯЛЬНИЙ АНАЛІЗ ПАРАДИГМ ПРОГРАМУВАННЯ ПРИ РОЗРОБЦІ ПРОГРАМНИХ СИСТЕМ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ

У статті наведено результати досліджень різних парадигм програмування та мов, що підтримують ці парадигми, на предмет наявності інструментарію для розробки штучного інтелекту. Проведена оцінка зручності їх використання, аналіз сфер їх поширеності на ринку розробки програмного забезпечення та їх ефективність.

Ключові слова: штучний інтелект, парадигма програмування, програмне забезпечення, машинне навчання, нейронна мережа.

V. KRYVYI, O. YASHYNA, G. RADELCHUK, S. LYSENKO
Khmelnytskyi National University

COMPARATIVE ANALYSIS OF PROGRAMMING PARADIGMS IN THE DEVELOPMENT OF SOFTWARE SYSTEMS BASED ON ARTIFICIAL INTELLIGENCE

Today, with the growing popularity and demand for technologies of artificial intelligence, machine learning and neural networks, the question of choosing effective tools for their development and integration into software systems becomes relevant. With the progress of computer hardware, creating special programming languages and appearing of libraries that simplify the development and use of neural networks, artificial intelligence is no longer something futuristic and frightening. It is now a fairly flexible and widespread technology that is developing quickly. And on the one hand, it is gradually being introduced into our lives to perform those tasks that were previously unavailable or extremely difficult for the computer. On the other hand, artificial intelligence is not yet advanced enough to be called a completely reliable tool and cannot replace humans in many areas of their activity, so it is mostly developing for specific tasks such as digital content processing, data analysis, vehicle piloting, simulation of character behavior in games, etc. But as in the work with any other technologies in programming, we are interested in the selection of software tools and paradigms that allow us to effectively design and develop systems based on artificial intelligence.

This article presents the results of research on different programming paradigms and languages that support these paradigms, for the availability of tools for the development of artificial intelligence. It evaluates the ease of use, analyzes the areas of their prevalence in the software development market, and their effectiveness.

Keywords: artificial intelligence, programming paradigm, software, machine learning, neural network.

Вступ. Постановка проблеми. Сьогодні, із зростанням популярності та попиту на технології штучного інтелекту (ШІ), машинного навчання та нейронних мереж, актуальним стає питання вибору ефективних інструментів для їх розробки та інтеграції у програмні системи. З розвитком комп'ютерного обладнання та появою бібліотек, які спрощують розробку та використання нейронних мереж, штучний інтелект уже не є чимось футуристичним та лякаючим. Зараз це досить гнучка і поширена технологія, яка швидко розвивається. З одного боку, вона поступово впроваджується у наше життя для виконання тих завдань, які раніше були недоступними або надто важкими для виконання комп'ютером. З іншого боку, штучний інтелект ще не настільки високий, щоб його можна було назвати цілком надійним інструментом. Він не може замінити людей у багатьох сферах їх діяльності, тому здебільшого розробляється для конкретних завдань, таких, наприклад, як обробка цифрового вмісту, аналіз даних, пілотування транспортних засобів моделювання поведінки персонажа в іграх тощо. У цьому зв'язку, як і при роботі з будь-якими іншими технологіями програмування, особливий інтерес представляють парадигми програмування, які дозволяють ефективно розробляти програмні системи на основі ШІ.

Аналіз останніх досліджень та публікацій. Основою дослідження є праці відомих авторів та науковців у сфері програмування ШІ, таких як П. Грем, Д. Маккарті, І. Братко, П. Джоші та ін. Ми будемо аналізувати розробку штучного інтелекту на основі парадигм функціонального, імперативного, логічного та об'єктно-орієнтованого програмування, досліджуючи такі мови програмування як Prolog, Haskell, Python, Java, JavaScript, C#, F# та ін.

Метою роботи є: порівняльний аналіз різних парадигм та мов програмування, що підтримують ці парадигми, на предмет їх ефективності та зручності при розробці штучного інтелекту.

Виклад основного матеріалу. Парадигма програмування – це сукупність ідей та понять, які визначають стиль написання комп'ютерної програми (підхід до програмування, спосіб мислення розробника програми). Іншими словами, це концептуалізація, яка визначає організацію обчислень та структурування роботи, яку має виконувати комп'ютер [1].

