Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційних радіотехнологій і технічного захисту інформації
(повна назва)
Кафедра Радіотехнологій інформаційно-комунікаційних систем
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)
ГЮІК.ХХХХХХ.001ПЗ
(позначення документа)
Дослідження використання штучного інтелекту у кібер просторі
(тема)

Виконав:
студент II курсу, групи ІКТм-20-1
Юрченко О.О.
(прізвище, ініціали)
Спеціальність
122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми
освітньо-професійна
(освітньо-професійна фбо освітньо-наукова)
Освітня програма
Інформаційно-комунікаційні технології
(повна назва освітньої програми)
Керівник проф. Кузьомін О. Я.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ О.І. Цопа
(підпис) (прізвище, ініціали)

2021 р.

Не містить відомостей заборонених для відкритого публікування.

Керівник _____ Кузьомін О.Я.

Студент _____ Юрченко О.О.

Харківський національний університет радіоелектроніки

Факультет Інформаційних радіотехнологій і технічного захисту інформації
Кафедра Радіотехнологій інформаційно-комунікаційних систем
Рівень вищої освіти другий (магістерський)
Спеціальність 122 Комп'ютерні науки
(код і повна назва)
Освітня програма Інформаційно-комунікаційні технології
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)
"_____" _____ 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Юрченко Олександру Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи **Дослідження використання штучного інтелекту у кібер просторі**
затверджена наказом по університету від **05 листопада 2021 р. № 1648 Ст**

2. Термін подання студентом роботи до екзаменаційної комісії **9 грудня 2021 р.**

3. Вихідні дані до роботи розробити штучний інтелект, що буде проводити дослідження різноманітних мутацій коронавірусної хвороби

4. Перелік питань, що потрібно опрацювати в роботі аналіз предметної області, постановка задачі, аналіз існуючих алгоритмів розв'язання задачі, вибір алгоритму розв'язання задачі, розробка структури додатку, розробка алгоритму функціонування, розробка алгоритму достлідження мутацій.

5. Перелік графічного матеріалу із зазначенням комп'ютерного коду, схем, комп'ютерних ілюстрацій (слайдів)

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основна частина	проф. Кузьомін О.Я.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	*Ознайомлення із завданням. Уточнення ТЗ*	01.09.21	вик.
2	*Підбір літератури за темою роботи*	13.09-22.09.21	вик.
3	*Теоретичний розділ*	23.09-10.10.21	вик.
4	*Проектний розділ*	11.10-25.10.21	вик.
5	*Оформлення презентаційного матеріалу,*	26.10-30.11.21	вик.
	підготовка до захисту у ЕК		

Дата видачі завдання __1__ ___вересня___ 2021 р.

Студент _____ Юрченко О.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ проф. Кузьомін О.Я.
(підпис) (посада, прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка: 89 сторінок, 13 малюнків, 2 таблиць, 27 джерел.

Ключові слова: *ШТУЧНИЙ ІНТЕЛЕКТ, ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ, КІБЕР ПРОСТІР, НЕЙРОННІ МЕРЕЖІ, МАШИННЕ НАВЧАННЯ*

Актуальність роботи пояснюється тим, що у сучасному інтернеті, користувач без досвіду може легко стати жертвою шахраїв, які розповсюджують віруси у мережі. Штучний інтелект, у свою чергу, постійно розвивається і вдосконалюється, і вже зараз може допомогти у вирішенні проблем безпеки у інтернеті.

Метою роботи є сучасне теоретичне дослідження та аналіз існуючих систем з використанням штучного інтелекту. Аналіз переваг та недоліків їх застосування, у тому числі у кіберпросторі, та пошук шляхів підвищення ефективності існуючих систем. Розробка додатку для виявлення потенційних загроз у тексті.

Об'єкт дослідження - кіберпростір, безпека кіберпростору.

Предмет дослідження - методи виявлення та боротьби з загрозами кіберпростору.

Перший розділ присвячено аналізу обробки даних та принципам функціонування систем.

Другий розділ розкриває загальну інформацію та алгоритми вирішення складних завдань штучного інтелекту.

У третьому розділі розглядається використання штучного інтелекту у кібер просторі: перспективи та напрямки застосування систем штучного інтелекту.

ABSTRACT

Explanatory note: 89 pages, 13 figures, 2 tables, 27 sources.

Keywords: *ARTIFICIAL INTELLIGENCE, INTELLECTUAL SYSTEMS, CYBER SPACE, NEURAL NETWORKS, MACHINE LEARNING*.

Relevance of the work is explained by the fact that in today's Internet, a user without experience can easily become a victim of hackers, who spread viruses in the network. Intellect, for its part, is constantly evolving and improving, and even now can help in solving security problems on the Internet.

The aim of the work is modern theoretical research and analysis of existing systems using piecemeal intellekt. Analysis of advantages and disadvantages of their use, including in the cyber space, and the search for ways to improve the efficiency of existing systems. Development of an addendum for detection of potential threats in the text.

The object of research - Cyber space, safety of the Cyber space.

The subject of the research - methods of detection and combating threats to the cyber space.

The first section is devoted to the analysis of data processing and the principles of system operation.

The second section reveals general information and algorithms for solving complex problems of artificial intelligence.

The third section considers the use of artificial intelligence in cyberspace: prospects and directions of application of artificial intelligence systems.

ЗМІСТ

ВСТУП

Поняття штучного інтелекту, як і інтелекту, є досить розпливчастим. Узагальнюючи все, що було сказано за останні тридцять років, виходить, що людина хоче лише створити своє власне, як у тій чи іншій формі, хоче, щоб деякі дії виконувались раціональніше, з меншими витратами часу та енергії.

Починаючи з кінця 1940-х років, вчені у дедалі більшій кількості університетських та промислових дослідницьких лабораторій прагнули досягти амбітної мети: Побудувати комп'ютери, які працюватимуть таким чином, щоб вони були невідмінні від людського розуму в результатах своєї роботи. Останнім часом зростає інтерес до штучного інтелекту, викликаний зростаючими вимогами до інформаційних систем.[1]

Програмне забезпечення стає розумнішим, побутова техніка — розумнішою. Ми невблаганно рухаємося до нової інформаційної революції, порівнянної за масштабами з розвитком інтернету, що називається штучним інтелектом.

Штучний інтелект є гарячою точкою дослідження. Саме тут зосереджено найбільші зусилля кібернетистів, лінгвістів, психологів, філософів, математиків та інженерів. Саме тут вирішуються багато фундаментальних питань щодо шляху розвитку наукової думки, про вплив досягнень комп'ютерної науки та робототехніки, про вплив на майбутні покоління.

Саме тут з'являються нові методи наукових міждисциплінарних досліджень і знаходять громадянську позицію. Саме тут формується новий погляд на роль того чи іншого наукового результату і виникає щось на кшталт філософського осмислення цих результатів. Тому я вважав за доречне розкрити цю тему в абстрактній манері.

Терпляче просуваючись уперед у своїй нелегкій роботі, дослідники, які працюють у галузі штучного інтелекту (II), виявили, що вони заглибилися в дуже заплутані проблеми [1], які виходять далеко за межі традиційної комп'ютерної

науки. Виявилося, що спочатку необхідно зрозуміти механізми навчального процесу, природу мови та сенсорного сприйняття.

Виявилося, що для того, щоб створити машини, що імітують людський мозок, необхідно зрозуміти, як працюють мільярди його взаємозалежних нейронів. І тоді багато дослідників дійшли висновку, що, мабуть, найважча проблема, що стоїть перед сучасною наукою, полягає в тому, щоб зрозуміти, як працює людський розум, а не просто наслідувати його роботу. Що безпосередньо вплинуло фундаментальні теоретичні проблеми психологічної науки.

Насправді вченим важко навіть дійти єдиної точки зору щодо самого об'єкта своїх досліджень — інтелекту. Одні вважають, що інтелект – це здатність вирішувати складні проблеми; інші розглядають його як здатність вчитися, узагальнювати та аналогізувати; треті розглядають його як здатність взаємодіяти із зовнішнім світом через спілкування, сприйняття та усвідомлення того, що сприймається.

Тим не менш, багато дослідників II схильні прийняти тест машинного інтелекту, запропонований на початку 1950-х років відомим англійським математиком та вченим у галузі обчислювальної техніки Аланом Т'юрінгом. "Комп'ютер можна вважати розумним, - стверджував Т'юрінг, - якщо він може змусити нас повірити, що ми маємо справу не з машиною, а з людиною" [2].

ТЕОРЕТИЧНІ ЗАСАДИ ПО СТВОРЕННЮ ТА ЗАСТОСУВАННЮ ШТУЧНОГО ІНТЕЛЕКТУ

1.1 Історія штучного інтелекту

Початок сучасного етапу розвитку систем штучного інтелекту (ІІ) може бути датований серединою 1950-х років. Він був розроблений А. Ньюеллом для доказу теорем у пропозиційному обчисленні та називався Логічно-Теретичним. Деякі автори називають цю систему експертною та простежують визначення її призначення до аналізу її можливостей, проведеного Клодом Шенноном та Марвіном Мінським.

Ця робота започаткувала дослідження в галузі ІІ, які стосуються розробки програм, що вирішують проблеми шляхом застосування різних евристичних методів та правил.

Евристика - це набір логічних методів та методологічних правил для теоретичних досліджень та пошуку істини, метод пошуку доказів.

Евристичні правила — неформальні правила, використовувані підвищення ефективності пошуку у певної предметної області.

Цей метод вирішення проблеми вважався характерним для міркувань людини в цілому, який характеризується тим, що робить «обґрунтовані здогади» про те, як вирішити проблему, а потім їх тестує. Евристичний метод контрастує з алгоритмічним (процедурним, процесуальним) методом, який використовується в комп'ютерах, який інтерпретувався як механічна обробка зумовленої послідовності кроків, що приводить детерміновано до правильної відповіді. Така інтерпретація евристичних методів вирішення проблем визначила появу та поширення терміну «штучний інтелект». [3]

У 70-80-ті роки дослідження в галузі ІІ характеризувалися тим, що увага фахівців перемістилася з проблем створення автономно функціонуючих систем на створення людино-машинних систем, що об'єднують людський інтелект та

комп'ютерні можливості для досягнення спільної мети – вирішення поставленої перед такою системою завдання.

Багато хто думав, що це дозволить створити новий напрямок в інформаційних технологіях — машинобудування, яке прийде на зміну роботі фахівця. Проте з низки причин ці очікування були реалізовані повною мірою.

Тим не менш, в останнє десятиліття цей напрямок було відроджено у вигляді науково-дослідних та дослідно-конструкторських робіт, спрямованих на створення експертних систем з базою знань. Вони використовуються в управлінській діяльності та багатьох галузях економіки (страхування, банківська справа тощо) для підвищення якості рішень з використанням правил і об'єктів, що узагальнюють накопичений досвід [4].

Проблема штучного осіменіння нині досить велика. Список дисциплін у штучному інтелекті стає дедалі довшим. Сьогодні це представлення знань, вирішення завдань, експертні системи, засоби спілкування з комп'ютером природною мовою, навчання, когнітивне моделювання, обробка візуальної інформації, робототехніка, нейрокомп'ютерні технології та ін.

Подання знань є найважливішою галуззю досліджень у галузі штучного інтелекту, основою всіх інших дисциплін. Знання набувають форми опису об'єктів, взаємозв'язків та процесів. Наявність адекватних знань та здатність їх ефективно використовувати означає «навичка».

Створення загальної теорії чи методу представлення знань є стратегічним завданням. Така теорія відкрила б можливість накопичення знань, які необхідні щодня для вирішення нових і нових завдань. Однак для досягнення цієї мети необхідно знайти спосіб висловити загальні закономірності предметних областей (SD), що є ядром проблеми знань.

Розв'язання задач - це знаходження шляху від початкової точки до мети. Люди роблять це дуже ефективно за допомогою дедуктивного міркування (аргументації), процедурного аналізу, аналогії та індукції. Люди можуть навчатися на власному досвіді. Комп'ютери, як правило, вирішують проблеми лише за допомогою дедуктивних міркувань та процедурного аналізу.

Тип завдання визначає метод, що найбільш підходить для її вирішення. Завдання, зведені до процедурного аналізу, зазвичай, найкраще вирішувати на комп'ютері. Проблеми обліку та аналізу є прикладами процедурних проблем, які можуть бути вирішені комп'ютерами швидше та надійніше, ніж людьми. Навпаки, завдання, які потребують використання аналогії чи індукції, ефективніше вирішуються людиною. Завдання, що вимагають дедуктивного та індуктивного міркування, швидше за все, вирішуватимуться за допомогою експертних систем (систем, заснованих на знаннях).

Експертні системи - це клас комп'ютерних програм, які надають консультації, проводять аналіз, класифікують, консультують та ставлять діагнози. Вони зосереджуються на завданнях, які зазвичай потребують людського досвіду. На відміну від програм, які використовують процедурний аналіз, експертні системи вирішують завдання у вузькій предметній галузі (конкретній галузі знань) на основі логічного міркування. Такі системи часто здатні знаходити рішення неструктурованих та погано певних проблем. Вони справляються з відсутністю структури, використовуючи евристику, яка може бути корисною у ситуаціях, коли відсутність необхідних знань чи часу не дозволяє провести повний аналіз.

Машини мають власну мову уявлення знань та вирішення проблем, тобто. набір символів для представлення знань (семантика) та правила обробки цих символів (синтаксис) та вирішення проблем [4]. Людина працює найбільш ефективно, коли вона опановує конкретні мови, які розвиваються відповідно до потреб конкретної галузі.

Якщо правила перекладу з природної мови на машинну і навпаки виражені у вигляді сукупності знань (символів та процедур), логічно припустити, що можна розробити засоби, що дозволяють комп'ютеру зрозуміти проблему природною мовою, а потім вивести рішення природною мовою. Це основний предмет досліджень з розробки засобів спілкування з комп'ютером природною мовою. Тут можна виділити чотири основні проблеми:

Машинний переклад — Використовуйте комп'ютери для перекладу тексту з однієї мови на іншу.

Пошук інформації — використання комп'ютерів для доступу до інформації з конкретного предмета, що зберігається у великій базі даних.

Виробництво документів — використання комп'ютерів для перетворення документів, які мають певну форму або вказані певною мовою, у відповідний документ в іншій формі або іншою мовою.

Комп'ютерна взаємодія – організація діалогу між користувачем та комп'ютером.

Вважається, що можливості навчання повинні бути присутніми майже в кожній прикладній програмі, якої потребує користувач. П'ятнадцять або двадцять років тому більшість обробки даних, пов'язаної з вирішенням завдань, виконувалася програмістами. Насправді вони діяли як посередники, ніби вони були сполучною ланкою між комп'ютером і тими, хто використовував отримані дані та приймав рішення. З появою персональних комп'ютерів різко змінилися відносини між користувачем та обчислювальними машинами, а також роль програмістів [4].

Замість того, щоб змушувати користувача засвоювати складність програмування, легко навчити комп'ютер складності конкретної задачі. Звичайно, це не означає, що потреба в програмістах зникне, але дещо змінює їхню роль у взаєминах між комп'ютером та кінцевим користувачем.

