Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМІВ ОБЧИСЛЮВАЛЬНОГО ДІАГНОСТУВАННЯ ГЕНЕТИЧНИХ ЗАХВОРЮВАНЬ
RESEARCH AND SOFTWARE IMPLEMENTATION OF COMPUTATIONAL ALGORITHMS FOR GENETIC DISEASE DIAGNOSIS

Виконав(ла): студент(ка) 4 курсу, групи КНТ-122
Спеціальності 121 Інженерія програмного
(код і найменування спеціальності)
забезпечення

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

КИРИЛЕНКО М.Є.
(ПРІЗВИЩЕ та ініціали)
Керівник СУББОТІН С.О.
(ПРІЗВИЩЕ та ініціали)
Рецензент КОРОТКИЙ О.В.
(ПРІЗВИЩЕ та ініціали)

2026

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____Сергій СУББОТІН
“____” _____2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

КИРИЛЕНКА Михайла Євгеновича
(ПРІЗВИЩЕ, ім’я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація алгоритмів обчислювального діагностування генетичних захворювань.
Research and Software Implementation of Computational Algorithms for Genetic Disease Diagnosis

керівник проєкту (роботи) д.т.н., професор, СУББОТІН Сергій Олександрович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім’я, по батькові)
затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року № 139
2. Строк подання студентом проєкту (роботи) 10 червня 2026 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Розробка програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СУББОТІН С.О., професор		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 20 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Розробка архітектури програми.	2 тиждень	Розділ 2
4	Розробка програми.	3-4 тижні	Розділ 3
5	Експлуатація, тестування та експериментальне	5 тиждень	Розділ 4
	дослідження програми		
6	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
7	Нормоконтроль та рецензування.	7 тиждень	
8	Захист роботи.	8 тиждень	

Студент(ка) _____ Михайло КИРИЛЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Сергій СУББОТІН
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 77 с., 4 табл., 24 рис., 2 дод., 10 джерел.

ГЕНЕТИЧНІ ДАНІ, ГЕНЕТИЧНІ ЗАХВОРЮВАННЯ, ДІАГНОСТУВАННЯ, АНАЛІЗ, АЛГОРИТМ, КОНВЕЄР.

Об’єкт дослідження – процес інженерії програмного забезпечення програмного забезпечення для опрацювання та аналізу генетичних даних.

Предмет дослідження – програмні засоби та алгоритми для опрацювання та аналізу генетичних даних.

Мета роботи – прискорити процес діагностування генетичних захворювань.

Матеріали, методи та технічні засоби: структурне та об’єктно орієнтоване програмування, мови програмування Python, інтегровані середовища розробки програм Microsoft Visual Studio Code.

Результати. Створено програмну систему за допомогою якої виконується обчислювальний аналіз генетичних даних, яка реалізує алгоритми обробки секвенованих даних для подальшого виявлення генетичних варіантів з метою діагностування генетичних захворювань.

Висновок. Розроблено програмну систему для виявлення генетичних захворювань, яка автоматизує та зменшує час обробки з діагностуванням захворювань.

Галузь використання – діагностування генетичних захворювань в медичних закладах за допомогою секвенатора та обчислювальних систем.

Економічна ефективність. При використанні розробленої системи вартість діагностування генетичних захворювань для медичних закладів зменшиться.

ABSTRACT

Explanatory note to the dimploma qualifying work of the bachelor: 77 pages, 4 tables, 24 figures, 2 appendixes, 10 sources.

GENETIC DATA, GENETIC DISEASES, DIAGNOSIS, ANALYSIS, ALGORITHM, PIPELINE.

Object of research – the process of processing and analyzing genetic data.

Subject of research – software tools and algorithms for processing and analyzing genetic data.

Purpose of the work – to increase the efficiency of diagnosing genetic diseases by automating the process of data processing and analysis.

Materials, methods, and technical means: structured and object-oriented programming, the Python programming language, and the Microsoft Visual Studio Code integrated development environment.

Results. A software system has been developed that performs computational analysis of genetic data and implements algorithms for processing sequencing data in order to identify genetic variants for the diagnosis of genetic diseases.

Conclusion. A software system for detecting genetic diseases has been developed, which automates and reduces the time required for processing and diagnosis.

Field of application – diagnosis of genetic diseases in medical institutions using a sequencer and computational systems.

Economic efficiency. The use of the developed system will reduce the cost of diagnosing genetic diseases for medical institutions.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧОК

BAM – Binary Alignment/Map;

CLI – Command Line Interface;

CSV – Comma-Separated Values;

IDE – Integrated Development Environment;

NCBI – National Center for Biotechnology Information;

SAM – Sequence Alignment/Map;

SRA – Sequence Read Archive;

VCF – Variant Call Format;

WSL – Windows Subsystem for Linux.

ВСТУП

Опрацювання генетичних даних є одним із ключових інструментів сучасної медицини та обчислювальних методів аналізу даних. Воно широко застосовується для діагностування генетичних спадкових захворювань, аналізу мутацій у онкології, розробкою та аналізом лікарських засобів у фармацевтиці, а також для дослідження інфекційних захворювань. Використання таких методів дозволяє виявляти генетичні відхилення, що є основою для точного та безпечного діагностування, а також прийняття обґрунтованих рішень щодо подальшого лікування пацієнтів.

Через розвиток технологій з видобутку генетичних даних, зростає обсяг даних, що потребує ефективної обробки з подальшим аналізом. Для чого й використовують алгоритми та інструменти для роботи з даними, які дозволяють отримати результати, що будуть використанні для подальшого аналізу. Розвитком та дослідженням даних алгоритмів з інструментами, а також створенням нових займаються наукові організації та інститути, такі як Європейський Біоінформатичний Інститут та Інститут Броуда. Зазвичай ці методи розраховані на використання під конкретні задачі та певний обсяг даних. Крім того, усі процеси обробки та аналізу зазвичай розділені на окремі кроки, що ускладнює використання та збільшує вірогідність людської похибки, наприклад у виборі невідповідного інструменту, що може вплинути на кінцевий результат, а саме на аналіз та подальшу діагностику геному. У свою чергу через це втрачається час та ресурси на однієї роботи, які могли бути використанні більш ефективно для подальших аналізів та розробок. Процес для опрацювання даних

Об’єктом дослідження є процес інженерії програмного забезпечення програмного забезпечення для опрацювання та аналізу генетичних даних.

Предмет дослідження – програмні засоби та алгоритми для обробки та аналізу генетичних даних.

Мета роботи – прискорити процес діагностування генетичних захворювань шляхом автоматизації обробки та аналізу великих масивів даних.

Завдання дослідження – для досягнення поставленої мети у роботі вирішувалися такі завдання:

- оглянути сучасні методи та програмні засоби обробки та аналізу генетичних даних;
- визначити особливості існуючих алгоритмів обробки генетичних даних;
- обрати підходи та алгоритми для обробки та аналізу даних;
- реалізувати програмне рішення для автоматизації процесу аналізу;
- перевірити працездатність та програмне рішення.

Методикою дослідження є поетапна обробка генетичних даних із застосуванням алгоритмів аналізу послідовностей та їх подальшому використанні у обчислювальному діагностуванні генетичних захворювань. Робота матиме вигляд послідовного перетворення від сирих секвенованих даних до готової для аналізу генетичної варіації, яка буде використана для подальшого діагностування.

Спершу здійснюється отримання даних у форматі FASTQ, після чого використовується вирівнювання прочитань за еталоним геномом із використанням алгоритму вирівнювання Берроуза-Вілера, результатом чого стає файл SAM який містить інформацію про вирівнювання послідовностей по еталоному геномі. За для подальшої роботи SAM перетворюють у бінарний формат, що робить процес виявлення генетичних варіантів швидшим, результатом якого є файл формату VCF.

Діагностика гену людини на мутації виконується за допомогою порівняння з відомими базами генетичних даних, що дозволяє знайти зв’язок між виявленими мутаціями з генетичними захворюваннями.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Сучасний стан досліджуваної області

Аналіз генетичних даних – один із напрямів сучасної обробки великих обсягів інформації, що передбачає отримання, обробку та інтерпретацію даних з метою їх подальшого використання у прикладних задачах. Даний процес включає кілька взаємопов’язаних етапів, кожен з яких відіграє важливу роль у формуванні кінцевого результату.

Перший етап є отримання генетичних даних за допомогою спеціалізованих технологій. Результатом цього процесу є великі масиви сирих даних, що потребують подальшої обробки. З розвитком технологій спостерігається зменшення вартості отримання даних, наведено на рисунку 1.1, що призводить до зростання їх обсягів, що наведено на рисунку 1.2 [1]-[2].

Генетичні дані потребують впорядкування, зберігання та подальшої обробки для отримання корисної інформації, а через постійне зростання обсягів даних ускладняється процес їх обробки та аналіз. Значні обсяги інформації вимагають використання спеціалізованих підходів до організації обчислювальних процесів, а також засобів для їх автоматизації. Крім того, обробка таких даних передбачає використання різних алгоритмів та інструментів, що виконують окремі етапи обробки та аналізу, через що виникає інша проблема – не цілісна робота.

У зв’язку з цим випливає необхідність поєднання різних інструментів та алгоритмів у єдиний узгоджений процес, що забезпечує послідовне виконання всіх етапів обробки даних. Відсутність такої узгодженості може призвести до труднощів у передачі даних між етапами та затримки у виконанні обчислень через неузгодженість між ними.

Основні поняття, пов’язані з обробкою та аналізом генетичних даних:

- генетичні дані – інформація про людину, що містить відомості відомості про будову та функціонування організму;
- секвенування – процес видобутку генетичних даних;

– сирі дані – первинна інформація, отримана в результаті секвенування, яка потребує обробки;

– обробка даних – сукупність методів та операцій, спрямованих на очищення, перетворення та підготовку даних до аналізу;

– аналіз генетичних даних – процес дослідження оброблених даних з метою виявлення закономірностей, змін або особливостей.