ISSN 2219-9365

Основними парадигмами, що використовуються при розробці штучного інтелекту, є парадигми функціонального (ФП), логічного (ЛП), імперативного (ІП) та об'єктно-орієнтованого (ООП) програмування. Для дослідження цих парадигм знадобиться розгляд їх використання у конкретних мовах програмування, які підтримують ці парадигми, та роботі з ними при розробці ШІ.

Методологія даного дослідження полягає у використанні наступних критеріїв порівняння парадигм та мов програмування, що використовують ці парадигми:

– концепція парадигми та її пристосованість до задач, пов'язаних з розробкою ШІ;

– поширеність на ринку розробки програмного забезпечення (ПЗ) зі штучним інтелектом;

– можливість та легкість застосування у прикладних програмних системах;

– швидкість вирішення задачі.

Кожна парадигма програмування є лише теоретичною і концептуальною складовою процесу програмування, тому для їх оцінки за конкретними критеріями ми будемо не просто розглядати парадигми самі по собі, а їх конкретну реалізацію у мовах програмування, оскільки нас цікавлять не тільки теоретичні, але й практичні аспекти, з якими стикатиметься розробник програмного коду при вирішенні широкого спектру задач.

На сьогодні більшість широковживаних мов програмування на ринку розробки програмних систем (таких як Python, JavaScript, C#, C++, Java та ін.) були розроблені на основі ООП- та ІП-парадигм, незалежно від того, чи є вони інтерпретованими чи компільованими. Імперативний стиль написання коду визначає програму як сукупність даних, що визначають її стан, і покрокових інструкцій, що змінюють цей стан. А об'єктно-орієнтований аспект цих мов спонукає розробника програмного коду об'єднувати ці частини коду у класи, які можуть використовуватись багаторазово, наслідувати один одного, мати зв'язки з іншими класами тощо. З моменту появи мови C++ об'єднання імперативного та об'єктно-орієнтованого стилю у мовах програмування стало трендом і залишається таким до сьогодні.

Причини такого явища є очевидними – послідовні інструкції мови імперативного програмування можна набагато легше і ефективніше конвертувати у машинний код, оскільки він також є послідовністю інструкцій. Більше того, такий підхід набагато легше пояснити початківцю у програмуванні, і з ним легше уявити процес вирішення прикладної задачі, оскільки спосіб, який полягає у заданні послідовності дій для досягнення цілі, дуже наближений до реального життя. Програма стає схожою на кулінарний рецепт, де над початковими інгредієнтами крок за кроком проводяться деякі дії, що змінюють їх стан, допоки ті не перетворяться у готову страву. Так само просто можна зрозуміти і принцип ООП, оскільки інформація про будь-який реальний чи абстрактний об'єкт розглядається в сукупності, як одна структурна одиниця, над якою визначено множину дій. Наприклад, дія «Розігрів» над об'єктом «Інгридієнт» змінює дані про «Температуру», «Смак» або «Колір» останнього.

Таким чином, можна зробити висновок, що процес програмування прикладних програмних систем на основі парадигм ІП та ООП є набагато простішим та ефективнішим, оскільки він здається «дуже логічним» з точки зору людини, і при цьому наближений до принципів виконання програм самим комп'ютером. Це все і обумовлює той факт, що ці парадигми застосовні до більшості мов програмування. Але чи достатньо ефективною та зручною є розробка штучного інтелекту за принципами, заснованими на цих парадигмах?

Як відомо, логіка роботи електронно-обчислювальних пристроїв (що обумовлено фізичними аспектами їх роботи), а також більшості мов програмування, є булевою, тобто будь-яке твердження у них представлене у вигляді «істини» (True) або «хибності» (False). Цей принцип можна без проблем застосовувати у математиці, де потрібне чітке визначення результату, чітке визначення істинності. Наприклад, можна однозначно описати істинність твердження «число X є парним», описавши алгоритм вирішення такої задачі в імперативному стилі наступним чином: якщо залишок від ділення X на 2 дорівнює 0, то твердженню присвоюється «істина», в іншому випадку – «хибність». Все, що треба знати програмі для вирішення такої задачі, – це саме число X. Але у реальному житті «природньому» інтелекту доводиться виконувати набагато складніші завдання, для яких буває дуже складно або майже неможливо описати чітку логіку рішення, наприклад:

Чи є підсудний «X» винним?

Чи посміхається людина на фотографії?

Чи належить силует, який ми бачимо перед собою, небезпечному хижаку?