Метою когнітивного моделювання є розробка теорій, концепцій та моделей людського мислення та його функцій. Вона дозволяє як реалізувати діагностичні і терапевтичні функції, а й виявити процеси, які у свідомості людини під час вирішення поставлених завдань.

Однак з цього ні в якому разі не випливає, що найкращі комп'ютери — це ті, які відтворюють роботу людського мозку, але можна припустити, які саме комп'ютери потрібні, як спроектувати комп'ютер, який розширить можливості людської думки та дозволить їй ефективно вирішувати завдання.

Сучасні роботи вже полегшили роботу багатьох виробничих робітників (особливо некваліфікованих), бездоганно виконуючи свої завдання. Дослідження в галузі робототехніки є невід'ємною частиною досліджень

штучного інтелекту, метою яких є оснащення комп'ютерів засобами візуальної обробки та маніпулювання об'єктами у заданому середовищі.

Це дослідження проводиться за трьома основними напрямками:

– розробка елементів сприйняття (особливо візуальної інформації) та розпізнавання інформації, що надходить від систем сприйняття;

– розробка маніпуляторів та систем управління ними;

– ідентифікація евристики для вирішення завдань руху у просторі та маніпулювання об'єктами (планування діяльності) [5].

Аналіз розробок у галузі нейрокомп'ютерних систем дозволив виявити перспективні ключові напрямки сучасних нейрокомп'ютерних технологій: Нейро-пакети, експертні системи з нейронними мережами, бази даних та системи управління базою знань, що містять нейромережні алгоритми, обробку зображень, динамічні системи, контроль та обробку сигналів, фінансовий менеджмент, оптичні нейрокомп'ютери, віртуальну реальність.

1.2 Теоретичні засади

Метою цієї роботи є розробки програми, яка використовує штучний інтелект для розпізнання написаного від руки тексту. Для розробки програм ми будемо використовувати мову програмування C++.

C++ - компільоване програмування мови загального призначення, поєднує властивості як високо-рівневі, так і низько-рівневі мови програмування. У порівнянні з його попередником - мовою програмування C, - найбільше увагу надає підтримку об'єктно -орієнтованого та узагальненого програмування. [9,10]

Назва «Мова програмування C++» відбувається від мови програмування C, в якому універсальний оператор ++ визначає інкремент змінної. Мова програмування C++ широко використовується для розробки програмного забезпечення. А саме створення різноманітних прикладних програм, розробка операційної системи, драйверів пристроїв, а також відеоігр та багато іншого.

Існує кілька реалізованих мов програмування C++ - як безкоштовних, так і комерційних. Їх виробничий Проект GNU, Microsoft та Embarcadero (Borland). Проект GNU - проект розробки вільного програмного забезпечення (СПО). Мова програмування C++ був створений на початку 1980-х років, співробітником фірми Bell Laboratories - Бьёрн Страуструп.[11] Він придумав ряд удосконалений до мови програмування C, для власних потреб. Тобто не планувалося створення мови програмування C++. Ранні версії мови C++, відомі під іменем "C з класами", почали проявлятися з 1980 року. Язик C, будучи базовою мовою системи UNIX, на якій працювали комп'ютери фірми Bell, є швидким, багатофункціональним і переносним. Страуструп додав до нього можливості роботи з класами та об'єктами, тим самим зародив переваги нового, на основі синтаксису C, мови програмування.

Синтаксис C++ був заснований на синтаксисі C, так як Бьорн Страуструп прагнув зберегти сумісність з мовою C. В 1983 році вийшло переіменування мови з "C з класами" у "програмування мови C++". В нього були додані нові можливості: віртуальні функції, перевантаження функцій та операторів, посилання, константи та багато іншого. Його перший комерційний випуск відбувся в жовтні 1985 року. Язик програмування C++ є вільним, тому ніхто не має на його правах.[6]

1.2.1 Опис предметного середовища

Після проведеного огляду програмних засобів ми вибрали середу програмування найбільш підходящу нам для розробки даного програмного продукту. Microsoft Visual Studio 2019 є найбільш вигідною для нас середовищем програмування.

Visual Studio - це стандартне інтегроване середовище розробки (IDE), яке розроблене Microsoft для звичайної розробки графічного інтерфейсу користувача, веб-програм, консолімобільних додатків, веб-додатків, хмарних та веб-служб тощо. За допомогою, зазвичай, цієї IDE можна створювати керований власний код. Зазвичай він використовує різні платформи програмного

забезпечення, однак, Microsoft для розробки програмного забезпечення, такі як Магазин Windows, Windows API, Microsoft Silverlight тощо. Це не є мовна IDE, оскільки ви певно можете використовувати це для написання коду на C#, C ++, VB (Visual Basic), Python, JavaScript та багато інших мов. Він забезпечує підтримку 36 різних мов програмування. Він доступний на таких платформах, як для Windows, так і для macOS.[7]

Еволюція Visual Studio: У 1997 році перша версія VS (Visual Studio) була випущена під назвою Visual Studio 97 з версією 5.0. Остання була випущена 7 березня 2017 року версія Visual Studio — 15.0, яка. Вона також називається Visual Studio 2017.

В ньому є наступні частини:

– редактор коду: користувач пише код;

– вікно виводу: тут Visual Studio показує попередження компілятора, результати, повідомлення про помилки та інформацію про налагодження;

– провідник рішень - він показує файли, над якими зараз працює користувач.

Властивості: Він надає додаткову інформацію про вибрані частини та контекст поточного проекту.

1.2.2 Опис процесу діяльності

Штучний інтелект сьогодні використовується у світі. Комп'ютери та інші обчислювальні машини вже давно є вірними помічниками людини, завжди готовими прийти на допомогу у вирішенні різноманітних завдань. У багатьох випадках ЕОМ забезпечує вищу швидкість розв'язання завдань і більш високу обчислювальну точність, ніж якби людина вирішувала ці завдання самостійно.[8]

Сьогодні штучний інтелект є невід'ємною частиною цивілізації та використовується у всіх сферах людської діяльності. Приклади варіюються від використання мікрокалькуляторів до використання роботів як лікарі, нянь, прибиральники (все частіше використовується в Японії).

У своїй недовгої історії дослідники штучного інтелекту завжди були в авангарді комп'ютерної науки. Багато поширених сьогодні розробки, включаючи передові системи програмування, текстові редактори та програмне забезпечення для розпізнавання образів, значною мірою ґрунтуються на досягненнях у дослідженнях штучного інтелекту.

Теорії штучного інтелекту, нові ідеї та розробки постійно привертають увагу тих, хто прагне розширити сфери застосування та можливості комп'ютерів, зробити їх «дружнішими», тобто більше схожими на розумних помічників та активних радників, ніж на допитливих і безжальних електронних рабів, якими вони завжди були. І поки ці ідеї не будуть реалізовані, комп'ютери ніколи не зможуть замінити людей, хоч би якими просунутими вони були.

Незважаючи на перспективні перспективи, жодну з розроблених досі програм штучного інтелекту не можна назвати «розумною» у звичайному розумінні цього слова. Це пояснюється тим, що вони є вузькоспеціалізованими, а найскладніші експертні системи нагадують приручені чи механічні ляльки, а не людину з її гнучким розумом та широкими обріями. Навіть серед дослідників ІІ багато хто зараз сумнівається, що більшість продуктів будуть корисними. Багато критиків штучного інтелекту вважають, що такі обмеження є абсолютно непереборними.[9]

Серед цих скептиків є Х'юберт Дрейфус, професор філології Каліфорнійського університету в Берклі. На його думку, справжній інтелект не може бути відокремлений від його людської основи, яка міститься у людському тілі. «Цифровий комп'ютер — це не людина, — каже Дрейфус, — комп'ютер не має ні тіла, ні емоцій, ні потреб». У ньому відсутня соціальна орієнтація, набута в результаті життя в суспільстві, яка робить поведінку розумною. Я не говорю, що комп'ютери не можуть бути розумними. Але цифрові комп'ютери, запрограмовані за допомогою фактів і правил нашого, людського життя, не можуть стати по-справжньому інтелектуальними. Тому штучний інтелект, як ми його уявляємо, є неможливим».

При всій своїй універсальності комп'ютер не може допомогти вирішити проблему, яку важко вирішити. Складне (нерозв'язне) завдання — це завдання,

на яку не відомий ефективний алгоритм швидкого рішення, або для якої взагалі не існує алгоритму рішення.[10] Ефективний алгоритм має залежність кількості обчислень від вхідних даних, що не збільшується. Такі алгоритми називаються багаточленами, і зазвичай, якщо завдання має алгоритм багаточленного рішення, її не можна вирішити на простому комп'ютері з низькою ефективністю. З ними можуть бути пов'язані проблеми сортування даних, багато математичних проблем програмування тощо.

Що не може зробити комп'ютер у сучасному (цифровому) значенні, і, мабуть, ніколи не зможе? Відповідь очевидна: виконати рішення повністю аналітично. Завдання полягає в тому, щоб замінити аналітичне рішення числовим алгоритмом, який рекурсивно виконує операції, що крок за кроком наближаються до рішення. Зі збільшенням кількості таких операцій збільшується час виконання, і, можливо, споживання інших ресурсів (наприклад, обмеженої пам'яті машини) має тенденцію до нескінченності. За допомогою алгоритмів їх вирішення задачі, які різко підвищують споживання ресурсів, не можуть бути вирішені на цифрових комп'ютерах, оскільки ресурси завжди обмежені.

В даний час наявність надвисокопродуктивних мікропроцесорів та дешевизна електронних компонентів дозволяють значно просунутися в алгоритмічному моделюванні штучного інтелекту. Такий підхід дає деякі результати на цифрових комп'ютерах загального призначення і полягає у моделюванні процесів життя та мислення з використанням чисельних алгоритмів, що реалізують штучний інтелект.

Тут можна навести безліч прикладів, починаючи від простої програми іграшки «Тамагочі» до моделей колонії живих організмів та шахових програм, здатних перемогти знаменитих гросмейстерів. Сьогодні такий підхід підтримується практично всіма великими розробниками апаратного та програмного забезпечення, оскільки досягнення у створенні алгоритмів, що реалізують штучний інтелект, також використовуються у вузькоспеціалізованих прикладних областях при вирішенні складних завдань і приносять розробникам знання.

1.2.3 Алгоритмічний та декларативний підходи до управління

На загальному рівні існує два підходи до управління складними системами та до програмування роботів та комп'ютерів, які можуть вирішувати певні завдання. У програмуванні сучасних комп'ютерів переважно реалізований традиційний алгоритмічний підхід, який також можна назвати імперативним. Такий підхід вимагає заздалегідь продумати та докладно описати, як вирішити проблему. Написання програми потребує чіткої послідовності вказівок. Якщо ви зможете написати таку послідовність - комп'ютер зможе вирішити це завдання, хоч би яким складним воно було. Але, звичайно, він дотримувався інструкцій, не розуміючи їхнього змісту. Якщо умови, в яких програма працює, зміняться, комп'ютер може бути безпорадним.[11]

Інший підхід – декларативний. Інтелектуальному виконавцю (людині чи комп'ютеру) достатньо робити те, що потрібно, тобто лише формулювати завдання, вибудовуючи всі взаємозв'язки між об'єктами у предметній галузі. Як буде виконано це завдання – має визначитися виконавець.

Запуск космічного корабля так, щоб він приземлився на Марсі, брав зразки ґрунту та повертав їх, - дуже складне завдання, але його можна точно алгоритмізувати.

Математичні методи дозволяють точно розрахувати траєкторію, якою має рухатися такий корабель.

Відправити роботу до магазину за пляшкою молока – завдання на кілька порядків складніше. Не говорячи вже про процес спілкування з продавцем, робот повинен вирішити низку не зовсім формалізованих підзавдань. Заздалегідь розрахувати траєкторію неможливо, оскільки робот повинен уникати зіткнень із людьми та машинами.

Якщо магазин закрито на перереєстрацію, він має знайти інший магазин. Неможливо спрогнозувати всі ситуації, які можуть виникнути, а отже – алгоритмізувати вирішення проблеми. Отже, таке завдання може виконати лише інтелектуальна система, яка може орієнтуватися у зовнішньому світі, аналізувати

поточні ситуації та коригувати, адаптувати свою поведінку на основі такого аналізу.[21]

1.2.4 Формалізація понять алгоритмічності та декларативності.

Спробуємо формалізувати деякі з раніше реалізованих концепцій. Введемо такі позначення:

– q - первинні інструкції, написані без змін у тому вигляді, у якому вони сформульовані автором; аналогічно ви можете визначити вихідне опис ситуації як результат її прямого сприйняття;

– S – безліч факторів, що визначають поточний стан виконавців; ми розглядатимемо S як комбінацію двох наборів: S_1 і S_2, де S_1 - набір контрольованих факторів, які відомі авторським процедурам і на які він може впливати, S_2 - набір неконтрольованих факторів;

– Z – знання, якими володіє виконавець;

– r_A - робочий алгоритм, який формується та реалізується виконавцем в алгоритмічному підході. В цьому випадку, як правило, автоматично формулюється програма pA, що відповідає даному алгоритму;

– r_D - робочий алгоритм, який формується та реалізується інтелектуальним виконавцем з декларативним підходом.

Однак цей алгоритм може не бути реалізований виконавцем і, отже, не може бути сформульований як алгоритм або програма (якщо повинна бути згенерована програма, відповідна алгоритму r_D, це проблема синтезу програми).

Тоді ви можете написати такі відносини:

$$r_A = f(q, S_1),$$
$$r_D = g(q, S_1, S_2, Z).$$

Наступне є фундаментальним. Алгоритм r_A однозначно формується з урахуванням первинних інструкцій q. При цьому у вихідну інструкцію також можуть вноситися зміни та доповнення, але цей процес повністю контролюється та детерміновано. Прикладом може бути компіляція програм, написаних мовами

високого рівня. Таким чином, автор процедури може бути впевненим у гарантованому результаті (якщо, звичайно, виконані певні формальні вимоги до вихідних інструкцій та належний стан виконавця забезпечено).

Навпаки, декларативний підхід не має такої визначеності.

Розумний виконавець чимось доповнює оригінальні інструкції, але їхній автор не завжди знає, як відбувається ця дозаправка. Отже, ви не можете бути впевнені в результаті дотримання інструкцій, і ця втрата гарантії є неминучою платою за відмову від алгоритмічної обробки. Таким чином, маємо справу з узагальненням поняття алгоритму, яке називається «квазіалгоритм».[12]

Алгоритм - це чітка, однозначна послідовність інструкцій, зрозуміла виконавцю, виконання якої обов'язково призводить до гарантованого результату кінцевого часу. Навпаки, інструкції квазіалгоритму можуть бути не дуже зрозумілими, і результат квазіалгоритмічної процедури не обов'язково гарантований.