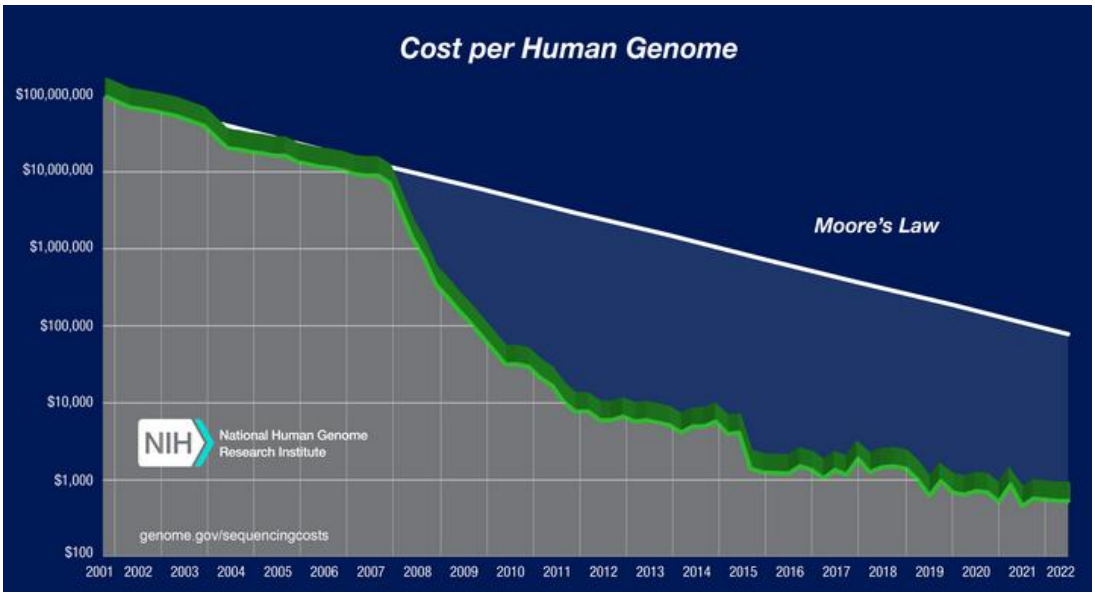

Рисунок 1.1 – Зменшення вартості секвенування [1]

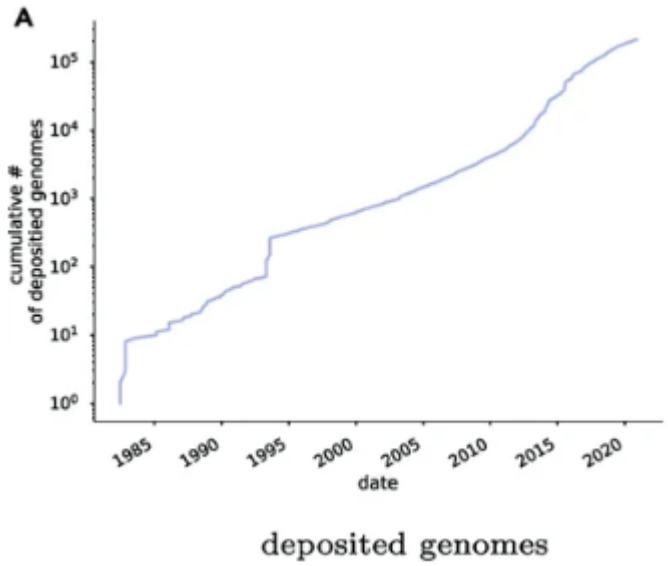

Рисунок 1.2 – Зростання обсягів генетичних даних [2]

1.2 Огляд існуючих рішень проблеми

1.2.1 Інструмент Fastq2vcf

Fastq2vcf – програмний конвеєр для аналізу даних секвенування. Даний інструмент орієнтований на автоматизацію процесу перетворення сирих даних у результаті аналізу, що значно спрощує виконання обчислювальних задач, блок-схема надана на рисунку 1.3 [3].

Fastq2vcf забезпечує інтеграцію декількох інструментів обробки даних у межах єдиного процесу. Це дозволяє виконувати послідовність операцій без необхідності ручного запуску кожного окремого етапу. Однією з ключових особливостей є простота у використанні, програма сама викликає команди для обчислення, що знижує вимоги до користувача.

Інструмент використовує декілька алгоритмів для аналізу результати яких об’єднані для підвищення узгодженості та надійності отриманих даних. Зокрема, формується як набір результатів для кожного окремого алгоритму, так і результат, отриманий на їх перетині.

Fastq2vcf дає можливість налаштування. Користувач може змінювати параметри обробки відповідно до поставленої задачі, а також розширювати функціональність шляхом додавання нових інструментів. А завдяки автоматичному формуванню та збереженню команд виконання забезпечується можливість повторного запуску обчислень з однаковими параметрами, що є важливим аспектом при роботі з великими обсягами даних. Використання програмного засобу Fastq2vcf передбачає підготовку вхідних даних та налаштування конфігураційний файлів, спершу заповнюється txt файл з вхідними даними і після чого генеруються скрипти, які будуть виконувати роботу.

Водночас, незважаючи на високий рівень автоматизації обчислювального конвеєра, Fastq2vcf не забезпечує повноцінного керування виконанням процесів. Зокрема, відсутні засоби моніторингу та автоматичного

відновлення обчислень у разі помилок, що ускладнює використання інструменту при обробці даних.

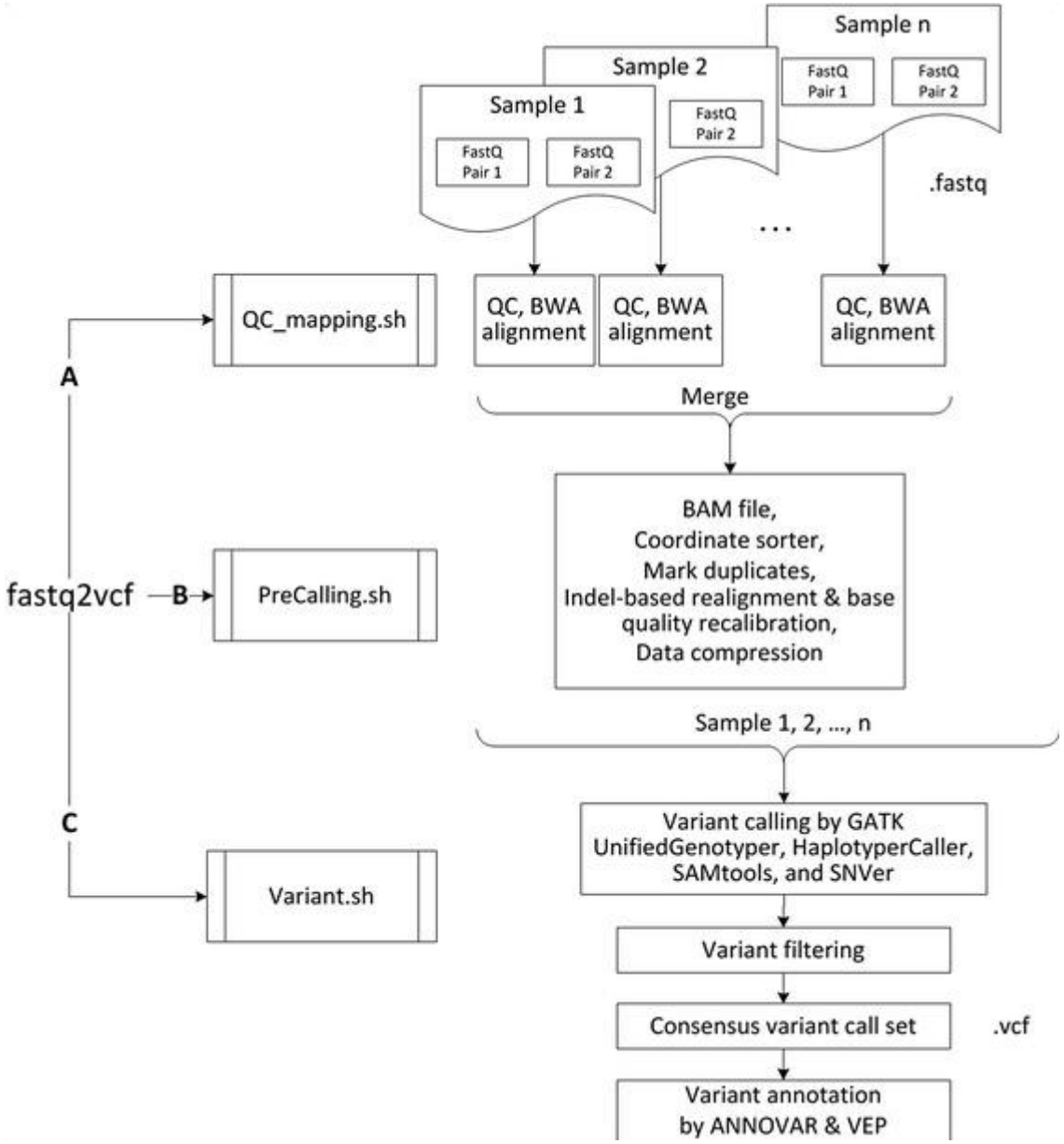

Рисунок 1.3 – Блок-схема Fastq2vcf [3]

1.2.2 Інструмент SeqMule

SeqMule – програмний конвеєр для автоматизованої обробки великих наборів даних, який призначений для виконання повного циклу аналізу сирих даних до отримання очікуваного результату [4]. Загальна схема роботи включає етапи попередньої обробки даних, виконання алгоритмічного аналізу

та отримання підсумкового результату, блок-схема зображена на рисунку 1.4 [4].

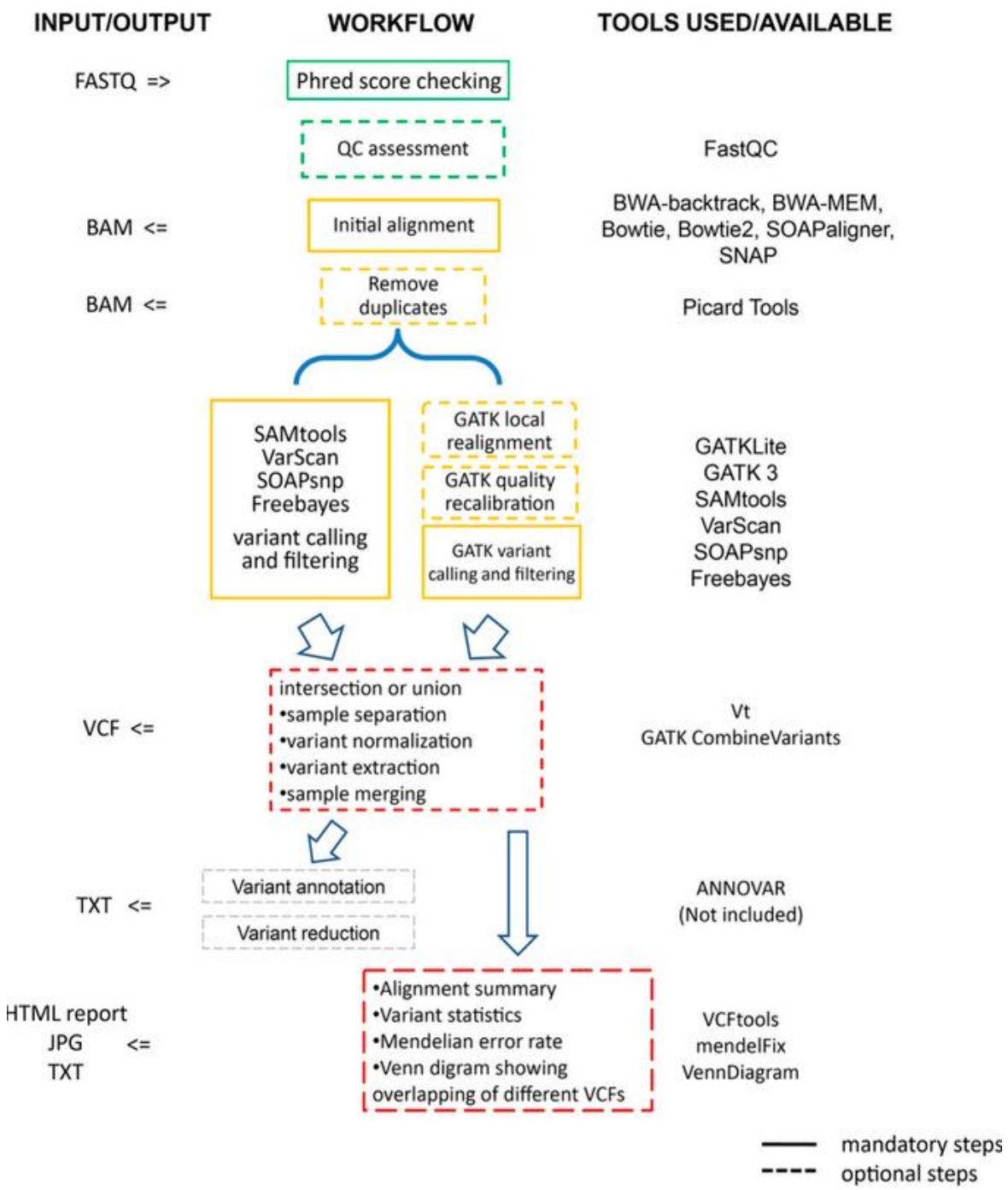

Рисунок 1.4 – Блок-схема SeqMule [4]

SeqMule забезпечує інтеграцію великої кількості окремих програмних інструментів у межах єдиного автоматизованого процесу. Це дозволяє виконувати послідовність обчислювальних операцій без необхідності ручного

запуску кожного окремого етапу, що зменшує складність роботи для користувача та зменшує ймовірність помилок при виконанні складних обчислювальних задач.