Перевірка таких тверджень може відбуватись при дуже різних наборах вхідних даних, які інколи взагалі не дають можливості знайти однозначний результат. Причому, при описі чіткого математичного алгоритму доводиться враховувати і кожну варіацію, і діапазон даних, що навіть у випадку достатньо простих задач може змусити нас збільшити об'єм вихідного коду в десятки, сотні і навіть тисячі разів. Тому, замість того, щоб розробляти рішення таких складних завдань на основі алгоритмічної логіки, їх часто проектують на основі принципів роботи біологічного розуму, а саме – через використання нейронних мереж та машинного навчання. Завдяки прогресу у швидкодії та можливостях комп'ютерних комплектуючих, а

також наявності у більшості популярних мов на основі ООП та ІП спеціалізованих бібліотек, такий спосіб розробки ШІ застосовується все частіше. Але чи існують парадигми, які усувають недоліки попередніх при розробці штучного інтелекту?

Розглянемо парадигму логічного програмування на основі мови Prolog. Prolog – це мова програмування, зосереджена навколо невеликого набору основних механізмів, включаючи зіставлення зі зразком, деревоподібне представлення структур даних та автоматичний перебір з поверненнями. Цей обмежений набір засобів утворює дуже потужне та гнучке середовище програмування. Prolog особливо добре підходить для вирішення завдань, в яких розглядаються об'єкти (зокрема, структуровані об'єкти) та відношення між ними. Зокрема, засобами мови Prolog зовсім не складно виразити просторові зв'язки між об'єктами, наприклад, вказати, що синя куля знаходиться за зеленою. Настільки ж просто можна визначити загальніше правило: якщо об'єкт X знаходиться ближче до спостерігача, ніж об'єкт Y, а Y – ближче, ніж Z, то X має бути ближче, ніж Z. Після цього система Prolog отримує можливість формувати міркування про просторові зв'язки та їх сумісність із загальним правилом. Завдяки таким особливостям Prolog стає потужною мовою для розробки штучного інтелекту та нечислового програмування в цілому [2].

Таким чином, логічну парадигму у мові Prolog можна назвати сильною стороною при розробці систем штучного інтелекту. Вона дозволяє розглядати програмування як опис пошуку логічного висновку на основі існуючих фактів. При цьому програмний код не повинен однозначно характеризувати шляхи до пошуку рішення за принципами булевої логіки, де будь-яке твердження визначається лише як «істина» або «хибність». В цьому і полягає робота рішень на основі штучного інтелекту, таких як експертні системи, обробка та аналіз мовлення, доведення теорем, трасування та евристичні оцінки, оскільки усі перераховані задачі здебільшого засновані на аналізі фактів, пошуку зв'язків між ними та синтез висновку. Мова Prolog дозволяє вирішувати такі задачі набагато швидше, ніж інші мови, оскільки логічна парадигма, яку вона підтримує, і є концепцією, призначеною для їх рішення (наприклад, перший чатбот ELIZA був розроблений з використанням Prolog для обробки конструкцій мовлення [3]).

З іншого боку, концепція логічного програмування в Prolog є набагато складнішою для розуміння і освоєння в порівнянні з парадигмами функціонального, імперативного чи об'єктно-орієнтованого програмування, що кардинально відрізняє цю мову від більшості популярних мов програмування. Тому її головна перевага, що полягає у написанні програмного коду як «дерева досягнення висновку» (а не «алгоритму» у випадку імперативної парадигми), значно ускладнює застосування у прикладному програмуванні, коли кожна програма розглядається як покроковий набір інструкцій. Це робить Prolog малопоширеною мовою програмування у широкому колі розробників, тому ця мова здебільшого застосовується в академічних або дослідницьких цілях.

Таким чином, логічна парадигма, яка хоч і забезпечує потужні механізми розробки ШІ, але демонструє свої суттєві концептуальні недоліки при розширенні спектру та специфіки завдань, які потрібно вирішувати. Але що, якщо якась інша парадигма може об'єднати принципи нечіткої логіки з логічної парадигми, і при цьому дозволить мові програмування набагато ширше використовуватись як прикладний інструмент? Виходом може стати функціональна парадигма програмування.

Функціональна парадигма довгий час трималась осторонь сфери прикладної розробки. Як і у випадку з логічним програмуванням, імперативний спосіб написання коду випереджає її як більш зрозумілий та наближений до людського сприйняття принцип опису алгоритму виконання завдань. А оскільки протягом довгого часу більшість мов функціонального програмування виконувались інтерпреторами і, отже, були достатньо повільними, то й не дивно, що їх імперативні та об'єктно-орієнтовані компільовані аналоги, такі як C, C++ або Java, завоювали значно більшу популярність серед розробників.