Існує як мінімум чотири основні джерела квазіалгоритмічності:

Перший – це дія випадкових чинників, які від виконавця. Строго кажучи, з урахуванням цього фактора слід розглядати будь-який квазіалгоритм, навіть формалізований алгоритм. Алгоритм призначений для роботи у певних умовах; при зміні цих умов алгоритм може призвести до бажаного результату. Якщо неможливо чітко визначити межі застосування алгоритму або забезпечити виконання необхідних умов його роботи, маємо справу з реальним квазіалгоритмом.

Другий – недостатнє опрацювання автором алгоритмічної процедури виконавця. Це призводить до того, що виконавець неправильно розуміє, що від нього вимагається. Процедура в принципі може бути алгоритмічною, за нормальних умов призводити до гарантованого результату, але цей результат може бути передбачений автором і, отже, бути йому несподіванкою.

Третє – нечіткість формулювань в описі процедури. Розглянемо будь-який рецепт, наприклад: "Замісити тісто, змішати з дрібно нарізаними яблуками, посолити і поперчити до смаку і обсмажити до рум'яної скоринки". Це типовий приклад нечіткості формулювань, що призводить до невизначеності. Наскільки

потрібно замісити тісто? Що означає «дрібно нарізаний»? Що означає «за смаком» і т. д. неінтелектуальна система, орієнтована на суто алгоритмічне управління, просто не зрозуміє цього опису і, отже, не зможе його виконати. Інтелектуальна система, наприклад людина, має спробувати доповнити та уточнити цей опис на основі наявних знань та досвіду. Сам собою факт виконання завдання свідчить у тому, що квазиалгоритмическое початкове опис процедури було зведено до якогось внутрішнього алгоритму, але навряд чи виконавець зможе чітко пояснити і описати цей алгоритм.

З іншого боку, ці внутрішні алгоритми будуть різними кожному за виконавця.

Четвертим – можна визначити «Свободу волі», яка може мати високоорганізовані, справді інтелектуальні системи. Ці системи можуть мати свої власні цілі, і якщо ця процедура суперечить цим цілям, вони можуть відмовитися робити це або не працювати належним чином. "Свобода волі" полягає в тому, що інтелектуальна система приймає власні рішення про свою поведінку і, залежно від цих рішень, може виконувати зовнішні накази, а не виконувати їх або робити те, що вважає за потрібне.

Інтелектуальна система може включати зовнішнє управління, але для неї характерний самоконтроль. Система має конкретну мету та прагне спланувати свої дії для досягнення цієї мети. Поточна ситуація, яка сприймається та аналізована системою, може розглядатися як вхідні стимули системи. Результатом реакції системи є зміна зовнішньої ситуації, і поведінка системи регулюється залежно від того, бажано це зміна чи небажано.

Людина має певний обсяг знань про світ, що дозволяє йому орієнтуватися в життєвих ситуаціях і приймати правильні рішення. Крім того, людина вміє певним чином використати ці знання. Ці функції повинні мати системи штучного інтелекту.

Також можна стверджувати, що здатність поповнювати базові знання є однією з ключових особливостей інтелектуальних систем. Ця властивість інтелектуальних систем називається здатністю до навчання. Розрізняють зовнішнє навчання та самостійне навчання.[13]

Узагальнюючи все це, можна сказати, що інтелектуальна система - це самоврядна кібернетична система, яка має певний обсяг знань про світ і здатна на основі прямого сприйняття та подальшого аналізу поточної ситуації планувати дії для досягнення мети та навчання.

Типова схема роботи інтелектуальної системи.

Функціонування інтелектуальної системи можна описати як постійне прийняття рішень з урахуванням аналізу поточних ситуацій задля досягнення певної мети. Схема функціонування інтелектуальної системи представлена на рис. 1.1.

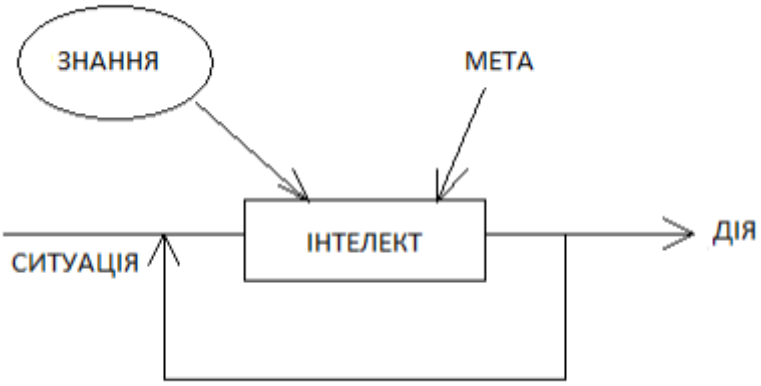

Рисунок 1.1 - Схема функціонування інтелектуальної системи

Вирізняють такі етапи функціонування інтелектуальних систем:

– безпосереднє сприйняття зовнішньої ситуації. Результатом є початковий опис ситуації;

– порівняння вихідного опису зі знаннями про систему та доповнення цього опису. Через війну формується вторинне опис ситуації у термінах знання системі. Цей процес можна розглядати як процес розуміння ситуації або як процес перекладу вихідного опису на внутрішню мову системи. Це може змінити внутрішній стан системи та її знання.[25] Вторинне опис може бути єдиним, і система може вибирати між різними вторинними описами. Крім того, система в процесі роботи може переходити від одного вторинного опису до іншого. Якщо ми зможемо формально визначити форми внутрішнього представлення описів ситуацій та операцій над ними, то ми сподіватимемося на деякий автоматизований аналіз цих описів;

– планування цілеспрямованих дій та прийняття рішень, аналіз можливих дій та їх наслідків та вибір тих дій, які найкраще узгоджуються з метою системи. Таке рішення зазвичай формулюється певною внутрішньою мовою (свідомо чи підсвідомо);

– зворотній інтерпретації рішення, тобто. формування робочого алгоритму реакції системи;

– реалізація реакції системи; наслідки – зміна зовнішньої ситуації, внутрішнього стану системи тощо;

Проте не слід вважати ці етапи повністю роздільними тому, що наступний етап починається тільки після повного завершення попереднього. Навпаки, взаємне проникнення цих етапів притаманно функціонування інтелектуальної системи. Наприклад, певні рішення можуть бути прийняті на етапі безпосереднього сприйняття ситуації. Насамперед це рішення про те, на які зовнішні подразники слід звернути увагу, а на які не варто. Зовнішніх подразників так багато, що їхнє сприйняття має бути вибірковим.

1.3 Методи аналізу т основні етапи розробки

У процесі дослідження штучного інтелекту відзначають три основні технології моделювання інтелектуальних систем:

– модель людського мозку. У поточному напрямі спостерігається розробка моделювання структури та механізми роботи мозку людини. За допомогою цієї моделі вчені намагаються виявити всі секрети якогось людського мислення. Для виникнення цієї моделі слід зробити такі етапи, як знаходження психофізіологічних даних, а також побудова та проведення досліджень моделей на підставі цього, вибудовування нових гіпотез;

– створення інтелектуальної машини з урахуванням комп'ютера. Даний напрямок дослідження спостерігається процес створення штучних систем за допомогою сучасних обчислювальних машин. Вчені направляють на формуванні алгоритмічного, програмного та апаратного забезпечення обчислювальних машин, що дозволяють вирішувати інтелектуальні завдання в будь-якій галузі

життя людини, наприклад, у прогнозування погоди або інтелектуальне отримання рішення;

– створення гібридної людини-машини. Третій підхід спирається створення змішаних машинно-людських (інтерактивних інтелектуальних систем), основою яких є можливість природного і штучного інтелекту;

– у процесі дослідження намагаються сформувати «супер-людини», яка може виконувати незвичайні відносини. З розвитком біотехніки та електромеханіки вчені потроху зможуть вживлювати пристрої у тіло людини.

Найважливішими проблемами у цих дослідженнях є:

– розподіл роботи між штучним та природним інтелектом;

– Організація діалогу між машиною і людиною.

Так само варто відзначити, що деякі з учених, зокрема Нік Бостром, вважають, що штучний інтелект може перевершити розум людини та використовувати її проти своєї волі, слід вміти контролювати технології.

Розглядають кілька видів розвитку систем штучного інтелекту:

– вирішення проблем, пов'язаних із наближенням штучного інтелекту до можливостей людини та їх об'єднання;

– створення штучного розуму - об'єднання систем штучного розуму в одну, що вирішує глобальні проблеми всього світу.

Комп'ютерний товарообіг розвивається з розвитком мікропроцесорної техніки та працюючих пристроїв, що дозволяють скористатися штучним інтелектом у різних напрямках.

Ускладнити застосування штучного інтелекту завтра може автоматизування програмування машин та у правильному донесенні експертних систем.

Перспектива розвитку штучного інтелекту є у всіх напрямах техніки. Швидкий процес обробки інформації дасть привілеї над ринком.

1.4 Ознайомлення з середовищами для розробки програмного додатку

Середовища розробки ПО (програмного забезпечення) є об'єднанням програмних засобів, які призначені для написання (створення) програмних продуктів. -Середовище розробки включає в свій зміст: компілятор, інтерпретатор, відладчик, засоби автоматизації збирання, а також редактор тексту.

Компілятор - це така програма, яка зчитує вихідні коди, написані програмістом і перетворює ці коди в програму.

Інтерпретатор - це програма яка зчитує команди, що знаходяться в вихідних кодах, відразу виконуючи їх.

Коли в середовищі розробки ПО присутні всі вищеназвані компоненти, тоді таке середовище називають інтегрованою. Такі середовища розробки збільшують темп, а також зручність розробки за рахунок: автоматизації, можливості виробляти весь цикл створення і розробки ПЗ.

Зазвичай середовище розробки ПО призначена для розробки тільки на одній мові програмування. А таке середовище розробки як інтегрована, надає право вибрати творцеві програми мову програмування для розробки, зручний розробнику (з мов підтримуваних цим середовищем). Прикладом цього є: Visual Studio, Komodo, Geany, Kylix, NetBeans, Eclipse.

MicrosoftVisualStudio - одна з інтегрованих середовищ розробки, розроблена на C ++ і C #, підтримується Windows OS. Дане середовище розробки переведена на десять мов (також і на російську мову). У Visual Studio творець може вести розробку веб-сайтів, веб-служб, писати консольні додатки, а також додатки з графічним інтерфейсом. Також VS підтримує різного роду доповнень. Найзнаменитіші доповнення - це Re Sharper (виконує пошук помилок в коді під час написання коду програми розробником, до компіляції); Visual Assist (на відміну від Re Sharper підтримує також і C ++); AnkhSVN (використовує в Visual Studio систему контролю версій, яка носить назву Subversion).

Переваги: Зрозумілий інтерфейс середовища розробки, зручність, автоматичне виявлення помилок в коді.

Недоліки: Складно для початківців програмістів.

Середовище особливо поширена в англомовних країнах, Росії, Китаї, Німеччині, Франції, Португалії, Італії, Японії, Іспанії та Кореї.

Geany також інтегроване середовище розробки програмного забезпечення. Підтримується на ОС Linux, а також на Mac Os і на Windows. Працює з тридцятьма двома мовами (також і з російською мовою). У складі Geany відсутня компілятор. Компілятор можна встановити як доповнення. Підтримує досить багато мов програмування, серед яких присутні класичний С. С ++ і С #.

Переваги: Простота і зручність, підсвічування вихідного коду, можливість підключати доповнення.

Недоліки: не включає до свого складу компілятор.

Серед розповсюджувалися в багатьох країнах (Більш ніж у тридцяти).

Komodo або ActiveStateKomodo - була написана на JavaScript, XUL, Python. Інтерфейс даного середовища тільки на англійською мовою. Работает на тех же операціонних система як Geany: на Os Linux, Windows і Mac Os.

Підтримує десять мов програмування, серед які присутні: PHP, Ruby, HTML5.

Переваги: Доповнення Code Explorer дозволяє переглядати об'єктне дерево скрипта або бібліотеки, середовище є кроссплатформенной, зручний відладчик з можливістю вилученого налагодження, можливість налаштувати інтерфейс середовища «під себе».

Недоліки: Висока вартість, підтримує мало мов програмування, сильно завантажує комп'ютер (а саме оперативну пам'ять), є складним для розуміння. Поширена в основному в англомовних країнах.

Kylix - інтегроване середовище. Функціонує на OS Linux. Працює з С, С ++ і ObjectPascal. В даному середовищі є можливість писати програми веб-служб. Kylix випускався в трьох пакетах. Ці пакети: Enterprise Edition - включав в себе сто дев'яносто компонентів (був найбільшим і самим дорогим пакетом програми); Professional Edition (дешевший варіант, який включав в себе близько 165 компонентів); Open Edition - безкоштовний пакет програми, що містить в собі 75 компонентів, в ньому відсутній кошти для роботи з базами даних.

Оновлена версія Kylix 2, на відміну від Kylix працювала набагато швидше. Наприклад, Kylix 1 здійснював сортування бульбашкою масиву з 115 елементів півтори хвилини, Kylix 2 - одну секунду.

У 2002 році дане середовище розробки припинив підтримувати розробник.

Переваги: Зручний в перенесенні написаного з однієї операційної системи на іншу.

Недоліки: Дане середовище більше не підтримується розробником.

Поширена в основному в Європейських країнах і США, через те що розробник (Borland) перестав підтримувати Kylix - стає все менш популярною і не затребуваною.

Netbeans - інтегроване середовище розробки програмного забезпечення. Була реалізована на програмному мовою Java. Це середовище розробки високої якості. Вміє працювати на декількох операційних системах, тобто є кроссплатформенной. Працює більш ніж з п'ятьма програмними мовами.

Переваги: Є безкоштовною, присутній система контролю версій, підсвічування синтаксису, можливо перейменовувати змінну / клас одним кліком, в тому випадку якщо вручну перейменовувати занадто довго (автоматизоване перейменування), є можливість форматування коду по CodeStyle, розробником середовище постійно вдосконалюється, поліпшується.

Недоліки: Часом в середовищі розробки виникають проблеми з кодуванням, довгий запуск програми.

Поширена в багатьох країнах, в силу того що є зручною і безкоштовною.

Eclipse - ще одна інтегрована середовище розробки ПО. Написана на мові Java в дві тисячі третьому році. Також є кроссплатформенной. За рахунок приєднуються до цього середовища доповнень - є можливість створювати програмні продукти більш ніж на п'яти мовах програмного коду.

Переваги: Постійне оновлення версій середовища розробки, підтримка багатьох мов (в тому числі і російського), є безкоштовною, підтримка багатьох мов програмування, середовище має промисловий рівень, є гнучкою - тобто легко налаштовується як під будь-яку платформу, так і під будь-якого користувача.

Недоліки: Сильно завантажує оперативну пам'ять комп'ютера, довго запускається, однак, якщо комп'ютер досить потужний - дана проблема легко вирішувана. Поширена в багатьох країнах, користується популярністю.

1.5 Ознайомлення з сучасними системами програмування

1995 рік був особливо цікавим роком у світі комп’ютерного програмування. Саме в цьому році було випущено чотири нові мови програмування, які вплинули на світове співтовариство програмування таким чином, що не передбачалося на момент їх офіційного оголошення. До речі, це був також час, коли Інтернет тільки починав робити хвилі, а Інтернет був на межі вибуху в основну цифрову культуру.