Особливістю SeqMule є використання кількох алгоритмів обробки на одному й тому ж етапі разом з різними інструментами. Отримані результати різних алгоритмів з інструментами порівнюються та суміжні поєднують, що дозволяє отримати більш точний результат. Але через таких підхід для виконання такого завдання зростають вимоги до апаратного забезпечення. Використання декількох алгоритмів з інструментами призводить до збільшення часу виконання та ускладнення процесу обчислень.

SeqMule підтримує додаткові функції, включаючи автоматичне додавання інформації до результатів, генерацію статистичних звітів та створення HTML-репортів. Формує логи для кожного етапу виконання, що забезпечує можливість відстежування процесу та діагностики помилок у разі їх виникнення. Використання програмного засобу передбачає налаштування конфігураційного файлу у якому задаються вхідні дані, обрані алгоритми разом з параметрами виконання, після чого запуск конвеєра здійснюється за допомогою командного інтерфейсу, результатом якого буде наданий у вигляді структурованого файлу разом зі звітом.

Отже, особливість SeqMule використання паралельного обчислення у обробці з аналізом даних з подальшим об’єднанням результатів задля підвищення точності отриманих даних, з чим приходить додаткове навантаження на апаратне забезпечення для підтримання виконання декількох робіт одночасно.

1.2.3 Порівняння існуючих програмних засобів

Для розробки власного програмного засобу потрібно провести порівняльний аналіз сучасних рішень. Завдяки якому можна виявити сильні та слабкі сторони, визначити основні обмеження, які можна вдосконалити.

Будуть порівнюватись існуючі інструменти, такі як Fastq2vcf та SeqMule, що реалізовують автоматизовані конвеєри обробки секвенованих даних. Порівняння буде відбуватися за такими критеріями як рівень автоматизації, складність використання, моніторинг процесів, вимоги до ресурсів, кількість алгоритмів, налаштування.

Обранні критерії для порівняння зумовлені специфікою задач обробки геномних даних, які характеризуються великими обсягами інформації та високими вимогами до точності результатів. Вимоги до обчислювальних ресурсів визначають можливість використання інструменту та безпосередньо навантаження на різних апаратних засобах. Рівень автоматизації та наявність засобів моніторингу процесів є критичним для забезпечення стабільності виконання обчислень, особливо при роботі з довготривалими задачами. Використання декількох алгоритмів впливають на точність і надійність аналізу. Критерій складності використання характеризує рівень зручності роботи з інструментом та зусилля для налаштування та запуску. Налаштування характеризує собою гнучкість, що впливає на можливість адаптації під різні задачі. В таблиці 1.1 наведено порівняння.

Таблиця 1.1 – Порівняння функціональності програмних засобів

Критерій	Fastq2vcf	SeqMule
Автоматизація	Часткова	Висока
Складність використання	Середня	Середня
Кількість алгоритмів	Невелика	Значна
Моніторинг процесів	Відсутній	Логи

Продовження таблиці 1.1

Критерій	Fastq2vcf	SeqMule
Вимоги до ресурсів	Помірні	Високі
Налаштування	Середнє	Високе

Порівняння наведені у таблиці 1.1, показують, що розглянуті інструменти мають суттєві відмінності за рядом критеріїв. Fastq2vcf характеризується частковою автоматизацію, оскільки забезпечує виконання основних етапів обробки даних, проте немає вбудованих засобів моніторингу та керуванням процесами. У свою чергу, SeqMule має вищій рівень автоматизації за рахунок підтримки логування, що дозволяє відстежувати процес виконання та впроваджує інтеграцію паралельного обчислення, що забезпечує підтримання більшої кількості інструментів на одному етапі обробки та аналізу. Водночас із цим це призводить до збільшення апаратних вимог, особливо порівнюючи з Fastq2vcf. SeqMule має більш гнучке налаштування, через більшу кількість інструментів, що дозволяє робити більш точний та детальний аналіз, але через це стає важчим у використанні, оскільки користувач буде більше навантажений інформацією, коли Fastq2vcf є простішим у використанні, проте має обмежені можливості.

1.3 Опис основних вимог

1.3.1 Формування вимог на основі аналізу існуючих рішень

На основі проведеного порівняльного аналізу програмних засобів Fastq2vcf та SeqMule визначимо ключові характеристики, які доцільно використати при розробці власного програмного засобу.

Fastq2vcf характеризується відносною простотою використання та помірними вимогами до обчислювальних ресурсів, що забезпечує його доступність до ширшого кола користувачів. Тому варто враховувати

спрощений підхід, а саме до налаштування та кількості використаних засобів для обробки з аналізом даних.

SeqMule маж високий рівень автоматизації обробки даних, підтримує значну кількість алгоритмів та забезпечує можливість моніторингу виконання за рахунок логування, що дає контроль над виконанням довготривалих процесів.

Враховуючи переваги цих двох засобів, при розробці програмного засобу доцільно додати такий набір функціоналу:

- високий рівень автоматизації;
- наявність засобів моніторингу виконання процесів;
- оптимізація використання обчислювальних ресурсів;
- зменшення складності використання;
- спрощення налаштування.

1.3.2 Формування функціональних вимог

На основі проведено аналізу існуючих рішень сформуємо функціональні вимоги до програмного засобу обробки даних з подальших аналізом та діагностуванням генетичних захворювань, який буде реалізовувати автоматизований конвеєр перетворення вхідних файлів у структуровані результати.

Програмний засіб повинен забезпечувати повний цикл обробки – від отримання вхідних файлів до формування кінцевого звіту у зручному форматі.

Отримання вхідних даних:

- завантаження даних із зовнішніх джерел за ідентифікатором;
- підтримка роботи з файлами у форматі FASTQ;
- можливість пропуску етапу завантаження.

Підготовка допоміжних даних:

- автоматичне завантаження необхідних файлів;
- перевірка їх наявності перед запуском;

- підготовка даних до подальшої обробки.

Обробка даних:

- виконання вирівнювання вхідних даних;
- перетворення результатів у проміжні формати;
- сортування та індексація результатів.

Генерація результатів:

- формування файлів результатів у форматі VCF;
- підтримка стиснених форматів даних;
- індексація результатів для швидкого доступу.

Порівняння даних:

- порівняння отриманих результатів із зовнішніми наборами даних;
- виділення спільних записів;
- можливість фільтрації за заданими параметрами.

Формування звітів:

- об’єднання результатів у табличну структуру;
- експорт результатів у структурований формат.

Автоматизація процесу:

- послідовне виконання всіх етапів у межах одного запуску;
- мінімізація ручного втручання.

1.4 Формальна постановка задачі

У даній роботі розглядатиметься задача автоматизованої обробки даних, яка полягатиме у перетворенні вхідних даних у структурований результат із подальшим аналізом.

Де вхідні дані:

- файли у форматі FASTQ;
- ідентифікатор набору даних для завантаження;
- допоміжні файли.

Вихідні дані:

- файл результатів у форматі VCF;
- підсумковий звіт у табличному форматі, що містить структуровані дані.

Суть задачі полягає в реалізації програмного засобу, який буде забезпечувати послідовне виконання етапів обробки даних, включаючи підготовку вхідних даних, їх обробку, формування результатів та створення звіту із мінімальним втручанням користувача, забезпечуючи коректність та контроль виконання етапів обробки.

Критерії оцінювання якості сформованих результатів:

- коректність обробки даних на кожному етапі;
- повнота отриманих результатів;
- ефективність використання обчислювальних ресурсів;
- зручність використання програмного засобу.

Основні завдання на роботу:

- проаналізувати існуючі рішення та визначити переваги і недоліки;
- розроблення структури програмного засобу у вигляді послідовного конвеєра обробки даних;
- реалізація автоматизованого процесу обробки даних, що включає завантаження, обробку та формування результатів;
- інтеграція зовнішніх інструментів у єдиний процес обробки;
- реалізація механізму порівняння отриманих результатів із зовнішніми наборами даних;
- формування структурованого звіту;
- забезпечення зручного запуску програмного забезпечення.

Цілі на розробку:

- створити автоматизований процес обробки та аналізу даних;
- розробити програмне забезпечення зі зменшеною складністю використання;
- забезпечувати відтворюваність результатів обробки при повторному запуску з однаковими параметрами;

- оптимізувати використання обчислювальних ресурсів при обробці великих обсягів даних.

1.5 Висновки до розділу 1

Після проведення аналізу предметної області було визначено, що процес обробки генетичних характеризується великими обсягами інформації, а з кожним роком об’єм даних стає ще більшим. Основними проблемами є необхідність обробки значних масивів даних, а також відсутність цілісних рішень, що об’єднують усі етапи обробки та аналізу в єдиний зручний інструмент. Що ускладнює використання окремих програмних засобів та підвищує складність організації процесу обробки.

Було проаналізовано існуючи програмні засоби для вирішення зазначених проблем, такі як Fastq2vcf SeqMule, які реалізують автоматизовані конвеєри обробки секвенованих даних. Вони дозволяють частково інтегрувати основні етапи обробки та зменшити кількість ручних дій. Було з’ясовано їхні переваги з недоліками і на основі цього було сформульовано вимоги до розроблюваного програмного засобу та поставлено цілі на розробку.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Загальна структура програми

Архітектура розроблюваної програми має базуватися на модульному підході до обробки генетичних даних, що забезпечуватиме розділення функціональності на окремі логічні компоненти. Такий підхід дозволить підвищити масштабованість, спростити підтримку системи та забезпечити можливість повторного використання окремих елементів.

Система має реалізувати пайплайн, який буде здійснювати послідовне перетворення сирих генетичних даних до структурованого результату із подальшим аналізом. Отже логічна структура повинна являти собою набір взаємопов’язаних етапів, кожен з яких виконує окрему функцію перетворення даних.

Етап завантаження даних – відповідає за отримання вхідних даних із зовнішніх джерел. Джерелами можуть виступати локальні файли або віддалені сховища. Також виконується базова перевірка доступності даних та їх відповідності очікуваному формату.

Етап попередньої обробки – відповідає за первинну обробку даних, включаючи перевірку їх якості, очищення від некоректних або пошкоджених записів, а також проведення відповідного формату для подальшої обробки.

Етап обробки – відповідає за перетворення даних до задачі системи. Обробка має включати трансформацію застосовуючи спеціалізованих алгоритмів.

Етап оптимізації – відповідає за структурування отриманих результатів, їх сортування, індексацію та оптимізацію для ефективного доступу. Що дозволить підвищити продуктивність подальших операцій над даними.

Етап формування результатів – на цьому етапі результати обробки приводяться до кінцевого формату, зручного для збереження або подальшого аналізу.

Етап аналізу – відповідає за додаткову обробку результатів з метою узагальненої інформації, виявлення закономірностей або формування висновків на основі отриманих даних.

Етап збереження даних – відповідає за збереження результатів обробки, що забезпечить можливість повторного використання результатів.