Однак, завдяки розвитку комп'ютерної техніки, можна говорити про нівелювання вказаного недоліку, оскільки сьогодні, наприклад, компілятор GHC (Glasgow Haskell Compiler) для мови Haskell створює виконуваний код, який не поступається за ефективністю програмам, розробленими засобами мов C чи C++ [4].

У зв'язку зі стрімким зростанням машинного навчання і великих даних ФП стало набирати популярність через простоту розпаралелювання чистих функцій. Функціональна парадигма також спрощує відстеження, тестування та обслуговування коду для задач аналізу даних та робочих процесів, що створює передумови для її активного використання у найближчому майбутньому.

Однією з найпоширеніших функціональних мов є Haskell. Це винятково функціональна мова, що базується на лямбда-численні. Вона не є надто популярною, але широко використовується як в дослідженнях, так і в реалізації комерційних проектів.

У мові Haskell, як і в більшості мовах функціонального програмування, не визначається чіткий порядок виконання обчислень, – лише декларуються залежності між даними, а порядок виконання визначається транслятором. Такий підхід дуже схожий на концепцію логічного програмування в Prolog. Більше того, Haskell, так само як і Prolog, допускає програмування нечіткої логіки і нечіткої математики, формування логічного висновку. Опис роботи з такими поняттями на основі функціональної парадигми

наведений у [4]. Автор називає парадигму ФП «найбільш пристосованою для вирішення завдань штучного інтелекту».

З усвідомленням переваг функціональної парадигми над імперативним стилем чимало мов програмування були розширені функціональними елементами. Наприклад, у мові Python з'явились такі оператори з Haskell, як map, filter, lambda, reduce. У мові C# з'явились LinQ та лямбда-вирази. Тому популярні (здебільшого імперативні та об'єктно-орієнтовані мови) розвиваються в бік мультипарадигмового програмування, об'єднуючи простоту імперативного стилю написання коду з декларативністю функціонального програмування. Оскільки парадигми програмування утворюють висхідну лінію концептуальності, то й не дивно, що більшість сучасних мов програмування є, по суті, мультипарадигмовими (табл. 1).

Таблиця 1

Мультипарадигмовість мов програмування

Мови	Парадигми			
	Функціональна	Логічна	Імперативна (процедурна)	Об'єктно-орієнтована
Ада	–	–	+	+
C	–	–	+	–
C++		–	+	+
C#		–	+	+
Java	–	–	+	+
Haskell	+		+	–
Common LISP	+		+	+
Python		–	+	+
Prolog		+	–	–
SmallTalk	+		+	+
JavaScript	–	–	+	+
F#	+	–	+	+

Висновки. У дослідженні виконано порівняльний аналіз різних парадигм та мов програмування, що підтримують ці парадигми, на предмет їх ефективності та зручності при розробці штучного інтелекту. Проведена оцінка зручності їх використання, аналіз сфер їх поширеності на ринку розробки ПЗ та їх ефективність. Встановлено, що жодна парадигма не вирішує завдань розробки ШІ найлегшим чи найефективнішим способом. Так само не існує ідеальної мови для ШІ – кожна має свої переваги та недоліки при вирішенні певних задач із різних предметних областей. Тому більшість відповідних рішень мають базуватись на комбінуванні (поєднанні) декількох технологій та парадигм, які включають різні варіанти для реалізації бажаної функціональності та досягнення високої ефективності.

References

1. Van Roy P. Programming Paradigms for Dummies: What Every Programmer Should Know [Online] / P. Van Roy // ResearchGate. – Available: https://www.researchgate.net/publication/241111987_Programming_Paradigms_for_Dummies_What_Every_Programmer_Should_/Know

2. Bratko I. Algoritmyi iskusstvennogo intellekta na yazyike PROLOG; 3-e izdanie: Per. s angl. / I. Bratko. – M. : Izdatelskiy dom "Vilyamc", 2004. – 640 s.

3. Weizenbaum J. ELIZA – A Computer Program For the Study of Natural Language Communication Between Man And Machine / J. Weizenbaum // Communications of the ACM. – 1966. – Volume 9(1). – P. 36–45.

4. Dushkin R. V. Funktsionalnoe programmirovanie na yazyike Haskell / R. V. Dushkin. – Izd-vo «DMK Press», 2008. – 609 s.