Чотири мови, які дебютували в 1995 році, були Java, JavaScript, PHP і Ruby. Незважаючи на те, що їхні відповідні релізи не супроводжували особливої фанфари, ці мови з часом виростуть і стануть повсюдними інструментами програмування для більшості розробників програмного забезпечення. До цього часу домінуючими мовами були C і C++. Незважаючи на те, що ці двоє були дуже потужними, вони за своєю суттю не підходили для всесвітньої мережі. Крім того, для початківців програмістів (особливо C++) вони часто вважалися дещо складними та залякуючими.

Серед чотирьох, Java виявилася шаленим успіхом. З його часто цитованим гаслом «Напишіть один раз, запустіть будь-де» Java стала миттєвим хітом, оскільки її було набагато легше вивчити та опанувати (порівняно з C++). Java також представила нову ідею віртуальної машини (JVM), яка дозволила писати програми, які запускалися б на різних платформах без необхідності перекомпіляції.

В останні роки JavaScript витіснив Java як провідну мову програмування у світі. Постійне зростання JavaScript значною мірою пов’язано з впровадженням Node.js, технології, яка зробила можливим запуск JavaScript на стороні сервера.

PHP став домінуючою силою у сфері веб-програмування, особливо в поєднанні з іншими популярними технологіями з відкритим вихідним кодом,

такими як Linux, Apache та MySQL. Разом вони утворили те, що зазвичай називають стеком LAMP.

Ruby здобув популярність серед веб-розробників після випуску шалено популярного веб-фреймворку Ruby on Rails у 2005 році датським програмістом Девідом Хайнемайєром Ханссоном (DHH).

Для великих (або командних) проектів в середу розробки повинні бути включені файловий менеджер, інтегроване середовище розробки програмного забезпечення, PlSql (використовується і для роботи з Системою Управління БД і як інструмент звітів), Cristal Reports (створення звітів), StarTeam (ведення журналу версій продукту, що розробляється).

Підводячи підсумки потрібно сказати про те, що інтегровані середовища розробки ПО дозволяють програмістам скоротити час на написання додатків, знизити затратність на написання (розробку), підвищити зручність розробки - що і є однією з основних цілей програмної інженерії.

Інтегровані середовища розробки зручні для командних проектів, остільки оскільки в таких середовищах можна виробляти весь цикл створення програмного забезпечення. Інтегровані середовища зручні в написанні програм.

1.6 Структура аналізу розпізнання рукописного тексту

Алгоритм роботи даних методів полягає у розбиття вихідного образу на складові, що описуються як стабільні ідеальні елементи.

Одні з методів даного напрямку використовують для розпізнавання топологічний опис зображень. Іншими словами, стандарт містить інформацію про взаємне становище окремих складових елементів знака. При цьому стає неважливим розмір літери, що розпізнається, і навіть шрифт, яким вона надрукована. Можливі зображення, що становлять той чи інший клас, можна як результат гомеоморфних перетворень деякого еталонного зображення, відповідного цього класу. Завдання розпізнавання у разі може бути зведена до встановлення гомеоморфності пред'явленого зображення з однією з еталонних. Її можна виявити за допомогою топологічних інваріантів - таких властивостей

зображення, які не змінюються за його гомеоморфних перетворень. Інваріантом, що дозволяє дати чисельний опис зображень, є, наприклад, кількість ліній, що сходяться в точці. Відповідний опис виходить обходом у порядку контурів зображення з одночасної фіксацією індексів точок. Встановлення гомеоморфності – власне розпізнавання – зводиться до порівняння описів пред'явленого зображення та еталонних зображень класів. Важлива перевага топологічного опису - його нечутливість до сильних деформацій зображення, що включає всі перетворення подібності, якщо пов'язувати з кожним зображенням певну характерну точку, з якої починається обхід. Навчання топологічного коду полягає у веденні набору еталонних описів.

Інші структурні методи реалізують алгоритм подієвого розпізнавання. Події метод спирається на топологічну структуру об'єкта, що складається з ліній і не змінюється при малих деформаціях образу. Лінією називається частина образу, у кожному перерізі якого є лише один інтервал. Лінії, що огрублені на деякій сітці, визначають події. Подієве уявлення є не лише формальним набором ознак, а й адекватним топологічним описом. Навчання методу зводиться до складання списку еталонів досить великий послідовності образів. При розпізнаванні вихідного растру визначається подієве уявлення, якому зіставляється еталонний клас.

Основною проблемою структурних методів розпізнавання є ідентифікація знаків, що мають дефекти.

Перевагами методу є здатність розпізнавання спотворених символів та швидкодія при малому алфавіті.

Шаблонні методи

Алгоритм роботи шаблонних методів спирається зіставлення вхідного графічного зображення, ідеальному шаблону. Першим етапом роботи шаблонного методу є перетворення відсканованого зображення на растрове. У процесі розпізнавання перебираються шаблони і обчислюється відстань від образу до шаблону. Клас, шаблони якого знаходяться на мінімальній відстані від вхідного образу є результатом розпізнавання.

Дані методи поділяються на два класи шрифтозалежні та шрифтонезавізні. Шрифтонезалежні методи використовують заздалегідь певні шаблони, і універсальні для всіх типів шрифтів. Однак за такого підходу знижується вірогідність правильного розпізнавання. Шрифтозалежні алгоритми розраховані тільки на один тип шрифту, це підвищує якість їх роботи, але вони зовсім не працездатні при використанні інших шрифтів. Були запропоновані методи розпізнавання великих обсягів тексту, коли частина символів гарантовано розпізнається шрифтонезалежними методами, а потім на підставі розпізнаних символів будуються шаблони для шрифтозалежних алгоритмів.

При достатку друкованої продукції в процесі навчання неможливо охопити всі шрифти та їх модифікації.

До переваг цього алгоритму відносяться: простота реалізації, надійна робота в умовах відсутності перешкод, висока точність розпізнавання дефектних символів, швидкість при малому алфавіті.

До недоліків можна віднести: сильну залежність від шаблонів та складність підбору оптимальних шаблонів, неможливість розпізнати шрифт, що відрізняється від закладеного в систему, повільна робота при великій кількості перешкод, чутливість до обертання, шумів та спотворень.

Ознакові методи

Ці методи базуються на тому, що зображенню ставиться у відповідність N-мірний вектор ознак. Розпізнавання полягає у порівнянні його з набором еталонних векторів тієї ж розмірності. Прийняття рішення про належність образу тому чи іншому класу, виходячи з аналізу обчислених ознак, має низку суворих математичних рішень у межах ймовірнісного підходу. Тип і кількість ознак значною мірою визначають якість розпізнавання. Формування вектора відбувається під час аналізу зображення. Цю процедуру називають вилученням ознак. Еталон для кожного класу отримують шляхом аналогічної обробки символів навчальної вибірки.

До переваг методу можна віднести: простота реалізації, висока узагальнююча здатність, стійкість до зміни форми символів, висока швидкодія.

До недоліків методу належать: нестійкість до різних дефектів зображення, втрата інформації про символ на етапі отримання ознак.

1.7 Огляд наявних алгоритмів

Алгоритм виявлення ліній.

Даний алгоритм розроблений для того, щоб сторінка, перевернута під якимось кутом, могла бути розпізнана без усунення нахилу - це дозволяє уникнути втрати якості зображення. Ключова частина даного процесу - фільтрація контурів і конструкція ліній. Вважаючи, що аналіз вмісту сторінки вже надав регіони тексту, приблизно одного розміру, простий фільтр висот видаляє великі літери, з яких починається сторінка в деяких текстових файлах.

Медіана висот апроксимує розмір тексту в регіонах, тому потрібно фільтрувати контури, менші за цю медіану. Дані елементи, з високою ймовірністю, є пунктуацією або шумом.

Відфільтровані контури задовольняють моделі паралельних ліній, що не перекриваються. Сортування та обробка контурів по «x» координаті дозволяє присвоїти контур унікальному рядку, відстежуючи нахил по сторінці, що значно зменшує ймовірність присвоєння до некоректного рядка у разі перекосу. Після того, як відфільтровані контури були закріплені за лініями, менша медіана квадратів використовується для оцінки базових ліній і відфільтровані контури встановлені назад у відповідні рядки.

Остання стадія процесу створення ліній з'єднує контури, що перекривають один одного по горизонталі хоча б на половину, накладаючи коректну базу та коректно співвіднесені частини поламаних символів.

Розпізнавання слів.

Завдання процесу розпізнавання для будь-якого методу розпізнавання образів це визначити, як слово має бути поділено на символи. Початковий результат сегментації за знайденими лініями класифікується першим. Решта етапу розпізнавання слів застосовується лише до тексту з не фіксованим кроком пробілів.

Поділ з'єднаних символів.

Необхідно зробити, обрізаючий контур, ґрунтуючись на даних класифікатора: видаляються елементи найгірше відповідні йому. Кандидатури точок поділу знаходяться з увігнутих вершин полігональної апроксимації кордонів і можуть бути як іншими вершинами, так і сегментами лінії. Для успішного поділу зв'язаних символів з «ASCII» набору необхідні три пари таких точок.

1.8 Висновки до розділу

В цьому розділі ми ознайомилися з мовами програмування, з середовищами програмування. З усіх мов та середовищ, ми обрали мову C++ та середовище Microsoft Visual Studio 2019. Також розповідається про штучний інтелект, про його використанні на практиці. Розглядаються аналоги програми для розпізнання тексту. Та ознайомлюється з методами та алгоритмами для розпізнання тексту.

ОПИС ПРОГРАМНОГО ДОДАТКУ

1.1 Опис технічного завдання

Розробити програму, яка реалізує систему аналізу тексту електронної пошти на наявність шкідливого контенту, з використанням Штучного інтелекту/Машинного навчання (TensorFlow). Реалізувати прогнозування різними методами. Розробити можливість взаємодії з користувачем. Надати можливість вибору введення даних вручну, автоматично або зчитуванням із файлу. Також додати можливість зберігати отриманні данні до файлу.

1.2 Опис середовища та системи для створення власного продукту

Середовище розробки - це набір процесів та інструментів, які використовуються для розробки вихідного програмного коду або програмного продукту. Це включає в себе все середовище, яке підтримує наскрізний процес, включаючи сервери розробки, підготовки та виробництва. Середовище розробки автоматизує або спрощує процедури, пов'язані зі створенням, тестуванням, налагодженням, встановленням виправлень, оновленням та супроводом програмного забезпечення, включаючи довгострокове обслуговування.

В рамках всеосяжного середовища розробки ви можете знайти інтегроване середовище розробки (IDE), яке є інструментом розробки, який розробники використовують для написання, складання, тестування та налагодження програмних програм. IDE надає зрозумілий та узгоджений інтерфейс, який підтримує процеси написання та тестування коду, а також його упаковки для випуску. Наявність єдиного інструменту для створення, зміни, компіляції, розгортання та налагодження програмного забезпечення скорочує час, необхідний для налаштування, коли команді доводиться збирати разом різноманітні утиліти розробки. Це також підвищує загальну продуктивність

розробника та призводить до більш ефективного загального середовища розробки.

1.3 Засоби розробки

Написання безпечного коду, що забезпечує безпеку без уразливостей, є критичним викликом для кібербезпеки. Написання коду без уразливостей вже давно принаймні так само складно, як написання коду без помилок. Хоча в програмному забезпеченні є багато інших потенційних джерел ризику безпеки, розробка коду без відомих класів уразливостей завжди здавалася посильною метою. Він покладається на людей-розробників, які використовують інструменти, методи та процеси для створення програмного забезпечення, яке не має особливо відомих типів дефектів.

Один з найефективніших підходів — дослідження мов та інструментів програмування — приніс технології, які, як показано, протистоять категоріям уразливостей, в основному, не допускаючи їх. Безпечні мови пам'яті, які керують виділенням та звільненням пам'яті, замість того, щоб вимагати від програміста це робити, унеможливлюють для розробників створення вразливостей переповнення буфера та деяких інших типів впливу через відсутні перевірки меж масиву, використання нульового покажчика та даних. витік через повторне використання пам'яті. Потокобезпечні мови можуть вирішувати випадки, коли умови гонки можуть бути використані, щоб підірвати перевірки, пов’язані з безпекою в програмі.

У межах спільноти розробників програмного забезпечення групи та організації, які мають на меті безпечну розробку програмного забезпечення, включили інструменти та методи у свій життєвий цикл розробки програмного забезпечення, щоб включити життєвий цикл безпечної розробки. Раннє програмне забезпечення з високою надійністю використовувало формальні методи для визначення властивостей безпеки системи, а також перегляд коду, щоб використовувати людину для пошуку таких недоліків на рівні кодування.2 Корпорація Майкрософт створила свій життєвий цикл розвитку безпеки,

додавши аналіз першопричин, навчання безпеки, моделювання загроз, конкретні вимоги до безпечного кодування та тестування безпеки, яке включало тестування на проникнення та нечіткое тестування. Практика, як правило, впроваджується на основі потреб бізнесу, відчутного впливу на безпеку та відповідає усталеній або розвивається практиці розвитку.

Необхідні дослідження, які впливають на те, що працює, а що може працювати для безпечного розвитку. Здається, що нинішні дослідження відіграють, на жаль, обмежену роль у створенні, пропонуванні, оцінці та підтвердженні інструментів, методів і процесів, які використовуються на практиці для безпечного розвитку. Зокрема, дослідження рідко стосуються безпосередньо інструментів і методів, оскільки вони використовуються, у контексті, в якому вони використовуються. Нам потрібні додаткові дослідження ефективності та результатів безпечних інструментів, методів і процесів розробки. Це дослідження можна судити про його вплив на те, як розробка програмного забезпечення працює на практиці. Властивості дослідження впливають на ймовірність такого впливу.

Суворість дослідницького наукового експериментування вимагає ряду вимог до процесу, включаючи формулювання гіпотези, що перевіряється, контроль змінних експерименту, щоб переконатися, що експеримент фактично перевіряє гіпотезу, і аналіз експериментальних даних і результатів для математичного підтвердження гіпотези (або спростувати нульову гіпотезу). Хоча ці процеси можуть стати основою важливих фундаментальних досліджень безпечної розробки, вони часто уникають безладних реалій, пов’язаних із впровадженням техніки на практиці, саме тому, що ці безладні реалії ускладнюють проектування експериментів.

Негативні результати дослідження, які не підтверджують, що безпечна техніка розробки підвищує безпеку, хоча й важливі для галузі досліджень, навряд чи вплинуть на безпечний розвиток на практиці. Перший урок розробника безпеки у великій технологічній компанії полягав у тому, що вказувати розробникам не робити чогось майже завжди було неефективним, якщо це не було поєднане з альтернативою, яку вони могли б використати для досягнення

мети застарілої практики. Крім того, виявити, що інструмент або техніка експериментально неефективні для забезпечення безпеки, не доводить, що вони неефективні за межами контрольованого експерименту, у більш широкому, безладному та різноманітнішому контексті розробки програмного забезпечення.