Етап керування виконанням – відповідає за виконання, оркестрацію всіх складових системи. Контролює запуск процесів, контроль їх виконання, обробку помилок та забезпечення коректної послідовності обробки даних.

2.2 Функціонування програми

Функціонування програми має забезпечувати повний та послідовний процес обробки генетичних даних, починаючи від отримання вхідних даних та завершуючи формуванням результатів, придатних для подальшого аналізу та інтерпретації. Програма повинна реалізовувати логічно впорядкований конвеєр обробки, у якому кожен етап є залежним від результатів попереднього та забезпечує підготовку даних до наступних перетворень.

На початковому етапі програма повинна забезпечувати прийом та завантаження вхідних даних. Вхідними даними мають бути результати секвенування, представленні у вигляді наборів генетичних даних. Програма має здійснювати перевірку наявності даних, їх доступності, цілісності та відповідності визначеним вимогам до формату. У разі невідповідності даних встановленим критеріям програма повинна забезпечувати формування повідомлень про помилки та припинення або корекцію подальшого виконання.

Після завантаження даних програма повинна виконувати їх попередню обробку. Даний етап має передбачити аналіз якості вхідних даних, виявлення некоректної або пошкодженої інформації, разом з їх усуненням або виправленням. Програма повинна забезпечувати очищення даних від непотрібної інформації, а також приводити до відповідного вигляду,

необхідного для подальшої обробки. У результаті цього етапу мають формуватися підготовлені дані, що відповідають вимогам точності та повноти.

Наступним етапом функціонування програми має бути встановлення відповідності між отриманими даними та порівняльною моделлю. Програма повинна забезпечувати визначення позиції кожного елемента даних відносно базової структури, а також оцінку коректності такого зіставлення. Результатом виконання цього етапу має бути структурований набір даних, що містить інформацію про розташування елементів та їх характеристики.

Після структурування програма повинна виконувати обробку отриманих даних з метою їх впорядкування та оптимізації. Цей етап має включати систематизацію даних за визначеними критеріями, забезпечення швидкого доступу до окремих фрагментів інформації. Результат – це підготовленні дані у вигляді, придатному для подальшого аналізу.

У наступному етапі програма здійснює аналіз підготовлених даних з метою виявлення відмінностей від зразка. Цей процес має передбачати порівняння отриманих даних із базовою структурою та визначення невідповідностей, що можуть свідчити про наявність змін. Програма повинна забезпечувати фіксацію таких відмінностей, їх класифікацію за визначеними ознаками та оцінку достовірності. Результатом цього етапу має бути набір структурованих даних, що містять інформацію про виявленні відхилення.

Програма повинна зберігати отриманні результати та забезпечувати запис даних у відповідному форматі з урахуванням вимог до їх подальшого використання.

2.3 Вибір алгоритму обробки

2.3.1 Алгоритм Сміта-Уотермана

Алгоритм Сміта-Уотермана є класичним методом локального вирівнювання рядків, який використовується для визначення найбільш подібних ділянок між двома послідовностями. Основною особливістю

алгоритму є те, що він не намагається вирівняти послідовності повністю, а знаходить лише ті фрагменти, які мають найбільшу схожість [5].

Принцип роботи відбувається за допомогою методів динамічного програмування. У процесі обчислення формується матриця оцінок, у якій кожна клітинка відображає найкращий можливий результат вирівнювання для відповідних підпослідовностей. На відміну від глобальних алгоритмів, у даному випадку значення не можуть бути меншими за нуль, що дозволяє переривати невигідні вирівнювання та починати нові.

Отже перевагою алгоритму є висока точність, але недолік це висока обчислювальна складність, що обмежує використання алгоритму для обробки великих обсягів даних.

2.3.2 Алгоритм Нідлмана-Вунша

Алгоритм Нідлмана-Вунша є методом глобального вирівнювання рядків, який забезпечує знаходження оптимального вирівнювання між двома послідовностями по всій їх довжині. На відміну від локального підходу, даний алгоритм враховує всі елементи послідовностей [6].

Алгоритм базується на методі динамічного програмування та використовує матрицю оцінок для визначення найкращого результату для відповідних підрядків. Після завершення обчислень виконується зворотне проходження, яке дозволяє відновити оптимальне вирівнювання відповідно до заданої функції оцінки.

Основна перевага алгоритму є отримання глобально оптимального рішення, що робить його гарним вибором для задач порівняння рядків, аналізу текстових даних та пошуку подібностей між рядками. Недолік – висока обчислювальна складність, що призводить до витрат часу та пам’яті при роботі з великими обсягами даних.

2.3.3 Алгоритм Барроуза-Вілера

Алгоритм Барроуза-Вілера застосовується для ефективного пошуку підрядків у великих наборах даних. Основна ідея – попереднє перетворення вхідного рядка з подальшою побудовою індексної структури, що дозволяє значно прискорити операції пошуку [7].

Перетворення Барроуза-Вілера полягає у перегрупування символів рядка таким чином, що однакові символи розташовуються поруч. Це створює сприятливі умови для подальшого стиснення та індексації даних.

Перевагами даного підходу є висока швидкість пошуку та помірне використання пам’яті, що робить його придатним для роботи з великими обсягами даних. Широко застосовується у задачах обробки рядків. Недолік – необхідність попередньої обробки даних, що може вимагати додаткових обчислювальних ресурсів.

2.3.4 Обґрунтування вибору алгоритму

Для визначення найбільш доцільного алгоритму обробки даних було проведено порівняння розглянутих підходів за основними критеріями: тип вирівнювання, точність, швидкість виконання, використання пам’яті, придатність до великих даних. Порівняння наведено у таблиці 2.1.

Таблиця 2.1 – Порівняння алгоритмів обробки

Критерій	Сміт-Уотерман	Нідлман-Вунш	Барроуз-Вілер
Тип вирівнювання	Локальне	Глобальне	Пошук-індексація
Точність	Висока	Висока	Висока
Швидкість	Низька	Низька	Висока

Продовження таблиці 2.1

Критерій	Сміт-Уотерман	Нідлман-Вунш	Барроуз-Вілер
Придатність до великих даних	Обмежена	Обмежена	Висока
Попередня обробка	Ні	Ні	Так

Алгоритми Сміта-Ватермана та Нідлмана-Вунша, попри їх високу точність, потребують значної обчислювальної складності, що ускладнює їх використання у задачах масової обробки даних.

Отже, незважаючи на необхідність попередньої обробки, алгоритм Барроуза-Вілера є найбільш придатним для реалізації системи обробки даних завдяки високій швидкості пошуку та помірному використанні пам’яті.

2.4 Вибір мови програмування

2.4.1 Мова програмування Python

Python високорівнева мова програмування, яка активно використовується для обробки даних та автоматизації процесів. Однією з основних особливостей мови є простий та зрозумілий синтаксис, що дозволяє швидко реалізовувати складні алгоритми [7].

Серед переваг Python варто виділити велику кількість бібліотек для обробки даних, зокрема інструментів для роботи з рядками, файлами та статистичними методами. Крім того добре підходить для інтеграції з зовнішніми програмами та побудови конвеєру роботи з даними.

Недоліки – низька швидкість виконання у порівнянні з компільованими мовами програмування. Це може бути критичним при виконанні обчислювально складних задач.

2.4.2 Мова програмування R

R мова програмування та середовище для статистичного аналізу даних. Основна особливість мови є орієнтація на аналіз та візуалізацію інформації, що робить її зручною для роботи з різними наборами даних і робить її популярною у наукових дослідженнях та аналітиці [8].

Серед переваг R варто виділити наявність великої кількості спеціалізованих пакетів для статистичного аналізу, машинного навчання та обробки даних. Мова має розвинуті засоби побудови засоби побудови графіків і візуалізації результатів, що дозволяє ефективно представляти результати досліджень. Крім того, широке використання R у науковому середовищі забезпечує доступ до значної кількості готових бібліотек та інструментів. Дозволяє швидко виконувати статистичні обчислення без необхідної реалізації складних математичних алгоритмів на низькому рівні.

2.4.3 Мова програмування C++

C++ компільована мова програмування, яка забезпечує високу продуктивність та низьке використання пам’яті. Широко використовується для реалізації обчислювально складних алгоритмів та систем, де важлива швидкість виконання [9].

Основна особливість можливість низькорівневого управління ресурсами, що дозволяє оптимізувати використання пам’яті.

Серед переваг мови слід відзначити високу швидкість виконання, що робить придатною для реалізації алгоритмів обробки рядків та великих обчислювальних задач.

Недоліками є високий ризик помилок при роботі з пам’яттю, що призводить до її витоків. Разом із тим має меншу кількість спеціалізованих бібліотек для аналізу даних, статистичних обчислень та обробки даних.

2.4.4 Обґрунтування вибору мови програмування

Отже, з урахуванням особливостей поставленої задачі, до мови програмування висуваються такі основні вимоги: можливість ефективної роботи з великими обсягами даних, зручна обробка файлів, автоматизація виконання послідовних етапів обробки, а також наявність бібліотек для аналізу даних і взаємодії із зовнішніми програмами.

Важливою складовою є необхідність запуску зовнішніх утиліт через командний рядок, обробки результатів їх виконання та організації послідовності виконання операцій, що вимагає від мови програмування наявності засобів для роботи з системними викликами та процесами.

Мова програмування Python характеризується простотою синтаксису та має велику кількість бібліотек для роботи з даними та зручні засоби інтеграції з іншими інструментами, що робить її універсальним рішенням для широкого кола задач

Мова програмування R орієнтована на статистичний аналіз і візуалізацію даних. Вона має значну кількість спеціалізованих пакетів, однак менш придатна для розробки повноцінних програмних систем та реалізації алгоритмів загального призначення.

Мова C++ забезпечує високу швидкість виконання, водночас складніша у використанні та має меншу кількість бібліотек для роботи з даними, що особливо важливо для цієї роботи.

Таблиця 2.2 – Порівняння мов програмування

Критерій	Python	R	C++
Швидкість	Низька	Низька	Висока
Бібліотеки	Висока	Середня	Низька

Продовження таблиці 2.2

Критерій	Python	R	C++
Призначення	Універсальна	Аналіз	Системне програмування
Робота з пам’яттю	Автоматична	Автоматична	Ручна
Робота з великими даними	Висока	Середня	Висока

Отже, з урахуванням особливостей поставленої задачі, до мови програмування висуваються такі основні вимоги: можливість роботи з великими обсягами даних, зручна обробка файлів, наявність бібліотек для роботи з даними та їх аналізом, а також можливість інтеграції з іншими програмними компонентами.

Проведений порівняльний аналіз показав, що мова програмування Python найбільш повно відповідає зазначеним вимогам. Вона забезпечує зручні засоби роботи з файлами, підтримує обробку великих обсягів даних та має широкий набір бібліотеки, що дозволить реалізувати необхідний функціонал.

Незважаючи на нижчу швидкість виконання у порівнянні з компільованими мовами програмування, переваги Python роблять її найбільш доцільним для реалізації поставленої задачі.

2.5 Вибір середовища розробки

2.5.1 Visual Studio Code

Visual Studio Code – це легке та розширюване середовище розробки, яке широко використовується для написання програмного коду різними мовами програмування, зокрема Python. Основною особливістю є модульна

архітектура, яка дозволяє розширювати функціональність за допомогою великої кількості плагінів.

Серед переваг Visual Studio Code варто виділити високу швидкість роботи, зручний та зрозумілий інтерфейс, інтеграцію з системами контролю версій, а також підтримку великої кількості розширень для роботи з даними, Docker, базами даних та інструментами розробки пайплайнів. Крім того, середовище підтримує інтеграцію з терміналом, що спрощує роботу з командним рядком та автоматизацію процесів.