Які з тих речей, які робляться в дослідженні, сподіваються на практичне перенесення в безпечний розвиток? Дві сучасні тенденції досліджень безпеки дають певну надію на безпечний розвиток. Одна з них полягає в тому, що безпечна розробка стала темою на конференціях з дослідження безпеки, охоплюючи такі теми, як оцінка інструментів, які допомагають розробникам уникати вразливостей, і вимірювання здатності розробників кодувати функціональні можливості, що стосуються безпеки. [17]

Інша обнадійлива тенденція — оцінка артефактів. Багато розробок програмного забезпечення базується на існуючому програмному забезпеченні, використовуючи фреймворки, бібліотеки та відкритий код. Пропонування артефакту, який використовується для встановлення та підтвердження ідеї дослідження, зменшує бар’єри для передачі цієї ідеї в розробку програмного забезпечення. Доступ до коду з відкритим кодом з умовами ліцензії, зручними для повторного використання, може збільшити його потенціал для використання. Деякі дослідницькі стимули змінюються, щоб заохочувати подання артефактів у рамках процесу подання та публікації наукової роботи на конференціях з безпеки, таких як USENIX Security і ACSAC.

1.4 Вимоги до технічного забезпечення

Запуск комп’ютерних програм на голій машині – не нова концепція. Так почалися обчислення десятиліття тому, коли програма завантажується вручну за допомогою перемикачів і запускається натисканням кнопки запуску. У цій статті ми про це не говоримо! Однак наші комп’ютерні системи складні, і вони виросли за межами пропорції, створюючи великий семантичний розрив між додатками та обладнанням. Наприклад, остання операційна система (ОС) Microsoft XP має 40 мільйонів рядків коду. Настав час переглянути еволюцію обчислювальної

техніки та шукати рішення, використовуючи нові парадигми. ОС і середовище швидко змінювалися з останніх 30 років, що робить речі застарілими, перш ніж вони зможуть стати продуктивними протягом свого життя. Наприклад, Microsoft випустила понад 20 основних випусків ОС за останні 25 років. Кожна мова та середовище випускають новий випуск кожні шість місяців. Коли ОС або її середовище змінюються, це має ефект пульсації, що призводить до застарілих програм. Як ми можемо зупинити таке поширення продуктів і технологій і зробити програми стабільними? Не забезпечуючи жодної перевірки, одним із можливих підходів є уникнення операційної системи та її середовища. У деяких областях обчислень це робиться повільно, непомітно.

Після завантаження ПК він перебуває в реальному режимі, де адресація обмежена 1 МБ і немає захисту вашого коду. Це просто в режимі дискової операційної системи (DOS). Щоб завантажувати та запускати більші програми та мати доступ до більшої пам’яті, вам потрібно навчитися переходити в захищений режим за допомогою інструкцій на мові асемблера. Однак усі виклики базової системи введення/виводу (BIOS) доступні лише в реальному режимі. Якщо ви хочете використовувати BIOS для вводу-виводу, вам потрібно мати механізми перемикання із захищеного режиму в реальний для виконання викликів BIOS. Таким чином, ваша програма на C++ або якась керуюча програма повинні виконувати це перемикання, прозоре для користувача. Якщо ви плануєте використовувати програмні переривання (їх 255) замість переривань BIOS, то вам потрібно написати власний код на зборці для вирішення всіх операцій введення-виводу. Ми використовували як BIOS, так і програмні переривання і написали перемикач захищеного режиму в режим реального для роботи в обох режимах. Це нетривіальна річ, яку потрібно зробити під час складання, яка вимагає від вас ознайомлення з документом специфікації архітектури Intel, який доступний на їх веб-сайті. Цей документ є корисним ресурсом для розуміння та реалізації переривань, схем адресації, засобів завдань, пасток і винятків.[18]

У звичайну програму C++ ми включаємо файли *.h, які є бібліотеками для ОС, які будуть включені до виконуваного файлу. Якщо ми використовуємо будь-які конструктори C++, вони автоматично включатимуть ці бібліотеки, які не

мають сенсу в середовищі без ОС. Таким чином, середовище компіляції має використовувати пакетні файли для компіляції та зв'язування необхідних модулів. Як програміст прикладних програм на рівні мови С++, ми розробили файл інтерфейсу, який містить усі інтерфейси, необхідні для цих програм. Цей файл має API, включаючи: введення/виведення, завдання, пам'ять, таймер, спільну пам'ять, блокування тощо. Для створення виконуваних модулів ми використовуємо компілятор Visual Studio C++ (пакетний режим), ассемблер MASM 6.11 і компілятор Turbo. Ми написали всі пакетні файли для компіляції та компонування для завантаження, початкових програм і прикладних програм. Ці пакетні файли прості та легкі для розуміння та використання. API - це виклик С++, який, у свою чергу, викликає виклик С, а той, у свою чергу, викликає виклик збірки. Програміст С++ бачить лише API. Якщо інтерфейс можна створити на рівні С++, то код буде реалізовано на самому рівні С++. Ми уникаємо використання будь-яких прямих викликів збірки в програмі С++; однак такий виклик дозволений компілятором.

1.5 Опис архітектури програмного забезпечення

Архітектура програмного забезпечення дає пояснення того, як ваші системи поводяться на структурному рівні. Системи, які ви використовуєте, мають набір компонентів, розроблених для виконання певного завдання або набору завдань. Архітектура програмного забезпечення забезпечує фундамент, на якому все програмне забезпечення, яке є у компанії, можна змінити, створити або вилучити з експлуатації.

Архітектура програмного забезпечення впливає на якість, продуктивність, обслуговування та успіх системи на основі дизайну. Не розглядаючи архітектуру програмного забезпечення на регулярній основі, компанія відкриває себе для довгострокових наслідків, і проблеми, які можуть поставити їх системи під загрозу поломки, злому або низької продуктивності.

У сучасних системах існують загальні шаблони в архітектурі програмного забезпечення, які називаються архітектурними системами для програмного

забезпечення. У більшості випадків для створення цілісної системи використовується кілька різних архітектурних систем, особливо для систем, які створювалися роками або працювали, або тих, які були побудовані різними розробниками.

1.5.1 Опис класів

Клас у C++ є будівельним блоком, який веде до об'єктно-орієнтованого програмування. Це визначений користувачем тип даних, який містить власні члени даних і функції-члени, до яких можна отримати доступ і використовувати, створивши екземпляр цього класу. Клас C++ схожий на план для об'єкта.

Наприклад: розглянемо клас автомобілів. Може бути багато автомобілів з різними назвами та марками, але всі вони будуть мати деякі спільні властивості, як-от усі вони матимуть 4 колеса, обмеження швидкості, діапазон пробігу тощо. Отже, автомобіль – це клас і колеса, обмеження швидкості, пробіг. їх властивості.[19]

Клас — це визначений користувачем тип даних, який має члени даних і функції-члени.

Члени даних — це змінні даних, а функції-члени — це функції, які використовуються для маніпулювання цими змінними, і разом ці члени даних і функції-члени визначають властивості та поведінку об’єктів у класі.

На прикладі класу Car членом даних буде обмеження швидкості, пробіг тощо, а функціями члена можуть бути гальмування, збільшення швидкості тощо.

Об'єкт - це екземпляр класу. Коли клас визначено, пам’ять не виділяється, але коли він створюється (тобто створюється об’єкт), пам’ять виділяється.

Визначення класу та оголошення об'єктів

Клас визначається в C++ за допомогою ключового слова class, за яким слідує ім'я класу. Тіло класу визначається всередині фігурних дужок і закінчується крапкою з комою в кінці.[18, 22]

Оголошення об'єктів: коли визначено клас, визначається лише специфікація об'єкта; пам'ять або сховище не виділено. Для використання даних і функцій доступу, визначених у класі, потрібно створити об’єкти.

Синтаксис:

ClassName ObjectName;

Доступ до членів даних і функцій-членів: До членів даних і функцій-членів класу можна отримати доступ за допомогою оператора dot(‘.’) з об’єктом. Наприклад, якщо ім’я об’єкта – obj, і ви хочете отримати доступ до функції-члена з ім’ям printName(), тоді вам доведеться написати obj.printName() .

Доступ до членів даних.

Доступ до загальнодоступних членів даних також здійснюється таким же чином, але об’єкт не може мати прямий доступ до членів приватних даних. Доступ до члена даних залежить виключно від контролю доступу цього члена даних.

Цей контроль доступу надається модифікаторами доступу в C++. Існує три модифікатори доступу: відкритий, приватний і захищений.

Клас - це шаблон або план, який зв'язує властивості та функції сутності. Ви можете помістити всі сутності або об’єкти, що мають подібні атрибути, під одним дахом, відомим як клас. Класи далі реалізують основні концепції, такі як інкапсуляція, приховування даних та абстракція. У C++ клас діє як тип даних, який може мати кілька об’єктів або екземплярів типу класу.

Розглянемо в якості прикладу клас Products:

```
class Products
{
public:
    Products();

    void Create(int count_tov, int count_day, int num);
    void Print();
    void Method1();
    void Method2();
```

```
    void Method3();
    void save();

    ~Products();

private:
    vector<vector<int>> arr;
    vector<string> product_name;
    vector<double> prognoz;
};
```

В ньому є конструктор, деструктор. У private частині зберігаються змінні. Також в ньому є декілька функцій які зберігаються у public секції.

1.5.2 Опис методів

Створення програми включає написання коду, тестування коду та виправлення будь-яких неправильних частин коду або налагодження. Проаналізуйте процес написання програми та дізнайтеся, як цей процес можна спростити за допомогою програмного забезпечення для редагування коду.

Загальні кроки для написання програми включають:

– зрозуміти проблему, яку ви намагаєтесь вирішити;

– розробите рішення;

– намалюйте блок-схему;

– написати псевдокод;

– написати код;

– тестування та налагодження;

– протестуйте з реальними користувачами;

– програма випуску.

Нижче представленна діаграма компонентів, які будуть використані у програмі Рис. 2.1

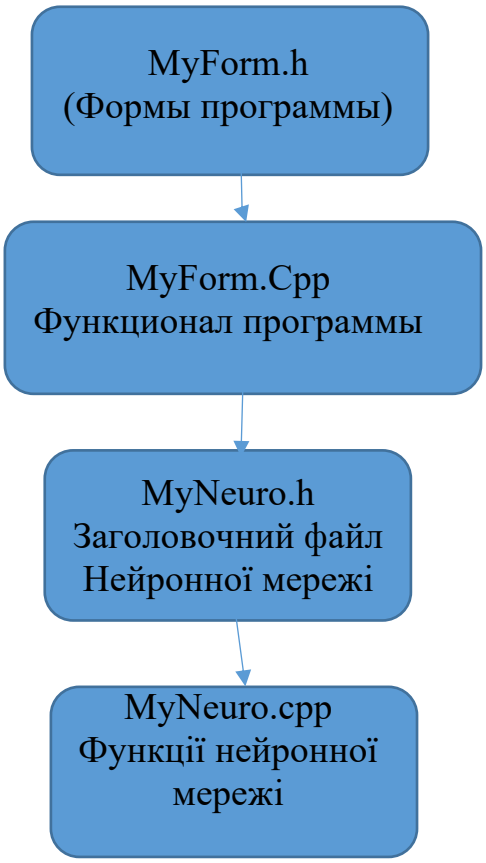

Рисунок 2.1 – Діаграма компонентів

1.6 Висновки до розділу

У другому розділі розповідається про технічне завдання. Також описані середовище та системи які потрібні для створення власного складу. Розповідається про засоби розробки та які необхідні вимоги до технічного забезпечення.

Також розповідається про класи які використовувалися у роботі та методи якими користувалися для прогнозування товарів. Реалізована діаграма компонентів програми.

РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

1.7 Опис розроблювального алгоритму

Завдяки бурхливому розвитку науки інформатики і проникненню їх у різні галузі народного господарства слово " алгоритм " стало найпоширенішим і найбільш вживаним у життєвому плані поняттям для широкого кола фахівців. Більше того, з переходом до інформаційного суспільства алгоритми стають одним з найважливіших факторів цивілізації.

Відомо, що математична теорія алгоритмів склалася зовсім у зв'язку з бурхливим розвитком інформатики та обчислювальної техніки, а виникла надрах математичної логіки на вирішення її проблем. Вона, перш за все, дуже вплинула на світогляд математиків і їх науку.

Тим не менш, взаємовплив теоретичних областей, пов'язаних з обчислювальною технікою, та теорії алгоритмів також, безсумнівно.[19]

Теорія алгоритмів вплинула на теоретичне програмування. Зокрема, велику роль теоретичному програмуванні відіграють моделі обчислювальних автоматів, які, за суттевістю, є обмеженнями тих представницьких обчислювальних моделей, створених раніше у теорії алгоритмів. Трактування програм, як об'єктів обчислення, оператори, що використовуються для складання структурованих програм (послідовне виконання, розгалуження, повторення), прийшли в програмування з теорії алгоритмів. Зворотний вплив виявилося, наприклад, у тому, що виникла потреба у створенні та розвитку теорії обчислювальної складності алгоритмів. Отже, можна сказати, що теорія алгоритмів застосовується у інформатиці, а й у інших галузях знань.

Поняття алгоритму та його властивості.

Слово «алгоритм» означає «процес або набір правил, яких слід дотримуватися під час обчислень або інших операцій з вирішення проблем». Тому Алгоритм відноситься до набору правил/інструкцій, які крок за кроком визначають, як має виконуватися робота, щоб отримати очікувані результати.

Це можна зрозуміти, взявши приклад приготування за новим рецептом. Щоб приготувати новий рецепт, потрібно читати інструкції та кроки та виконувати їх по черзі в заданій послідовності. Отриманий таким чином результат – нове блюдо, приготоване ідеально. Аналогічно, алгоритми допомагають виконати завдання в програмуванні, щоб отримати очікуваний результат.

Розроблений алгоритм не залежить від мови, тобто це прості інструкції, які можна реалізувати будь-якою мовою, але результат буде таким же, як і очікувалося.[20]

Оскільки не слід дотримуватись жодних письмових інструкцій, щоб приготувати рецепт, а лише стандартний. Так само не всі письмові інструкції з програмування є алгоритмами.[21] Для того, щоб деякі інструкції були алгоритмом, вони повинні мати такі характеристики:

Чітко і однозначно: алгоритм повинен бути зрозумілим і недвозначним. Кожен із його кроків має бути зрозумілим у всіх аспектах і вести лише до одного значення.

Добре визначені вхідні дані: якщо алгоритм каже приймати вхідні дані, це повинні бути чітко визначені вхідні дані.

Добре визначені результати: алгоритм повинен чітко визначити, який вихід буде отримано, і він також повинен бути чітко визначений.

Скінченність: Алгоритм повинен бути скінченним, тобто він не повинен опинитися в нескінченних циклах або тому подібних.

Можливість: Алгоритм повинен бути простим, загальним і практичним, щоб його можна було виконувати за наявності доступних ресурсів. Він не повинен містити якусь майбутню технологію чи щось таке.

Незалежний від мови: розроблений алгоритм повинен бути незалежним від мови, тобто це повинні бути просто прості інструкції, які можна реалізувати будь-якою мовою, але результат буде таким же, як і очікувалося. [20]

Переваги алгоритмів:

– це легко зрозуміти;

– алгоритм — це поетапне представлення рішення заданої задачі;

– в алгоритмі проблема розбивається на менші частини або кроки, отже, програмісту легше перетворити її на справжню програму.

Недоліки алгоритмів:

– написання алгоритму займає багато часу, тому займає багато часу;

– розгалуження та циклічні оператори важко показати в алгоритмах.

Як розробити алгоритм?

Для того, щоб написати алгоритм, необхідні такі речі як передумова:

– задача, яку потрібно вирішити за допомогою цього алгоритму;

– обмеження проблеми, які необхідно враховувати при розв’язуванні задачі;

– вхідні дані, які потрібно зробити для вирішення проблеми;

– результат, якого слід очікувати, коли проблема буде вирішена;

– рішення цієї задачі, в заданих обмеженнях.

Потім алгоритм записується за допомогою вищевказаних параметрів так, щоб він вирішував задачу.[22]

Приклад: Розглянемо приклад, щоб додати три числа та надрукувати суму.

Крок 1: Виконання попередніх умов.

Як обговорювалося вище, для того, щоб написати алгоритм, повинні бути виконані його передумови.

Задача, яку потрібно розв’язати за цим алгоритмом: Додайте 3 числа та виведіть їх суму.

Обмеження задачі, які необхідно враховувати при розв’язуванні задачі: числа повинні містити тільки цифри і ніяких інших символів.

Вхідні дані, які потрібно зробити для розв’язання задачі: три числа, які потрібно додати.

Вихід, якого слід очікувати, коли задачу буде розв’язано: сума трьох чисел, взятих як вхідні дані.

Розв’язання цієї задачі за наведених обмежень: Рішення складається з додавання 3 чисел. Це можна зробити за допомогою оператора «+», або порозрядно, або будь-яким іншим способом. [22]

Крок 2: Розробка алгоритму

Тепер давайте розробимо алгоритм за допомогою наведених вище умов:

Алгоритм додавання 3 чисел і виведення їх суми:

СТАРТ

Оголосити 3 цілі змінні num1, num2 і num3.

Візьміть три числа, які потрібно додати, як вхідні дані у змінних num1, num2 та num3 відповідно.

Оголосіть цілу змінну sum, щоб зберегти підсумкову суму трьох чисел.

Додайте 3 числа та збережіть результат у сумі змінної.

Вивести значення змінної суми

КІНЕЦЬ

Крок 3: Тестування алгоритму шляхом його реалізації.

Напишемо блок схему до нашої програми Рис 3.1

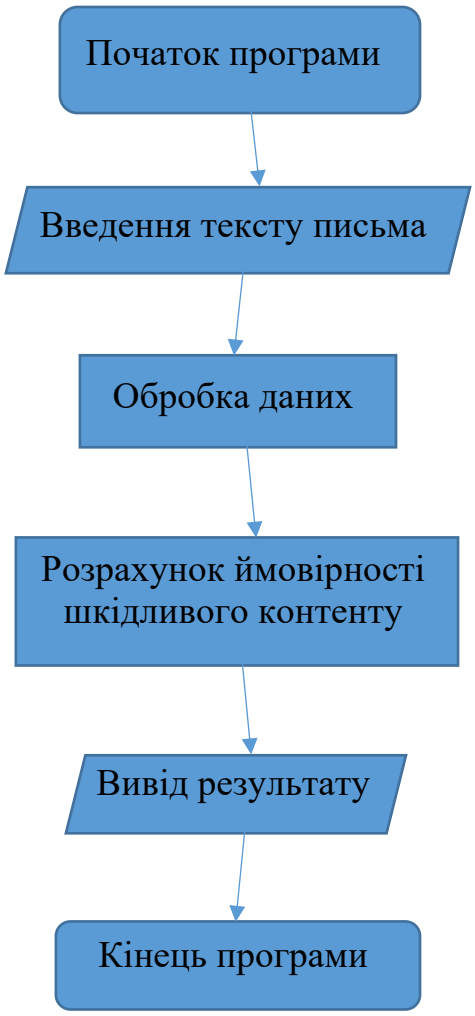

Рисунок 3.1 – Блок схема алгоритму програми

1.8 Розробка власного продукту

Етапи створення власного продукту.

1. Генерація ідеї.

Багато підприємців-початківців застрягають на першому етапі: створення ідей та мозковий штурм. Це часто відбувається тому, що вони чекають геніальності, щоб виявити ідеальний продукт, який вони повинні продавати. Хоча створення чогось принципово «нового» може бути креативним, багато з найкращих ідей є результатом повторення існуючого продукту.

2. Дослідження.

Зважаючи на свою ідею продукту, ви можете відчути бажання перескочити у виробництво, але це може стати помилкою, якщо ви не спершу підтвердите свою ідею.[21]

Перевірка продукту гарантує, що ви створюєте продукт, за який люди будуть платити, і що ви не витрачатимете час, гроші та зусилля на ідею, яка не продається. Є кілька способів підтвердити свої ідеї щодо продуктів, зокрема:

– обговоріть свою ідею з родиною та друзями;

– надсилання онлайн-опитування, щоб отримати відгук;

– початок краудфандингової кампанії;

– запит на відгуки на таких форумах, як reddit;

– дослідження ринкового попиту за допомогою google trends;

– запуск сторінки «незабаром», щоб оцінити інтерес за допомогою електронної пошти або попереднього замовлення;

3. Планування.

Оскільки розробка продукту може швидко стати складною, важливо приділити час для планування, перш ніж почати створювати свій прототип.

Коли ви врешті-решт звертаєтесь до виробників або починаєте шукати матеріали, якщо ви не маєте конкретного уявлення про дизайн вашого продукту та як він буде функціонувати, на наступних кроках легко заблукати.

Найкраще почати планування з намальованого від руки ескізу того, як буде виглядати ваш продукт. Ескіз має бути якомога детальнішим, з етикетками, що пояснюють різні функції та функції.

4. Прототипування.

Метою етапу створення прототипу під час розробки продукту є створення готового продукту для використання в якості зразка для масового виробництва.

Малоймовірно, що ви отримаєте готовий продукт за одну спробу — створення прототипу зазвичай включає в себе експерименти з декількома версіями продукту, повільне усунення опцій і вдосконалення, поки ви не будете задоволені кінцевим зразком. [22]

5. Пошук джерел.

Коли у вас є прототип продукту, яким ви задоволені, настав час почати збирати матеріали та залучити партнерів, необхідних для виробництва. Це також називають побудовою вашого ланцюга поставок: постачальників, діяльності та ресурсів, необхідних для створення продукту та передачі його в руки клієнта.

6. Калькуляція витрат.

Після того, як дослідження, планування, створення прототипів і пошук джерел будуть виконані, ви повинні мати більш чітке уявлення про те, скільки буде коштувати виробництво вашого продукту. Калькуляція витрат — це процес бізнес-аналізу, в якому ви берете всю зібрану на даний момент інформацію та додаєте, яку буде ваша вартість проданих товарів, щоб ви могли визначити роздрібну ціну та валову маржу.

7. Комерціалізація.

На цьому етапі у вас є прибутковий і успішний продукт, готовий для світу. Останній крок у цій методології – вивести свій продукт на ринок! На цьому етапі команда розробників передасть кермо маркетингу для запуску продукту.[23]

Почніть зі створення електронної таблиці, у якій кожна додаткова вартість буде розбита як окрема позиція. Це має включати всю сировину, витрати на заводські налаштування, витрати на виробництво та витрати на доставку. Важливо врахувати доставку, імпортні збори та будь-які мита, які вам доведеться сплатити, щоб отримати кінцевий продукт у руки клієнта, оскільки ці збори

можуть мати значний вплив на ваші ціни на вартість, залежно від того, де ви виробляєте продукт. На малюнку Рис. 3.2 зображено процес навчання нейронної мережі.

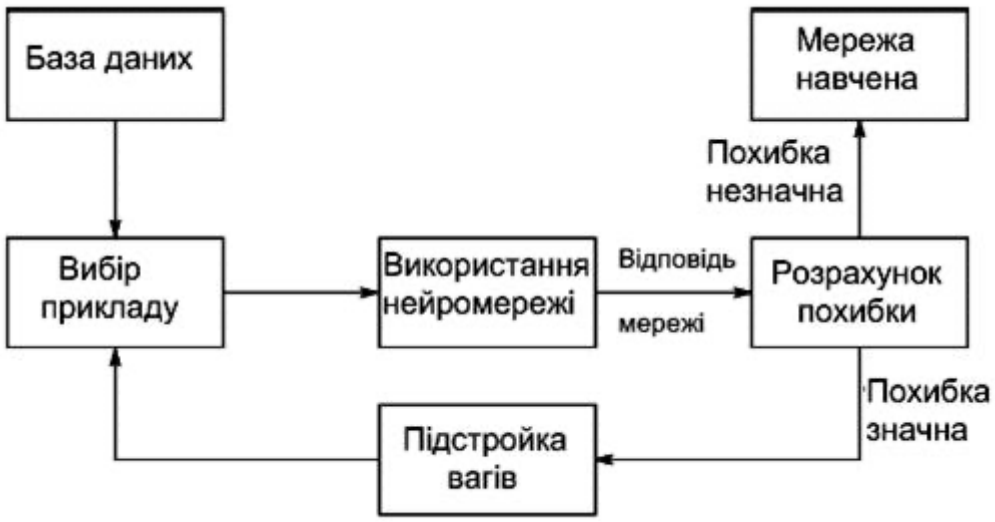

Рисунок 3.2 – процес навчання нейроної мережі

1.8.1 Розробка структури даних

Структура даних – це спосіб збору та організації даних таким чином, щоб ми могли ефективно виконувати операції з цими даними. Структури даних – це відтворення елементів даних у термінах певного зв’язку для кращої організації та зберігання.

Ми можемо організувати ці дані у вигляді запису на зразок Запису гравця, у якому буде вказано ім’я гравця та вік. Тепер ми можемо збирати та зберігати записи гравця у файлі чи базі даних як структуру даних.

Структура даних —спосіб організації та зберігання даних, щоб ефективно виконувати операції. Доступ, вставка, видалення, пошук і сортування даних є одними з основних операцій, які можна виконувати за допомогою структур даних. Не всі структури даних можуть ефективно виконувати ці операції, це призвело до розробки різних структур даних. Скажімо, вам потрібно знайти конкретну книгу в неорганізованій бібліотеці, це завдання займе величезну кількість часу. Подібно до того, як бібліотека організовує свої книги, нам

потрібно організувати наші дані так, щоб операції можна було виконувати ефективно.[24]

Для створення структури даних використовується попередньо визначений тип даних, наприклад Integer, Stings, Boolean. Використання типу даних залежить від вимог користувача та типу даних, які вони хочуть зберігати. Тепер давайте зануримося в структури даних, які використовуються в нашому повсякденному програмуванні, а також розглянемо підтримку структур даних для різних мов програмування.

Одновимірні масиви.

Базовою структурою даних, яку використовують у повсякденному програмуванні, є масив. Масив може містити фіксовану кількість контейнерів для зберігання даних, і над цими даними можна виконувати операції відповідно до потреб користувача.

Багатовимірні масиви.

Багатовимірний масив — це просто розширення звичайного масиву, де ми визначаємо масив із рядком контейнерів. У випадку багатовимірного масиву ми маємо рядки, а також стовпці. Щоб отримати доступ до індексу, нам потрібні два числа.

Динамічні масиви.

Більшість мов програмування дозволяють визначати динамічні масиви, що означає, що пам’ять виділяється під час виконання програми. Розглянемо це таким чином, ви визначили масив з 10 індексами, але вам потрібно лише 3 індекси, отже, решта 7 індексів знищені, що споживає додаткову пам’ять. Динамічні масиви дають можливість розподілити пам'ять відповідно до вимог програми.

Підтримка масивів для різних мов програмування.

Java: Є кілька класів для масивів, але найвідомішим є клас ArrayList.

C++: - Вектори використовуються для представлення динамічних масивів у C++. Його можна реалізувати за допомогою std::vector.

Python: Python має новий тип даних, відомий як список, як і Boolean і Integers. Тип даних списку можна використовувати для динамічного визначення масивів.[25]

Однозв’язаний список.

Пов’язаний список — це набір вузлів, які з’єднані за допомогою посилань. Пов’язаний список містить вузол, який зберігає елементи даних і адресу наступного вузла. Перший вузол зазвичай називають головним вузлом, а останній — хвостовим. Покажчик головного вузла вказує на наступний вузол, а вказівник хвостового вузла вказує на Null.

Динаміка цієї структури даних полегшує додавання або видалення вузлів. Щоб додати/видалити вузол, вам просто потрібно відстежити попередній вузол і вузол після нього і відповідно налаштувати покажчики.

Двозв’язаний список.

Двозв’язаний список мало чим відрізняється від однозв’язаного списку, єдине, що їх відрізняє, — це вказівник на попередній вузол. Зображення нижче може дати вам коротке уявлення про те, як повинна виглядати структура.

У випадку двозв’язаного списку попередній вказівник головного вузла вказує на Null, а наступний покажчик хвостового — на Null. [26] Попередній покажчик полегшує перехід в будь-якому напрямку. Отже, додавання та видалення вузлів стає дуже простим, все, що вам потрібно зробити, це відстежувати попередній і наступний вузли та відповідно налаштовувати вказівник.

Круговий зв’язаний список.

Круговий зв’язаний список – це зв’язаний список, де всі вузли з’єднані, щоб утворити коло. Головного чи хвостового вузла немає. Перевага такого типу зв’язаного списку полягає в тому, що будь-який вузол може бути використаний як відправна точка.

Порівняння між масивами та зв'язаним списком.