2.5.2 PyCharm

PyCharm – це спеціалізоване середовище розробки для мови програмування Python, яке забезпечує широкий набір інструментів для розробки, налагодження та тестування програм.

До переваг PyCharm належать глибока інтеграція з Python, потужні засоби автодоповнення коду, підтримка роботи з віртуальними середовищами, а також інструмент для роботи з базами даних і веб-розробки. Середовище також добре підходить для роботи з великими проєктами та складними структурами коду.

2.5.3 Cursor

Cursor – це сучасне інтегроване середовище розробки, побудоване на основі Visual Studio Code, яке розширює його можливість за рахунок інтеграції штучного інтелекту. Основною особливістю є підтримка AI-асистента для написання, аналізу та рефакторингу коду.

Серед переваг Cursor варто виділити можливість автоматичної генерації коду, пояснення складних фрагментів програм, а також допомогу у пошуку помилок та оптимізації алгоритмів. Це дозволяє значно прискорити процес розробки, особливо при роботі зі складними задачами обробки даних. Крім

того, Cursor підтримує більшість розширень Visual Studio Code, що забезпечує гнучкість та зручність у налаштуванні середовища.

Недоліками є залежність від інтернет-з’єднання для роботи AI-функцій, що є єдиною особливістю від Visual Studio Code.

2.5.4 Обґрунтування вибору середовища розробки

Для визначення найбільш доцільного середовища розробки проведемо порівняльний аналіз розглянутих варіантів за основними критеріями, що безпосередньо впливають із особливостей поставленої задачі. Основна увага приділяється підтримці мови Python, можливості розширення функціональності за рахунок плагінів, роботі з командним рядком, засобам налагодження програм та продуктивності середовища, порівняння наведено у таблиці 2.3.

Visual Studio Code забезпечує високу гнучкість за рахунок системи розширень, що дозволяє підтримку Python, покращене автодоповнення, а також інструменти для роботи з файлами та терміналом. Крім того, середовище характеризується високою швидкістю роботи та невеликими вимогами до ресурсів.

PyCharm надає потужні вбудовані засоби для розробки на Python, включаючи автодоповнення коду, налагодження та управління проєктами. Разом із тип, дане середовище є більш ресурсозатратним і меншим гнучким у налаштуванні, що може ускладнювати роботу у випадках, коли необхідно активно використовувати зовнішні інструменти.

Cursor є сучасним середовищем розробки з інтегрованою підтримкою штучного інтелекту, що дозволяє автоматизувати написання коду та отримувати пояснення до нього.

Таблиця 2.3 – Порівняння середовищ розробки

Критерій	Visual Studio Code	PyCharm	Cursor
Підтримка Python	Присутня	Присутня	Присутня
Кількість плагінів	Багато	Обмежено	Багато
Термінал	Вбудований	Вбудований	Вбудований
AI-підтримка	Присутня	Відсутня	Присутня

Проведений порівняльний аналіз показав, що середовище Visual Studio Code найбільш повно відповідає зазначеним вимогам. Воно забезпечує необхідну гнучкість, дозволяє зручно працювати з зовнішніми інструментами через термінал та має достатній набір функціональних можливостей для реалізації поставленої задачі.

Незважаючи на наявність альтернативних рішень,Visual Studio Code є найбільш збалансованим вибором з точки зори функціональності, продуктивності та зручності використання, що обумовлює його доцільність використання у даній роботі.

2.6 Вимоги до технічних та програмних засобів

2.6.1 Вимоги до технічних засобів

Оскільки розроблювальна система передбачає обробку великим обсягів даних, зокрема роботу з файлами значного розміру та виконання обчислювального складних операцій, до технічних засобів висуваються вимоги щодо достатньої обчислювальної потужності, обсягу оперативної пам’яті та швидкості доступу до даних.

Для забезпечення виконання роботи програмної системи необхідні відповідні апаратні ресурси:

- процесор з частотою не менше 3.0 ГГц;

- оперативна пам’ять не менше 8 Гб;
- вільне місце на диску не менше 20 Гб;
- багатоядерний процесор не менше 4 ядер.

2.6.2 Вимоги до програмних засобів

З огляду на те, що система реалізує обробку даних у вигляді послідовного конвеєру та передбачає інтеграцію різних програмних компонентів, до програмних засобів висуваються вимоги щодо підтримки автоматизації процесів, роботи з великими файлами та взаємодії із зовнішніми інструментами.

Для функціонування системи необхідне відповідне програмне забезпечення.

- операційна система Linux;
- мова програмування Python 3.12+;
- середовище розробки Visual Studio Code;
- встановлені спеціалізовані інструменти для обробки даних.

2.7 Висновки до розділу 2

У другому розділі було розглянуто основні теоретичні та практичні складові, необхідні для розробки програмного рішення обробки генетичних даних великого обсягу з подальшим формуванням результатів у структурованому форматі.

Проведено огляд алгоритмів обробки рядків і вирівнювання, таких як алгоритми Сміта-Уотермана та Нідлмана-Вунша, а також розглянуто підхід на основі перетворення Барроуза-Вілера. Визначено їхні особливості, переваги та обмеження, що дозволило сформувати уявлення про існуючі підходи до обробки та аналізу послідовностей даних і оцінити їх ефективність при роботі з великими обсягами інформації.

Для того щоб обрати мову програмування було проаналізовано Python, R та C++ за такими критеріями, як швидкість виконання, наявність бібліотек, спеціалізація, рівень контролю над ресурсами, обробка великих даних і робота з файлами. На основі проведеного порівняльного аналізу було обґрунтовано вибір мови програмування Python як оптимального інструменту, що забезпечує баланс між зручністю розробки, гнучкістю та спеціалізацією при роботі з великими наборами даних.

Було визначено вимоги до технічних та програмних засобів. Описано апаратні характеристики, необхідні для стабільної роботи конвеєру з обробки великих файлів, а також програмне забезпечення, включаючи операційну систему, середовище розробки, версію мови програмування та необхідні інструменти для роботи з генетичними даними та файлами великого обсягу.

Отже, у результаті було сформовано теоретичну та технологічну основу для реалізації програмного рішення, що забезпечує виконання конвеєру обробки генетичних даних, їх аналіз та подальше представлення у структурованому вигляді.

3 РОЗРОБКА ПРОГРАМИ

3.1 Загальна архітектура системи

Архітектура розробленої системи побудована за принципом послідовної обробки даних, що реалізує конвеєрну модель виконання. Такий підхід передбачає проходження вхідних даних через ряд взаємопов’язаних етапів, кожен з яких виконує окрему функцію роботи з даними.

Основною ідеєю архітектури є поетапне перетворення даних, від початкового неструктурованого або напівструктурованого вигляду до кінцевого структурованого результату. При цьому кожен наступний етап використовує результати попереднього, що забезпечує логічну цілісність процесу та спрощує контроль виконання пайплайну.

Запропонована архітектура орієнтована на роботу з великими обсягами даних, тому ключовим аспектом є використання обчислювальних ресурсів. З цією метою обробка організована таким чином, щоб уникати повного завантаження даних у пам’ять і забезпечити їх поетапне опрацювання. Це дозволяє підвищити продуктивність системи та забезпечити її стабільну роботу навіть при значних обсягах вхідної інформацію.

Ще однією важливою особливістю архітектури є її простота та розширюваність. Конвеєрна модель дозволяє легко модифікувати або доповнювати окремі етапи без необхідності суттєвих змін у загальній структурі системи. Це забезпечує гнучкість у подальшому розвитку програмного забезпечення.

Система складається з таких етапів роботи з даними:

- етап завантаження даних – забезпечує отримання необхідних файлів із зовнішніх джерел. На цьому етапі здійснюється автоматизоване завантаження даних;
- етап зчитування даних – відповідає за відкриття та поетапне читання вхідних файлів. Забезпечує коректну роботу з файлами великого обсягу без необхідності їх повного завантаження у пам’ять;

- етап попередньої обробки – включає очищення, перевірку та підготовку даних до подальшого аналізу. На цьому етапі дані приводяться до уніфікованого формату, що спрощує їх обробку;
- етап аналізу даних – реалізує основну логіку роботи системи. Передбачає виконання операцій над даними відповідно до поставленої задачі та отримання проміжних або кінцевих результатів;
- етап формування результатів – відповідає за структурування отриманих даних та їх збереження у вигляді вихідних файлів або звітів, придатних для подальшого використання.

Взаємодія між етапами здійснюється послідовно, де вихідні дані кожного етапу є вхідними для наступного, наведено на рисунку 3.1. Така організація забезпечує зрозумілу логіку функціонування системи, спрощує її тестування та підвищує надійність.

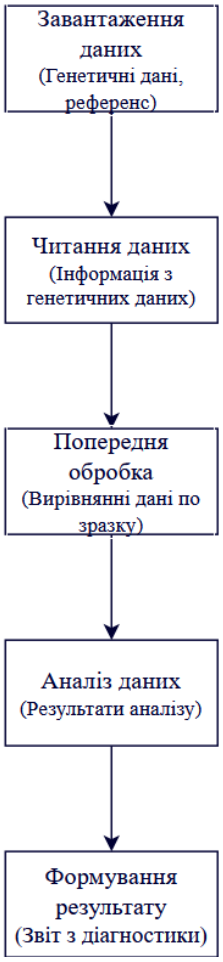

Рисунок 3.1 – Взаємодія між етапами

3.2 Реалізація основних етапів системи

3.2.1 Етап завантаження даних

Етап завантаження даних є початковим у роботі системи та відповідає за отримання вхідної інформації із зовнішніх джерел. На цьому етапі формуються всі необхідні дані, які будуть використанні у подальшій обробці.

Вхідними даними для цього етапу є ідентифікатори наборів даних з яких необхідно отримати інформацію. На основі цих даних система ініціює процес завантаження відповідних файлів із віддалених серверів або сховищ.

Окрім основних вхідних файлів, на цьому етапі також може здійснюватися завантаження допоміжних ресурсів, необхідних для коректної роботи системи. До таких ресурсів належать еталонні послідовності, індексні файли та інші службові дані, які використовуються під час подальшої обробки.

У процесі виконання здійснюється передача запитів до зовнішніх джерел, отримання файлів та збереження у локальній файловій системи. Перед початком завантаження виконується перевірка наявності необхідних файлів у сховищі. У разі, якщо файли вже існують, повторне завантаження не здійснюється. Такий підхід дозволить уникнути дублювання даних, зменшити час виконання програми.

Окрім безпосереднього завантаження, на цьому етапі може виконуватися додаткова обробка отриманих файлів, зокрема їх розпакування. Це дозволяє забезпечити готовність даних до наступного етапу без необхідності додаткових перетворень.

Важливою характеристикою цього етапу є його орієнтація на роботу з великими обсягами даних. Завантаження виконується пофайлово, що дозволяє краще керувати використовувати ресурси системи та уникати перезавантаження пам’яті.

Результатом виконання етапу є набір локально доступних файлів, що зберігаються у файловій системи та передаються на етап зчитування даних. Ці файли є вихідною основою для подальших перетворень у системи.

Для детального представлення процесу завантаження даних та перевірки наявності необхідних ресурсів наведено рисунок 3.2, що відображає послідовність виконання операцій на даному етапі.

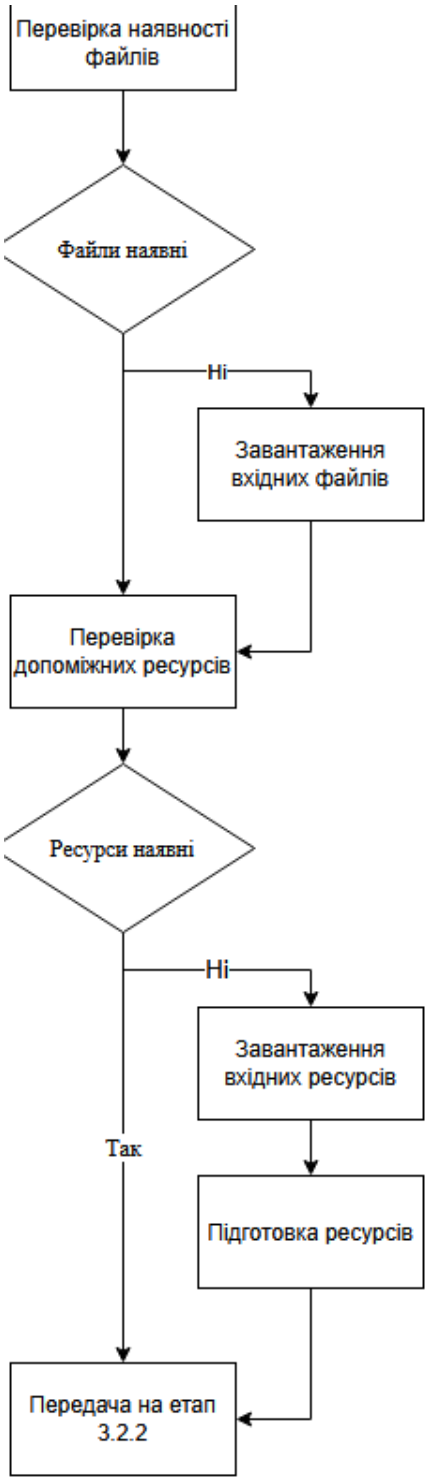

Рисунок 3.2 – Етап завантаження даних

3.2.2 Етап структурування даних

Етап зчитування даних забезпечує отримання інформації з вхідних файлів та її перетворення у внутрішнє представлення, придатне для подальшої обробки в системі. На цьому етапі здійснюється підготовка даних до виконання наступних операцій без заміни їх змісту.

Вхідними даними є файли отримані на попередньому етапі завантаження. Ці файли можуть мати значний обсяг, тому процес зчитування організовано з урахуванням обмежень оперативної пам’яті.

Зчитування виконується послідовно, що дозволяє обробляти великі масиви даних без необхідності повного завантаження файлу у пам’ять. У процесі зчитування дані інтерпретуються відповідно до їх структури та формату, після чого перетворюються у внутрішні об’єкти або записи, з якими працює система.

Представлення процесу зчитування даних, включаючи їх поетапну обробку наведено на рисунку 3.3, що показує внутрішню логіку виконання даного процесу.

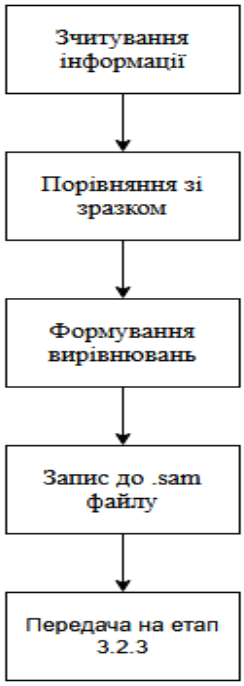

Рисунок 3.3 – Етап структурування даних

Основним завданням цього етапу, є перетворення неструктурованих фрагментів даних у впорядковану модель шляхом їх прив’язки до зразку. Для забезпечення високої швидкості та точності обробки в системі реалізовано алгоритм вирівнювання Барроуза-Вілера. Застосування цього алгоритму дозволяє ефективно працювати з великими масивами даних завдяки використанню спеціальних індексів.

3.2.2.1 Функціональна архітектура етапу структурування даних

Етап реалізує логіку перетворення вхідних об’єктів у структуровану координатну модель. Процес включає:

- обчислювальне вирівнювання (функція align_reads) – порівняння кожного фрагменту FASTQ з референсом, визначаючи його точне місцезнаходження. Результат записується у форматі SAM, який містить інформацію про кожне співпадіння;
- індексація (функція prepare_reference) – створює цифрову структуру для миттєвого пошуку координат під час зіставлення;
- зіставлення та об’єднання даних (функція align_reads) – зчитується інформація з пари вхідних файлів, порівнює їхні фрагменти зі зразком та зводить до спільного результату у форматі SAM;
- впорядкування та конвертація – трансформація SAM-файлу до компактної бінарної форми.

3.2.3 Етап попередньої обробки даних

Цей етап призначений для трансформації вхідних бінарних структур у фінальний текстовий звіт. Основним завданням є виявлення розбіжностей у даних та їхня класифікація згідно із зовнішніми базами даних. Вхідними даними є структуровані записи формату BAM, отримані на попередньому етапі.

Програмна реалізація базується на принципах потокової обробки, що забезпечує стабільну роботу системи при опрацюванні великих масивів інформації без перевантаження оперативної пам’яті. Процес обробки включає послідовне застосування алгоритмів фільтрації, порівняння та нормалізації даних, зображено на рисунку 3.4.

Рисунок 3.4 – Етап попередньої обробки даних

Фільтрація виконується за програмно заданими критеріями, що дозволяє виключити нерелевантні записи та зосередити обчислювальні ресурси на цільових сегментах даних. Усі операції приводять інформацію до

уніфікованого вигляду, що забезпечує сумісність результатів із зовнішніми аналітичними системами.

3.2.3.1 Функціональна архітектура етапу обробки даних

Цей етап об’єднує ключові обчислювальні модулі проєкту:

- пошук варіацій (функція call_variants) – використовує конвеєр bcftools для ідентифікації точок відмінності між геномом пацієнта та зразком;
- синтаксичний аналіз (функція intersect_with_clinvar) – здійснюється порівняння знайдених мутацій із базою ClinVar. Результатом чого стає звіт у форматі VCF, який містить структурований та відфільтрований перелік генетичних варіацій, готовий до фінального аналізу.

3.2.4 Етап аналізі та інтерпретації результатів

Цей етап є завершальною стадією обробки інформації, де здійснюється трансформація виявлених відхилень у клінічно значущий діагностичний звіт. Основною метою етапу є верифікація знайдених записів шляхом їхнього автоматизованого зіставлення з глобальними реєстрами патологій, зображено на рисунку 3.5.

Процес реалізовано за допомогою методів реляційного аналізу даних. На відміну від попередніх кроків, де відбувалася фізична обробка файлів, на цьому рівні система оперує високорівненими структурами даних, що дозволяє виконувати точне порівняння за сукупністю параметрів.

Ключовим аспектом етапу є сематичне анотування, тобто кожне виявлене відхилення збагачується інформацією про пов’язані з ним захворювання, назви генів та сукупність клінічної значущості. Це перетворює набір координат у структуровану базу знань, готову для прийняття медичних рішень.

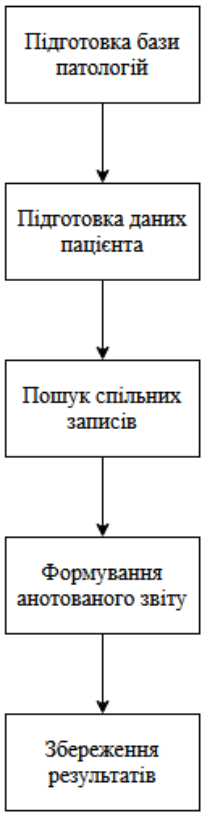

Рисунок 3.5 – Етап аналізу

3.2.4.1 Функціональна архітектура етапу аналізу

Логіка діагностування генетичних відхилень реалізоване через наступні програмні компоненти:

- завантаження зразку (функція load_clinvar) – здійснює десеріалізацію стиснутих даних бази ClinVar. Витягуються певні атрибути для сформування опорної структури для порівняння;
- підготовка вибірки (функція load_sample) – готує дані отримані на попередніх етапах, до проведення аналізу, фільтруючи записи без альтернативних алелей;
- діагностика генетичних захворювань (функція compare) – виконує операцію внутрішнього об’єднання двох масивів даних. Це дозволяє миттєво відсіяти безпечні варіації та залишити лише ті записи, що мають зафіксоване клінічне підтвердження.

Кінцевим результатом стає файл, фінальний діагностичний звіт, що містить повний перелік підтверджених генетичних захворювань із зазначенням конкретних патологій та генів, що до них призвели.

3.3 Висновки до розділу 3

У третьому розділі було виконано програмну реалізацію обчислювального конвеєра для діагностування генетичних захворювань. Розроблена система забезпечує повний цикл опрацювання інформації, який було розділено на такі ключові етапи:

- етап завантаження даних та підготовки вхідних даних – на цьому початковому етапі здійснюється отримання необхідних для аналізу ресурсів, а саме генетичних даних, які будуть аналізуватися та допоміжних ресурсів, що застосовуються для аналізу;
- етап структурування даних – виконується обробка необроблених даних шляхом їх зіставлення зі зразком за допомогою алгоритму вирівнювання Барроуза-Вілера.;
- етап обробки даних – здійснюється аналіз сформованої структури для пошуку точок розбіжності зі зразком. Завдяки використати потокової обробки та інструментів фільтрації, система ідентифікує мутації та формує їх структурованих перелік;
- етап діагностичного аналізу та інтерпретації – завершальний етап, на якому виявлені відхилення автоматично порівнюються з базою ClinVar. За допомогою методів реляційного аналізу даних система визначає клінічну значущість мутацій та формує фінальний звіт із переліком підтверджених патологій.

Реалізація цих етапів дозволила створити масштабне програмне рішення, яке автоматизує шлях від отримання сирих даних до формування зрозумілого діагностичного висновку.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Опис застосування програми

Цей підрозділ визначає функціональне призначення розробленого програмного продукту, умови його експлуатації, технічні обмеження та методи, що були використані для розв’язання поставлених завдань.

4.1.1 Призначення програми

Програма призначена для автоматизації процесів аналізу великих масивів генетичних даних та виявлення розбіжностей у цифрових послідовностях відносно еталона. Основним функціональним призначенням є обчислювальна ідентифікація патологій через порівняння результатів аналізу з базами знань у напівавтоматичному режимі. Програмний засіб дозволяє мінімізувати участь людини в рутинних операціях порівняння даних, забезпечуючи високу точність результатів.

Через стрімке зростання обсягів генетичних даних, що генеруються сучасними системами секвенування, традиційні методи ручної роботи стають неможливими. Необхідність використання розробленого програмного продукту обґрунтована такими факторами:

- потреба в інтеграції розрізнених інструментів: Процес діагностування зазвичай вимагає використання багатьох утиліт. Розроблена програма об’єднує їх у єдиний конвеєр, що дозволяє уникнути помилок при передачі даних між етапами та забезпечити цілісність аналізу;
- висока вартість помилки: У медичній діагностиці хибні позитивні або хибні негативні результати можуть призвести до неправильного лікування. Автоматизація етапу аналізу дозволяє усунути людський фактор при ідентифікації мутацій.

4.1.2 Область застосування

Враховуючи викладені фактори необхідності впровадження, зокрема потребу в автоматизації обробки великих масивів даних та мінімізації ризиків, пов’язаних із людським фактором, розроблений програмний продукт має широку область практичного застосування. Завдяки універсальності реалізованих алгоритмів вирівнювання та аналізу, система може бути інтегрована в різні сегменти, такими як:

- клінічна діагностика: використання програми як інтегрованого модуля в системах підтримки прийняття медичних рішень для автоматизації рутинних процесів порівняння великих послідовностей даних;
- фармакологічні дослідження: застосування алгоритмів для аналізу даних впливу специфічних препаратів на цифрові моделі біологічних структур та виявлення закономірностей у відхиленнях;
- персоналізована медицина: створення індивідуальних цифрових профілів на основі аналізу відхилень від еталона для прогнозування потенційних ризиків та розробки превентивних заходів.