Список — це відмінний від масиву тип структури даних. Замість того, щоб зберігати дані в шматках пам'яті, масив потребує безперервного простору пам'яті. Якщо додати елемент до масиву, це може змінити розташування пам’яті

всього масиву, але дані залишаться неушкодженими. Масив дає вам номер індексу, отже, ви можете отримати до нього прямий або послідовний доступ. Тоді як для доступу до будь-якого члена даних потрібно пройти через весь список.[27]

1.8.2 Написання програмного продукту

Для початку роботи програми підключимо наступні бібліотеки.

```
#include <ctime>
#include <math.h>
#include "stdlib.h"
#include"myNeuro.h"
#include <msclr\marshal_cppstd.h>
```

Далі створюємо клас myNeuro

```
class myNeuro
{
public:
    myNeuro();

    struct nnLay {
        int in;
        int out;
        float** matrix;
        float* hidden;
        float* errors;
        int getInCount() { return in; }
        int getOutCount() { return out; }
        float** getMatrix() { return matrix; }
        void updMatrix(float* enteredVal)
        {
            for (int ou = 0; ou < out; ou++)
            {

                for (int hid = 0; hid < in; hid++)
```

```
                {
                    matrix[hid][ou] += (learnRate * errors[ou]
* enteredVal[hid]);
                }
                matrix[in][ou] += (learnRate * errors[ou]);
            }
        };
        void setIO(int inputs, int outputs)
        {
            srand(time(NULL));
            in = inputs;
            out = outputs;
            hidden = (float*)malloc((out) * sizeof(float));

            matrix = (float**)malloc((in + 1) * sizeof(float));
            for (int inp = 0; inp < in + 1; inp++)
            {
                matrix[inp]     =     (float*)malloc(out     *
sizeof(float));
            }
            for (int inp = 0; inp < in + 1; inp++)
            {
                for (int outp = 0; outp < out; outp++)
                {
                    matrix[inp][outp] = randWeight;
                }
            }
        }
        void makeHidden(float* inputs)
        {
            for (int hid = 0; hid < out; hid++)
            {
                float tmpS = 0.0;
                for (int inp = 0; inp < in; inp++)
                {
                    tmpS += inputs[inp] * matrix[inp][hid];
                }
```

```
                tmpS += matrix[in][hid];
                hidden[hid] = sigmoida(tmpS);
            }
        };
        float* getHidden()
        {
            return hidden;
        };
        void calcOutError(float* targets)
        {
            errors = (float*)malloc((out) * sizeof(float));
            for (int ou = 0; ou < out; ou++)
            {
                errors[ou] = (targets[ou] - hidden[ou]) *
sigmoidasDerivate(hidden[ou]);
            }
        };
        void calcHidError(float* targets, float** outWeights,
int inS, int outS)
        {
            errors = (float*)malloc((inS) * sizeof(float));
            for (int hid = 0; hid < inS; hid++)
            {
                errors[hid] = 0.0;
                for (int ou = 0; ou < outS; ou++)
                {
                    errors[hid] += targets[ou] *
outWeights[hid][ou];
                }
                errors[hid] *= sigmoidasDerivate(hidden[hid]);
            }
        };
        float* getErrors()
        {
            return errors;
        };
        float sigmoida(float val)
```

```
        {
            return (1.0 / (1.0 + exp(-val)));
        }
        float sigmoidasDerivate(float val)
        {
            return (val * (1.0 - val));
        };
    };
    string GetStr11() {
        return to_string(atof(str13.c_str()) * 100) + "%";
    }
    string GetStr12() {
        return to_string(atof(str14.c_str()) * 100) + "%";
    }
    void feedForwarding(bool ok);
    void backPropagate();
    void train(float* in, float* targ);
    void query(float* in);

private:
    struct nnLay* list;
    int inputNeurons;
    int outputNeurons;
    int nlCount;
    string str11="", str12="",str13 = "", str14 = "";
    int count = 0;
    float* inputs;
    float* targets;
};
```

Далі були описані декілька методів в класі

```
void myNeuro::backPropagate()
{
    //-------------------------------ERRORS-----CALC---------
    list[nlCount - 1].calcOutError(targets);
    for (int i = nlCount - 2; i >= 0; i--)
```

```
        list[i].calcHidError(list[i + 1].getErrors(), list[i +
1].getMatrix(),
            list[i     +     1].getInCount(),     list[i     +
1].getOutCount());
    //-------------------------------UPD-----WEIGHT---------
    for (int i = nlCount - 1; i > 0; i--)
        list[i].updMatrix(list[i - 1].getHidden());
    list[0].updMatrix(inputs);
}

void myNeuro::train(float* in, float* targ)
{
    inputs = in;
    targets = targ;
    feedForwarding(true);
}

void myNeuro::query(float* in)
{
    inputs = in;
    feedForwarding(false);
}
```

1.9 Опис функціоналу

Переходячи за посиланням до файлу Mail.exe (Рис 3.3)

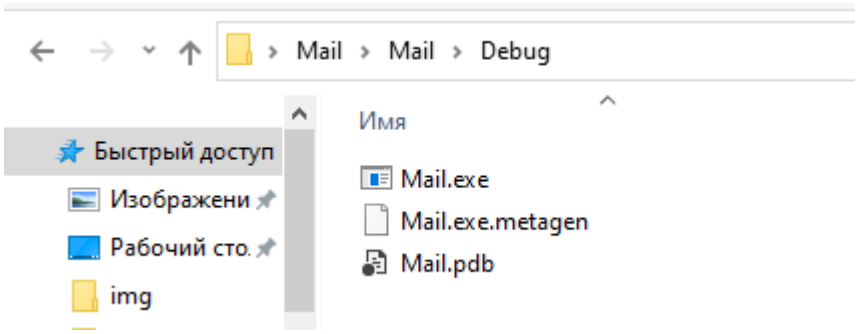

Рисунок 3.3 – Розташування програми

Перед нами з'явиться наступна картина (Рис. 3.4)

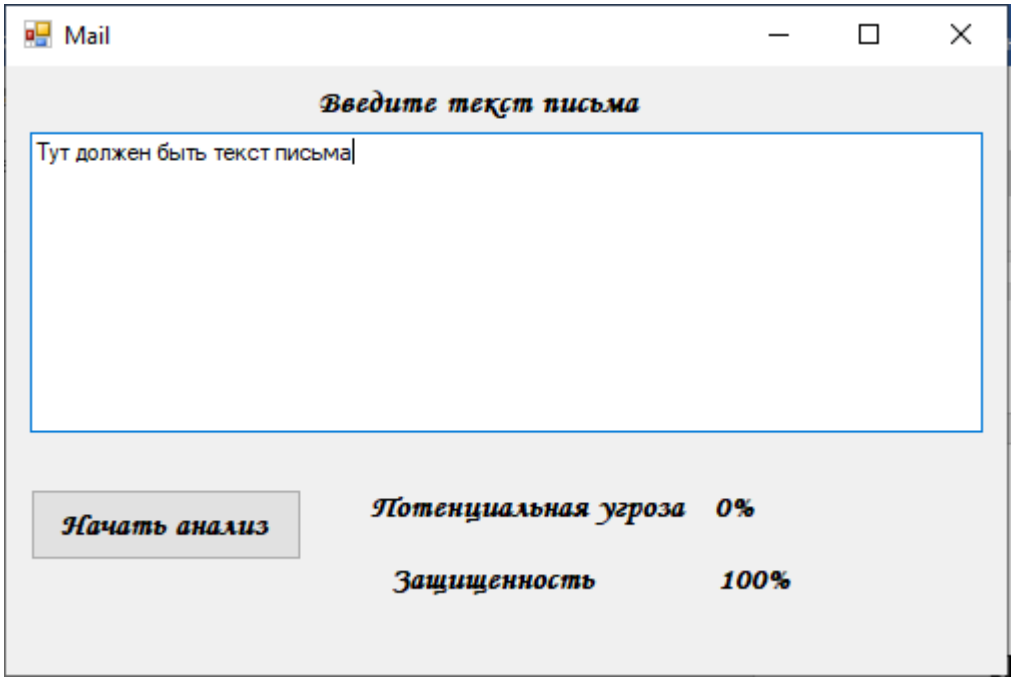

Рисунок 3.4 – Стартове вікно програми

Заповнивши змістом відповідне поле, та натиснувши кнопку «Почати аналіз», відбудеться аналіз (Рис. 3.5)

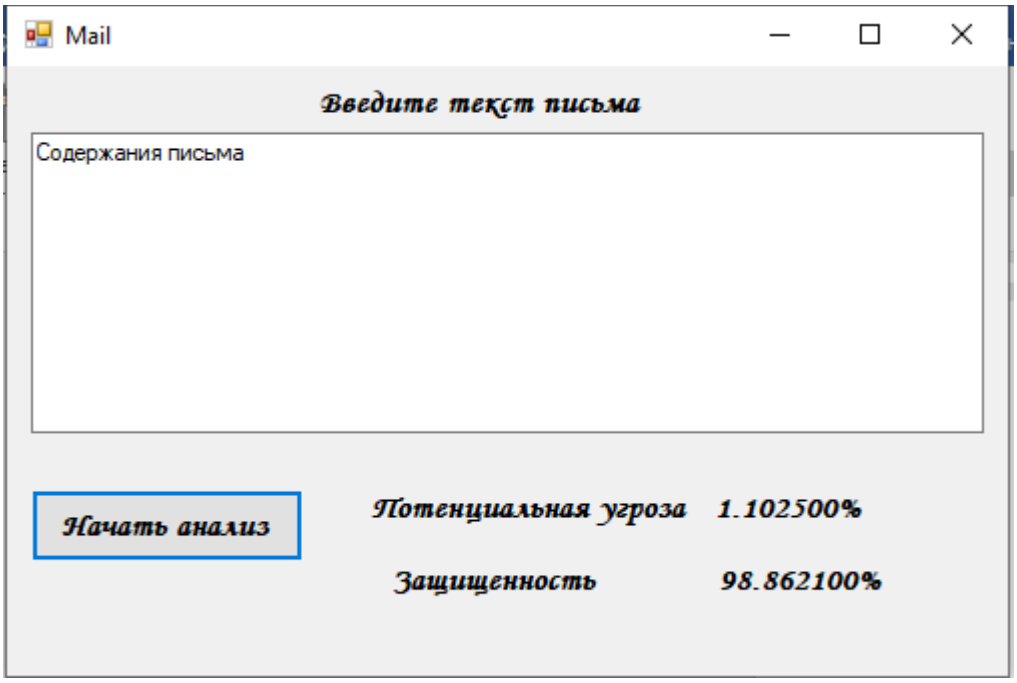

Рисунок 3.5 – Аналіз тексту

Інший приклад аналізу тексту. (Рис 3.6)

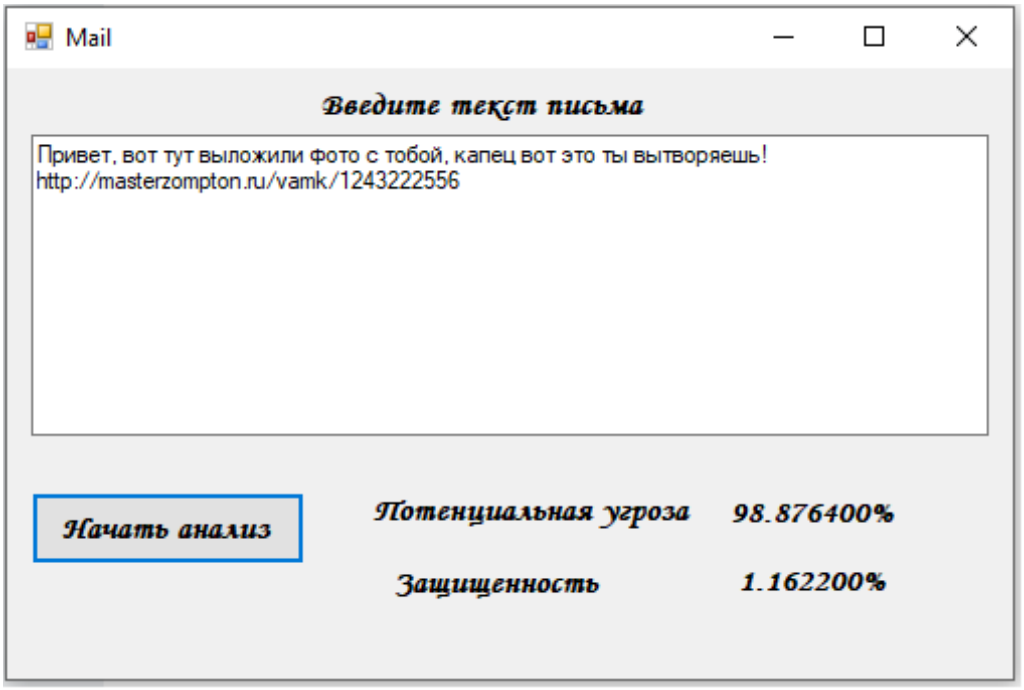

Рисунок 3.6 – Аналіз тексту

Якщо текст немає потенціальних загроз, то перед нами з’явиться наступна картина (Рис. 3.7)

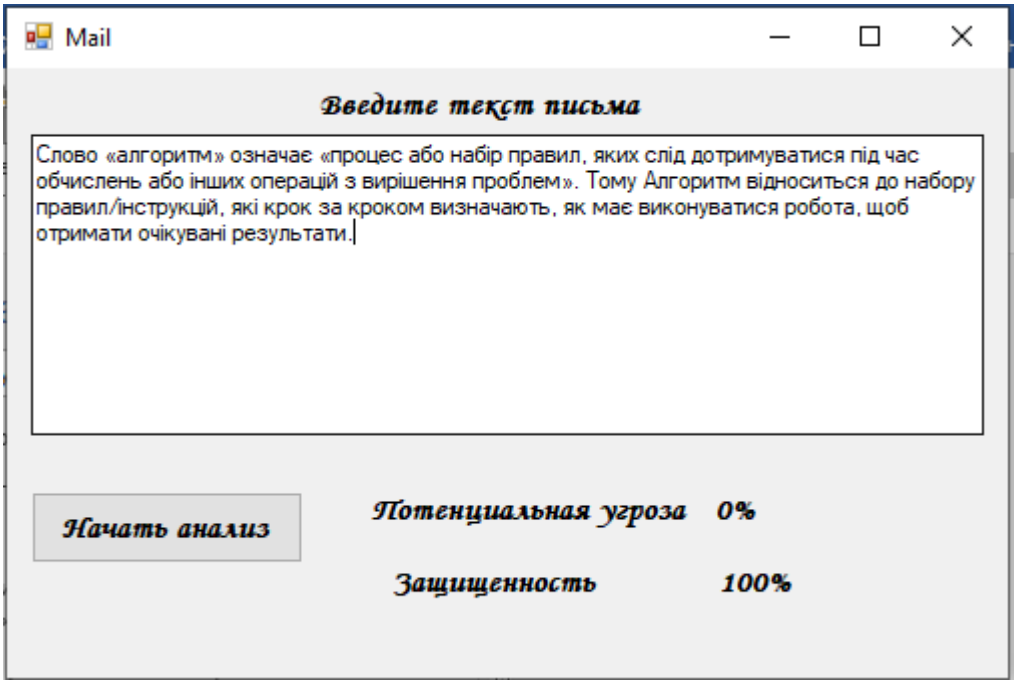

Рисунок 3.7 – Текст без загроз

Якщо текст має загрозу результат буде наступний (Рис 3.8)

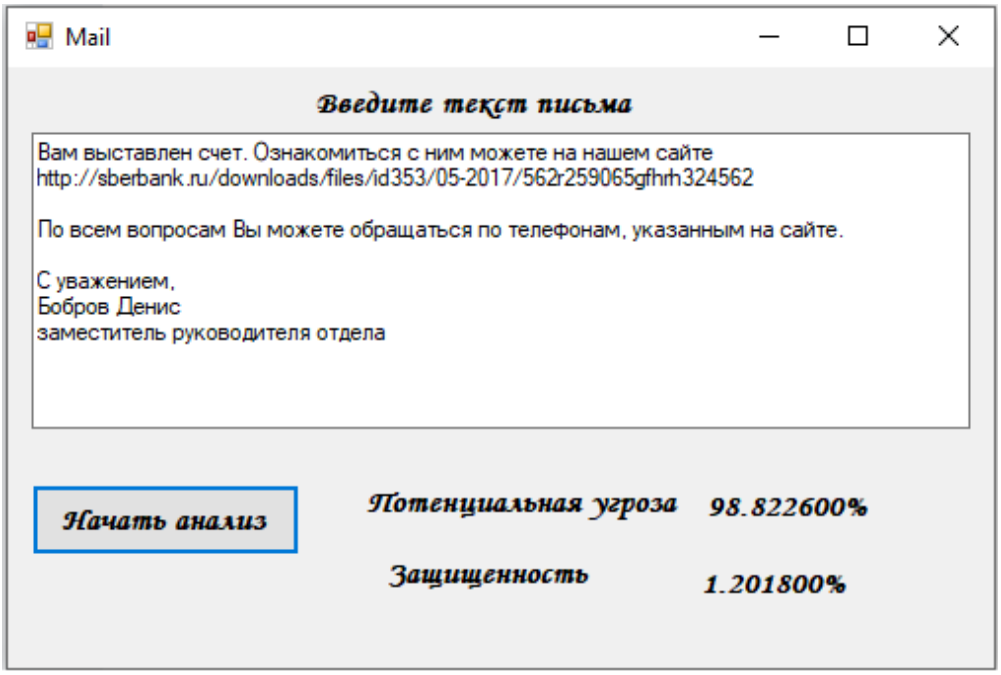

Рисунок 3.8 – Текст з загрозами

1.10 Тестування

Під час тестування було виявлені деякі помилки, та виправлені у програмі. (Рис. 3.9)

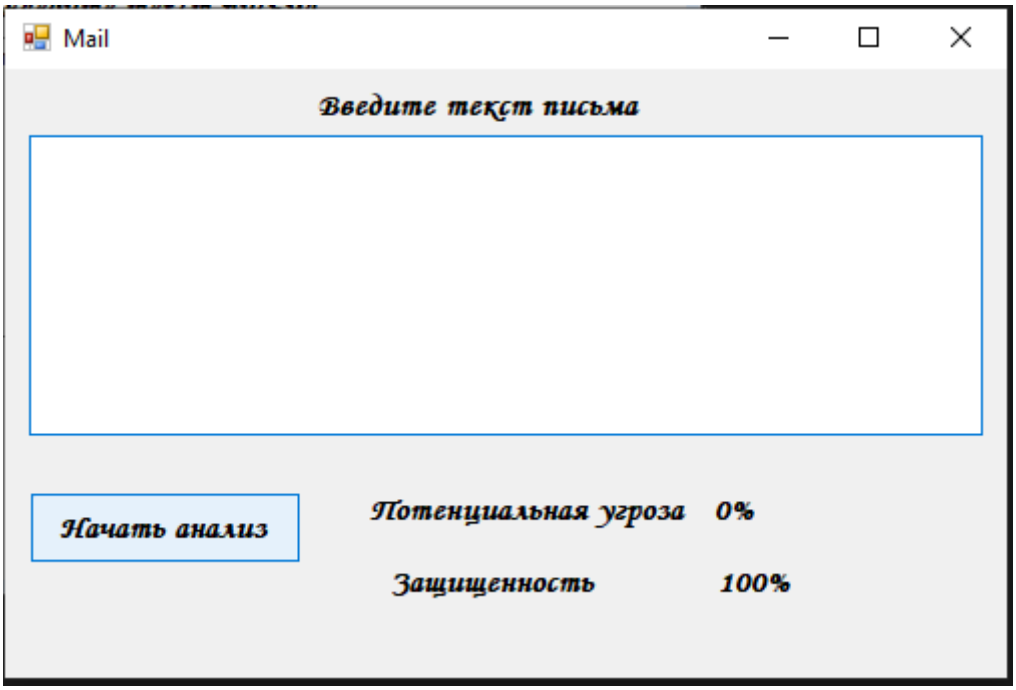

Рисунок 3.9 – Тестування(пусте поле)

Також було протестовано програму на різноманітних прикладах (Рис. 3.10)

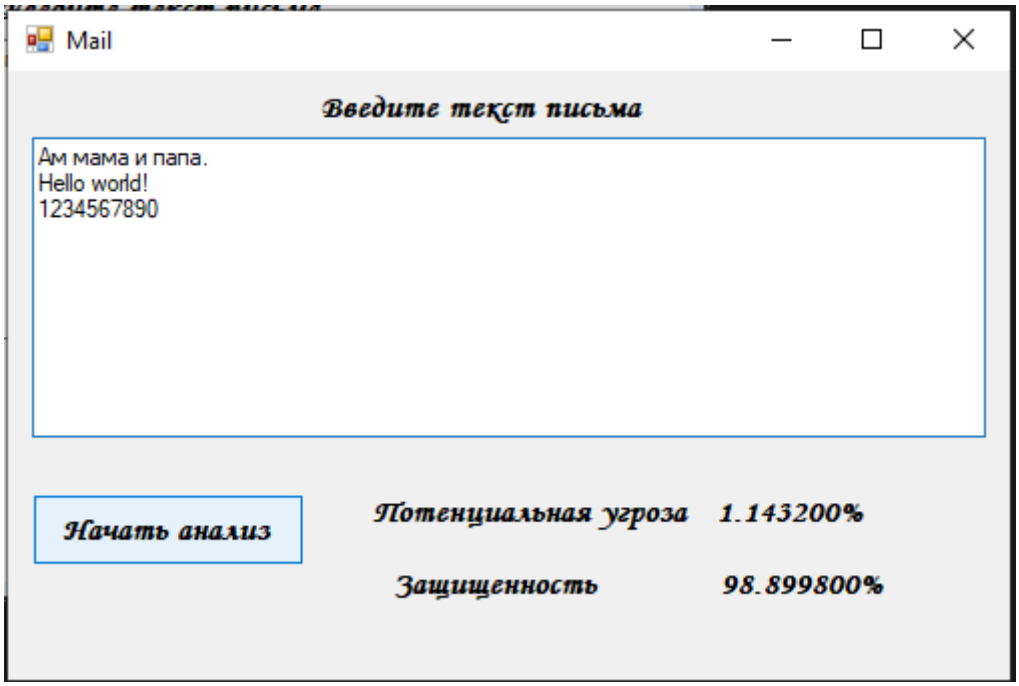

Рисунок 3.10 – Тестування програми

Загалом, всі помилки які виникали під час тестування, одразу ж виправлялися.

1.11 Висновки до розділу

В цьому розділі розповідається про етапи розробки та створення програмного додатку. Описується розроблений алгоритм та за якими етапами відбувалося написання програмного коду. Розповідається про розробку структури даних та описується функціонал програми. Також розповідається як відбувалося тестування, та які помилки виникали під час тестування.

ВИСНОВКИ

У цьому дослідженні ми проаналізували наявну літературу в базі даних в інтернеті, посилаючись на основні моделі прогнозування майбутніх атак. Аналіз зосереджувався на прогнозах попиту на звичайні та швидко доступних атаках. Ці моделі сприяють покращенню виробничих планів організацій, зменшенню кількості атак на комп’ютери.

Історичні дані шляхом аналізу часових рядів і невизначених змінних, які збирають вплив зовнішніх факторів, є основними вхідними змінними, які використовуються в аналізованих моделях прогнозу. Крім того, методи м’яких обчислень та часових рядів є найбільш прогнозованими моделями, які використовуються в літературі, а найбільш часто використовуваними методами для розрахунку їх точності є середня абсолютна відсоткова помилка та середня квадратична помилка кореня.

Завершуючи роботу про захист від уражень, доцільно зробити прогноз щодо методів, які будуть використовуватися в короткостроковому та довгостроковому майбутньому.

Як ми вже говорили, передбачити найближче майбутнє не так вже й складно, оскільки довгострокові тенденції не змінюються за одну ніч. Багато з описаних методів знаходяться лише на ранніх стадіях застосування, але все ж ми очікуємо, що більшість методів, які будуть використовуватися в наступні п’ять років, будуть обговорюватися тут, можливо, у розширеній формі.

Постійна тенденція зниження вартості обчислень комп’ютера разом із спрощеннями обчислень зроблять такі методи, як метод Бокса-Дженкінса, економічно доцільними, навіть для деяких додатків для контролю запасів. Пакети комп'ютерного програмного забезпечення для статистичних методів і деякі загальні моделі також стануть доступними за номінальною ціною.

В даний час більшість короткострокових прогнозів використовує тільки статистичні методи з малою якісною інформацією. Якщо використовується якісна інформація, вона використовується лише зовнішньо і не включається

безпосередньо в обчислювальну рутину. Ми прогнозуємо зміну систем загального прогнозування, де кілька методів пов’язані разом із систематичною обробкою якісної інформації.

Економетричні моделі будуть ширше використовуватися протягом наступних п’яти років, при цьому більшість великих компаній розроблятимуть та вдосконалюють економетричні моделі своїх основних підприємств. Маркетингові імітаційні моделі для нових продуктів також будуть розроблені для продуктів більшого обсягу з системами відстеження для оновлення моделей та їх параметрів. Евристичне програмування забезпечить засіб уточнення моделей знаходження шкідливого контенту.

Хоча деякі компанії вже розробили власні моделі витрат-випусків у тандемі з державними даними-випуском і статистичними прогнозами, пройде ще п'ять-десять років, перш ніж моделі витрат-випусків будуть ефективно використовуватися більшістю великих корпорацій.

Проте протягом п’яти років ми побачимо широке використання систем «людина-машина», де статистичні, причинно-наслідкові та економетричні моделі програмуються на комп’ютерах, а люди часто взаємодіють. У міру того, як ми набуваємо довіри до таких систем, щоб менше повідомлялося про винятки, людське втручання зменшуватиметься. В основному, комп'ютеризовані моделі будуть виконувати складні обчислення, а люди будуть більше служити генераторами ідей і розробниками систем. Наприклад, ми будемо вивчати динаміку ринку та встановлювати більш складні взаємозв’язки між фактором, що прогнозується, та факторами системи прогнозування.

Надалі споживчі імітаційні моделі стануть звичайним явищем. Моделі передбачатимуть поведінку споживачів та прогнозують їхню реакцію на різні маркетингові стратегії, такі як ціноутворення, рекламні акції, впровадження нових продуктів та конкурентні дії. Імовірнісні моделі будуть часто використовуватися в процесі прогнозування.

Нарешті, більшість комп’ютеризованих прогнозів буде стосуватися аналітичних методів, описаних у цій роботі. Комп’ютерні додатки будуть переважно в усталених і стабільних продуктах бізнесу. Безсумнівно, нові

аналітичні методики будуть розроблені для прогнозування нових продуктів, але проблема точного прогнозування різних факторів нового продукту, таких як продажі, прибутковість і, можливо, буде тривати протягом принаймні 10-20 років і, ймовірно, набагато довше. тривалість життєвого циклу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Кормен Т.,Лейзерсон Ч.,Ривест Р. Алгоритмы: построение и анализ, 2001

2. Окулов С. М. Программирование в алгоритмах. - М.:БИНОМ. 2002

3. Д.Э. Кнут. Искусство программирования, том 3. Пер. С англ. М.: Издательский дом “Вильямс”, 2007. (и др. издания)

4. Буч Г., Объектно-ориентированный анализ и проектирование с примерами приложений. Издательство Вильямс, С.-Петербург, 2008.

5. Бертран Мейер. Объектно-ориентированное конструирование программных систем. -М.:Русская редакция, 2005

6. Лафоре Р. Объектно-ориентированное программирование в C++. С.-Петербург: Питер, 2006

7. Синтес А. Освой самостоятельно объектно-ориентированное программирование за 21 день. Москва4 С.-Петербург: Вильямс, 2002.

8. Річард Барасс: Towards a theory of innovation in services, -1986, С. -183.

9. Хазієва А. М. Сучасна державна статистика // Тенденції і перспективи розвитку статистичної науки та інформаційних технологій. Збірка наукових статей, -2013, -С. 94-98.

10. Стеценко, І.В. Моделювання систем: навч. посіб. [Електронний ресурс, текст] / І.В. Стеценко ; М-во освіти і науки України, Черкас. держ. технол. унт. – Черкаси : ЧДТУ, 2010. – 399 с

11. Навроцкий, А.А. Основы алгоритмизации и программирования в среде Visual C++: учеб.-метод. пособие / А. А. Навроцкий. – Минск : БГУИР, 2014. – 160 с.

12. Шилдт, Г. C++ Базовый курс, 3-е издание / Г. Шилдт. Пер. с англ. – М.: Издательский дом «Вильямс», 2015. – 624 с.

13. Макконнелл, С. Совершенный код. Мастер-класс / С. Макконнелл. Пер. с англ. – М. : Издательство «Русская редакция», 2010. — 896 стр.

14. Документация по Visual Studio [электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/visualstudio/ide/?view=vs-2017 (дата звернення 20.11.2021)

15. Todd Hoff C++ Coding Standard [электронный ресурс]. Режим доступа: http://www.possibility.com/Cpp/c++_coding_standards.pdf (дата звернення 21.11.2021)

16. Google C++ Style Guide [электронный ресурс]. Режим доступа: https://google.github.io/styleguide/cppguide.html (дата звернення 01.10.2021) 7. ГОСТ 19.701-90 Схемы алгоритмов, программ, данных и систем. 8. Доманов А.Т. Стандарт предприятия СТП 01-2017 / А.Т. Доманов, Н.И. Сорока. – Минск : БГУИР, 2017. – 169 с

17. Введение в языки программирования C и C++ [электронный ресурс] / режим доступа: http://www.intuit.ru/studies/courses/1039/231/info (дата звернення 09.12.2021)

18. Верификация программного обеспечения [электронный ресурс] / режим доступа: http://www.intuit.ru/studies/courses/1040/209/info (дата звернення 09.12.2021)

19. Инструменты, алгоритмы и структуры данных [электронный ресурс] / режим доступа: http://www.intuit.ru/studies/courses/683/539/info (дата звернення 20.11.2021)

20. Мюссер, Д. C++ и STL : справочное руководство / Д. Мюссер, Ж. Дердж, А. Сейни. – М.: Изд. Дом "Вильямс", 2010.

21. Саттер, Г. Стандарты программирования на C++: серия "C++ In-Depth" / Г. Саттер, А. Александреску. – М.: Изд. Дом "Вильямс", 2008.

22. Седжвик, Р. Алгоритмы на C++ / Р. Седжвик. – М.: Изд. Дом "Вильямс", 2010.

23. Шилдт, Г. C++: методики программирования Шилдта / Г. Шилдт. – М.: Изд. Дом "Вильямс", 2008.

24. Дэвис, С. C++ для "чайников" / С. Дэвис. – 6-е изд. – М.: Изд. Дом "Вильямс", 2010.

25. Либерти, Дж. Освой самостоятельно C++ за 21 день / Дж. Либерти, Дж. Брэдли. – 5-е изд. – М.: Изд. Дом "Вильямс", 2010.

26. Страуструп, Б. Программирование с примерами на C++: принципы и практика / Б. Страуструп. – М.: Изд. Дом "Вильямс", 2010.

27. Шилдт, Г. Полный справочник по C++ / Г. Шилдт. – М.: Изд. Дом "Вильямс", 2010.