4.1.3 Застосовувані методи вирішуваних завдань

Процес використання програмного продукту побудований за принципом послідовного обчислювального конвеєра. Роль користувача зводиться до підготовки вхідних параметрів, тоді як основні етапи трансформації даних відбуваються в автоматизованому форматі.

Сценарій функціонування програми включає наступні технологічні етапи:

Автоматизоване завантаження та підготовка ресурсів. Завантаження з підготовкою вхідних даних та допоміжних ресурсів до подальшої обробки;

Алгоритмічне структурування даних. На даному етапі відбувається трансформація необроблених даних з метою подальшого аналізу.

Аналіз відмінностей. Здійснюється автоматизований аналіз сформованої структури для пошуку розбіжностей із зразком.

Діагностування генетичних захворювань. На основі виявлених розбіжностей між досліджуваною та еталонною послідовностями здійснюється інтерпретація результатів аналізу, що дозволяє ідентифікувати потенційні патології або генетичні особливості.

Така модель використання дозволяє мінімізувати участь людини в обчислювальному процесі. Користувач лише ініціює запуск системи, після чого програма самостійно виконує повний цикл трансформації даних – від завантаження сирих даних до видачі готового результату.

4.2 Інструкція по експлуатації програми

4.2.1 Керівництво системного програміста

Даний підрозділ містить відомості необхідні для встановлення, налаштування та забезпечення працездатності програмного комплексу в умовах конкретного обчислювального середовища.

Для коректного функціонування обчислювального конвеєра необхідно забезпечити наявність наступного ПЗ:

- ОС: Linux (рекомендовано Ubuntu 24.04.2 LTS або новіше) або середовище Windows Subsystem for Linux (WSL);
- інтерпретатор мови: Python версії 3.12 або вище;
- системні утиліти: BWA для вирівнювання прочитань відносно еталонної послідовності; SAMtools для обробки, сортування та індексації генетичних файлів.

Програма має модульну структуру, що включає:

- модуль управління зовнішніми викликами;
- модуль потокової фільтрації та пошуку відхилень;
- блок аналізу для зіставлення даних із базою знань.

Взаємодія між модулями здійснюється через стандартні інтерфейси передачі аргументів та файлову систему для проміжних результатів.

Для перевірки програми після встановлення системного ПЗ необхідно виконати контрольне тестування:

- метод: запуск програми з ідентифікатором тестового набору даних;
- результат: успішним вважається завершення всіх етапів без помилок та поява вихідного файлу звіту report.csv.

Програма здійснює автоматичне протоколювання всіх етапів роботи. У разі виникнення непередбачених збоїв, системний програміст може звернутися до файлу логів, для аналізу та системних повідомлень.

4.2.2 Керівництво програміста

Даний підрозділ містить технічні відомості про внутрішню побудову програми, її характеристики та протоколи обміну даними, що необхідно для технічного супроводу та можливої модифікації програмного продукту.

Звернення до програми здійснюється через CLI. Точкою входу є головний модуль main.py. Формат виклику передбачає передачу ключових аргументів, що визначають ідентифікатор зразка для аналізу.

Програма ініціалізує системні виклики та передає керування послідовно кожному етапу.

Вхідні дані:

- цифрові послідовності у форматі FASTQ;
- еталонна послідовність у форматі FASTA;
- база клінічних знань ClinVar у форматі VCF.

Вихідні дані:

- структурований звіт у форматі CSV, що містить ідентифіковані відхилення, назви пов’язаних патологій та оцінку їхньої клінічної значущості;
- файл протоколювання, що містить детальні відомості про хід виконання кожного етапу.

У процесі експлуатації програма генерує наступні типи повідомлень:

- інформаційні, сповіщення про початок кожного етапу;
- діагностичні, вивід результатів роботи системних утиліт для моніторингу прогресу обчислень;
- повідомлення про помилки, виникають у разі відсутності вхідних файлів, некоректного формату даних або збою системних викликів.

4.3 Експериментальні дослідження

Метою експериментальних досліджень є оцінка працездатності розробленого програмного забезпечення для обчислювального діагностування генетичних захворювань, перевірка коректної реалізації етапів конвеєра та валідація результатів аналізу мутацій.

4.3.1 Загальна схема експерименту

Експериментальне дослідження розробленого програмного засобу проводилося відповідно до послідовної схеми, що відображає основні етапи його функціонування.

На першому етапі здійснювалося формування тестового середовища, яке включало встановлення та налаштування програмного забезпечення, необхідного для виконання обчислювального конвеєра.

Другий етап передбачав запуск програмного засобу з відповідними параметрами, що забезпечувало ініціалізацію процесу обробки генетичних даних у автоматичному режимі.

На завершальному етапі проводився аналіз отриманих результатів, який включав перевірку коректності виконання кожного етапу конвеєра, оцінку сформованих вихідних даних та їх відповідність очікуваним результатам.

4.3.2 Використане устаткування та програмне забезпечення

Для проведення серії експериментів було використано наступне апаратне та системне забезпечення:

- центральний процесор (CPU): AMD Ryzen 3 7330U, 4 фізичних ядра, обробка у 8 обчислювальних потоків;
- графічний процесор (GPU): AMD ATI 05:00.0;
- операційна система (OS): Ubuntu 24.04.2 LTS;
- середовище розробки (IDE): Visual Studio Code.

4.3.3 Вхідні дані для проведення експериментів

Вхідними даними для проведення експериментальних досліджень слугували відкриті генетичні набори даних, отримані з міжнародної бази даних NCBI. Дані представлені у форматі FASTQ, який використовується для зберігання результатів секвенування та містить як самі нуклеотидні послідовності, так і інформацію про якість їх зчитування.

На рисунку 4.1 наведено фрагментований приклад структури FASTQ-файлу, який включає службову інформацію про ідентифікатор прочитань, нуклеотидну послідовність та показники якості.

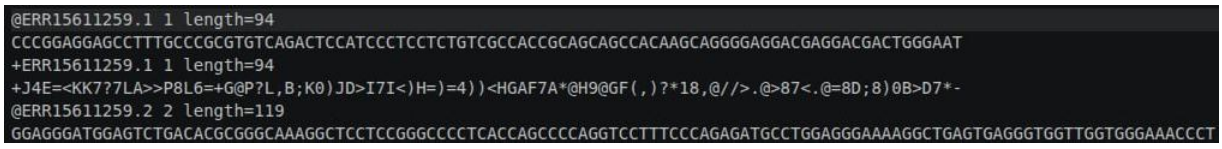

Рисунок 4.1 – Структура FASTQ файлу

4.3.4 Параметри запуску методів і програмних засобів

Запуск обчислювальних методів та програмних засобів здійснюється через консольний інтерфейс операційної системи шляхом виконання

відповідної команди. Такий підхід забезпечує гнучкість налаштування процесу обробки даних та можливість автоматизації виконання програмного конвеєра.

Приклад запуску програми наведено на рисунку 4.2.

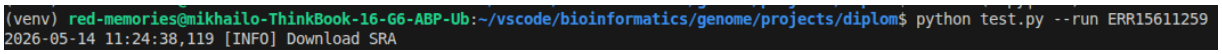

Рисунок 4.2 – Запуск обчислювальних методів

У процесі запуску користувач має можливість задавати параметри, що впливають на поведінку програми та окремі етапи обробки даних.

Опис основних параметрів запуску:

– --run ERR15611259 – параметр, що визначає унікальний ідентифікатор набору вхідних даних. На основі цього ідентифікатора здійснюється отримання або обробка відповідного генетичного масиву;

– --skip_download – параметр, який дозволяє пропустити етап завантаження вхідних даних. Використовується у випадку, коли необхідні файли вже присутні у локальному середовищі, що дозволяє зменшити час виконання програми.

Таким чином, використання параметрів запуску забезпечує адаптивність програмного засобу до різних сценаріїв використання.

4.3.5 Результати виконання програмного засобу

У результаті виконання розробленого програмного засобу формується сукупність вихідних даних, що відображають як технічні результати обробки, так і підсумки аналітичного дослідження.

Ключовим результатом роботи програми є звіт діагностування, який містить узагальнену інформацію про виявленні розбіжності між досліджуваними генетичними послідовностями та еталонним зразком, наведено на рисунку 4.3. У звіті подається інтерпретація отриманих результатів, що дозволяє зробити висновки щодо наявності можливих

генетичних відхилень або патологій. Даний звіт є основним інформаційним продуктом системи та використовується для подальшого аналізу і прийняття рішень.

Додатковим результатом є файл журналу виконання, який містить детальну інформацію про всі етапи роботи програмного конвеєра, наведено на рисунку 4.4. У ньому фіксуються виконанні операції, параметри запуску, часові характеристики, а також можливі помилки або попередження. Лог-файл забезпечує можливість контролю коректності виконання програми, спрощує процес налагодження та підвищує відтворюваність результатів.

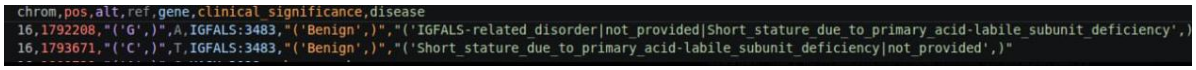

Рисунок 4.3 – Фрагментований звіт діагностування

Рисунок 4.4 – Зміст файлу журнала виконання

За отриманими результатами експериментальних досліджень та їх аналізі, можна підтвердити успішність практичної реалізації розроблених методів та моделей. Верифікація працездатності створеного програмного забезпечення на реальних наборах даних із відкритих міжнародних репозиторіїв засвідчила його високу обчислювальну стабільність і точність. Сформовані програмою дані та службові журнали безпосередньо вказують на досягнення мети, що була поставлена у вступі.

Розроблена конвеєрна архітектура дозволила досягти мети за рахунок усунення часових витрат на ручні операції. Замість послідовного ручного

запуску оператором окремих консольних утиліт, контролю створення проміжних файлів та їх конвертації, програмний засіб виконує весь цикл обробки автономно за одну команду виклику.

Такий підхід не лише мінімізував ймовірність виникнення механічних помилок оператора на етапах зіставлення, а й повністю усунув очікування та простої обчислювальної системи між ізольованими етапами розрахунків.

ВИСНОВКИ

З метою підвищення швидкості діагностування генетичних захворювань, у роботі вирішено актуальну задачу створення автоматизованого програмного комплексу для обчислювального діагностування. Отримані результати підтверджують успішність вирішення поставленої проблеми, а розроблений інструмент є швидкодійним, надійним та автономним рішенням для автоматизації складних конвеєрів з обробки генетичних даних.

Спершу було проведено аналіз сучасних методів, алгоритмів та інструментальних засобів аналізу генетичних даних. Встановлено, що існуючі рішення, не повною мірою задовольняють вимоги оперативності та доступності на локальних робочих станціях через високу складність налаштування середовища, надмірні вимоги до серверних ресурсів та відсутність наскрізної автоматизації, що змішує оператора вручну координувати проміжні файли. Це чітко обґрунтувало мету роботи та підтвердило необхідність розробки нового програмного забезпечення з повної автоматизацією.

Після чого досліджено та теоретично обґрунтовано вибір методів та алгоритмів для усунення виявлених недоліків існуючих систем. На основі проведеного аналізу сформовано стійкий алгоритм обчислювального конвеєра, що включатиме алгоритм Бароуза-Вілера для швидкісного структурування і вирівнювання послідовностей та методи реляційного аналізу для семантичного зіставлення даних. Такий підхід дозволив закласти теоретичний фундамент для мінімізації обчислювальної складності майбутнього програмного застосунку.

Спираючись на аналіз існуючих рішень та обраний функціонал, було спроєктовано та програмно реалізовано архітектуру обчислювального конвеєра мовою Python. Програма забезпечує стабільне послідовне виконання таких етапів конвеєру: завантаження даних, зчитування даних, обробка даних та діагностування генетичних захворювань.

Розроблений програмний застосунок було протестовано в операційній системі Ubuntu на локальному комп’ютері. Результатами тестування було підтверджено стабільність роботи застосунку, а також успішне автоматичне формування вихідного звіту та файлу журналу виконання.

За результатами проведено тестування та аналізі отриманих даних було доведено, що поставлену мету роботи – збільшення швидкості діагностування за допомогою автоматизації – було виконано. Це можна підтвердити за тим, що оператор запускає процес виконання діагностування за рахунок лише однієї команди. Це повністю усунуло застої виконання обчислень між етапами та знизило ризик помилок людини під час ручного керування файлами.

Перспективою подальших досліджень у цій галузі полягають у технічному вдосконаленні програмного комплексу. Основними напрямками розвитку програми є інтеграція зручного графічного інтерфейсу, розширення переліку підтримуваних баз знань, а також подальша оптимізація обчислювальних процесів разом з інтеграцією ширшого спектру спеціалізованих інструментів та алгоритмів. Подальше впровадження та масштабування цього застосунку в медичну практику дозволить суттєво покращити якість проведення профілактичних оглядів населення, а також значно підвищити шанси раннього діагностування генетичних захворювань, що важливо для збереження життя.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Wetterstrand K. A DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP). National Human Genome Research Institute. 2023. URL: https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data (date of access: 30.04.2026).

2. Pandey D. Genetic Testing Market Size, Share and Trends 2025 to 2035. Precedence Research 2026. URL: https://www.precedenceresearch.com/genetic-testing-market (date of access: 30.04.2026).

3. Gao X., Xu J., Starmer J. Fastq2vcf: a concise and transparent pipeline for whole-exome sequencing data analyses. BMC Research Notes. 2015. Vol. 8. P.72. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC4376134/ (date of access: 30.04.2026).

4. Guo Y., Ding X., Shen Y., et al. SeqMule: automated pipeline for analysis of human exome genome sequencing data. Scientific Reports. 2015. Vol. 5. P. 14283. URL: https://pubmed.ncbi.nlm.nih.gov/26381817/ (date of access: 30.04.2026).

5. Кисляк С. В., Настенко Є. А., Основи молекулярної біології та біоінформатики: комп’ютерний практикум. Київ : КПІ, 2018. 95 с.

6. Lee C. T., Peng S. L. A Pairwise Alignment Algorithm for long sequences of High Similarity. Information and Communication Technology. 2017. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC7123042/ (date of access: 30.04.2026).

7. Li H., Durbin R. Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics. 2009. URL: https://pmc.ncbi.nlhm.nih.gov/articles/PMC2705234/ (date of access: 01.05.2026).

8. Python Documentation. Python Software Foundation. 2026. URL: https://docs.python.org/3/ (date of access: 10.05.2026).

9. R Documentation and Manuals. The R foundation for Statistical Computing. 2026. URL: https://cran.r-project.org/manuals.html (date of access: 10.05.2026).

10. C++ Reference Documentation. Standard C++ Foundation. 2026 URL: https://devdocs.io/cpp/ (date of access: 10.05.2026).

ДОДАТОК А

Текст програми

А.1 Текст файлу main.py

```
import argparse
import subprocess
import logging
from pathlib import Path
from diagnosis import run_diagnosis

REFERENCE_URL = "https://ftp.ensembl.org/pub/release-110/fasta/homo_sapiens/dna/Homo_sapiens.GRCh38.dna.primary_assembly.fa.gz"
CLINVAR_URL = "https://ftp.ncbi.nlm.nih.gov/pub/clinvar/vcf_GRCh38/clinvar.vcf.gz"
CLINVAR_INDEX_URL = "https://ftp.ncbi.nlm.nih.gov/pub/clinvar/vcf_GRCh38/clinvar.vcf.gz.tbi"
REFERENCE_FASTA = "Homo_sapiens.GRCh38.dna.primary_assembly.fa"
log_file = "pipeline.logs"

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s [%(levelname)s] %(message)s",
    handlers=[
        logging.FileHandler(log_file),
        logging.StreamHandler()
    ]
)

logger = logging.getLogger(__name__)

# Запуск команд
```

```
def run(cmd, shell=False):
    cmd_str = cmd if shell else ' '.join(cmd)
    logger.info(f"START {cmd_str}")
    subprocess.run(cmd, check=True, shell=shell)



# Завантаження генетичних даних
def download_sra(run_id: str):
    logger.info("Download SRA")
    subprocess.run(["prefetch", run_id, "--type", "sra"], check=True)
    subprocess.run([
        "fasterq-dump", run_id
    ], check=True)
#   Завантаження та підготовка референсу
def prepare_reference():
    logger.info("Prepare ref")
    fasta = Path(REFERENCE_FASTA)

    if not fasta.exists():
        gz = fasta.with_suffix(".fa.gz")

        if not gz.exists():
            run(["wget", REFERENCE_URL])

        run(["gunzip", "-k", str(gz)])

    bwa_index_files = [
        fasta.with_suffix(".fa.amb"),
        fasta.with_suffix(".fa.ann"),
        fasta.with_suffix(".fa.bwt"),
```

```
            fasta.with_suffix(".fa.pac"),
            fasta.with_suffix(".fa.sa")
        ]

        if not all(f.exists() for f in bwa_index_files):
            run(["bwa", "index", REFERENCE_FASTA])


    #   Вирівнювання, (збирання пазлу) FASTQ -> BAM
    def align_reads(run_id: str):
        logger.info("Align reads")
        r1 = f"{run_id}_1.fastq"
        r2 = f"{run_id}_2.fastq"

        sam = f"{run_id}.sam"
        bam = f"{run_id}.bam"
        sorted_bam = f"{run_id}.sorted.bam"

        run(f"bwa mem {REFERENCE_FASTA} {r1} {r2} > {sam}",
shell=True)
        run(f"samtools view -Sb {sam} > {bam}", shell=True)
        run(["samtools", "sort", bam, "-o", sorted_bam])
        run(["samtools", "index", sorted_bam])

        return sorted_bam


    # Варіантування (пошук мутацій), BAM -> VCF
    def call_variants(sorted_bam: str, run_id: str):
        logger.info("Calling variants")
```

```
    vcf = f"{run_id}.vcf"
    gz_vcf = f"{run_id}.vcf.gz"

    run(
        f"bcftools mpileup -f {REFERENCE_FASTA} {sorted_bam} | "
        f"bcftools call -mv -Ov -o {vcf}",
        shell=True
    )

    run(["bgzip", "-f", vcf])
    run(["tabix", "-p", "vcf", gz_vcf])

    return gz_vcf


#   Підвантаження та перевірка ClinVar
def prepare_clinvar():
    logger.info("Prepare clinvar")
    if not Path("clinvar.vcf.gz").exists():
        run(["wget", CLINVAR_URL])

    if not Path("clinvar.vcf.gz.tbi").exists():
        run(["wget", CLINVAR_INDEX_URL])


#   Порівняння з ClinVar, знаходження мутацій
def intersect_with_clinvar(vcf: str, output_dir="clinvar_matches"):
    logger.info("Intersect with clinvar")
    Path(output_dir).mkdir(exist_ok=True)
```

```
    run([
        "bcftools", "isec",
        "-p", output_dir,
        vcf,
        "clinvar.vcf.gz"
    ])

    print("\n[INFO] Results:")
    print(f"  {output_dir}/0000.vcf -> only your variants")
    print(f"  {output_dir}/0001.vcf -> only ClinVar variants")
    print(f"  {output_dir}/0002.vcf -> common variants (important)")


#  Фільтрування за хромосомою
def extract_chromosome(chromosome: str):
    logger.info("Extract chr")
    output = f"clinvar_{chromosome}.vcf"

    run(
        f"bcftools view -r {chromosome} clinvar.vcf.gz -Ov -o {output}",
        shell=True
    )

    print(f"[INFO] Saved chromosome-specific ClinVar file: {output}")


#  Окрестрація етапів та запуск застосункку
def main():
    parser = argparse.ArgumentParser(
        description="Fastq -> Sam -> Bam -> VCF > Diagnosis"
```

```
    )
parser.add_argument(
        "--run",
        required=True,
        help="SRA Run ID"
    )

    parser.add_argument(
        "--skip-download",
        action="store_true",
        help="Skip SRA download step"
    )

    args = parser.parse_args()

    run_id = args.run

    if not args.skip_download:
        download_sra(run_id)

    prepare_reference()

    sorted_bam = align_reads(run_id)
    vcf = call_variants(sorted_bam, run_id)

    prepare_clinvar()
    intersect_with_clinvar(vcf)

    if args.chromosome:
        extract_chromosome(args.chromosome)
```

```
    logger.info("Start Diagnosis")
    run_diagnosis("report.csv")
    logger.info("End Diagnosis")

    if __name__ == "__main__":
        main()
```

A.2 Текст файлу diagnosis.py

```
import pandas as pd
import pysam

#   Завантаження бази із захворюваннями
def load_clinvar(CLINVAR_VCF):

    data = []

    vcf = pysam.VariantFile(CLINVAR_VCF)

    for record in vcf:
        if record.alts is None:
            continue

        if len(record.alts) == 0:
            continue

        key = (
            record.chrom,
            record.pos,
```

```
            record.ref,
            str(record.alts[0])
        )

        data.append({
            "chrom": record.chrom,
            "pos": record.pos,
            "gene": record.info.get("GENEINFO", "unknown"),
            "alt": str(record.alts),
            "ref": record.ref,
            "clinical_significance": record.info.get("CLNSIG", "unknown"),
            "disease": record.info.get("CLNDN","unknown")
        })

    vcf.close()

    return pd.DataFrame(data)

#   Зчитування нашого VCF з попереднього етапу
def load_sample(SAMPLE_VCF):
    data = []
    vcf = pysam.VariantFile(SAMPLE_VCF)

    for record in vcf:
        if record.alts is None or len(record.alts) == 0:
            continue

        data.append({
            "chrom": record.chrom,
            "pos": record.pos,
```

```
                "alt": str(record.alts),
                "ref": record.ref
            })

        vcf.close()
        return data

    # Порівняння нашого VCF з базою захворювань
    def compare(CLINVAR_VCF, SAMPLE_VCF):
        clinvar_df = pd.DataFrame(load_clinvar(CLINVAR_VCF))
        sample_df = pd.DataFrame(load_sample(SAMPLE_VCF))

        result = sample_df.merge(
            clinvar_df,
            on=["chrom", "pos", "alt", "ref"],
            how="inner"
        )
        return result

    #   Виклик функцій із збереженням результатів діагностики до CSV
    def run_diagnosis(CLINVAR_VCF, SAMPLE_VCF,
out_file="report.csv",):
        report = compare(CLINVAR_VCF, SAMPLE_VCF)
        report.to_csv(out_file, index=False)
```

ДОДАТОК Б

Слайди презентацій

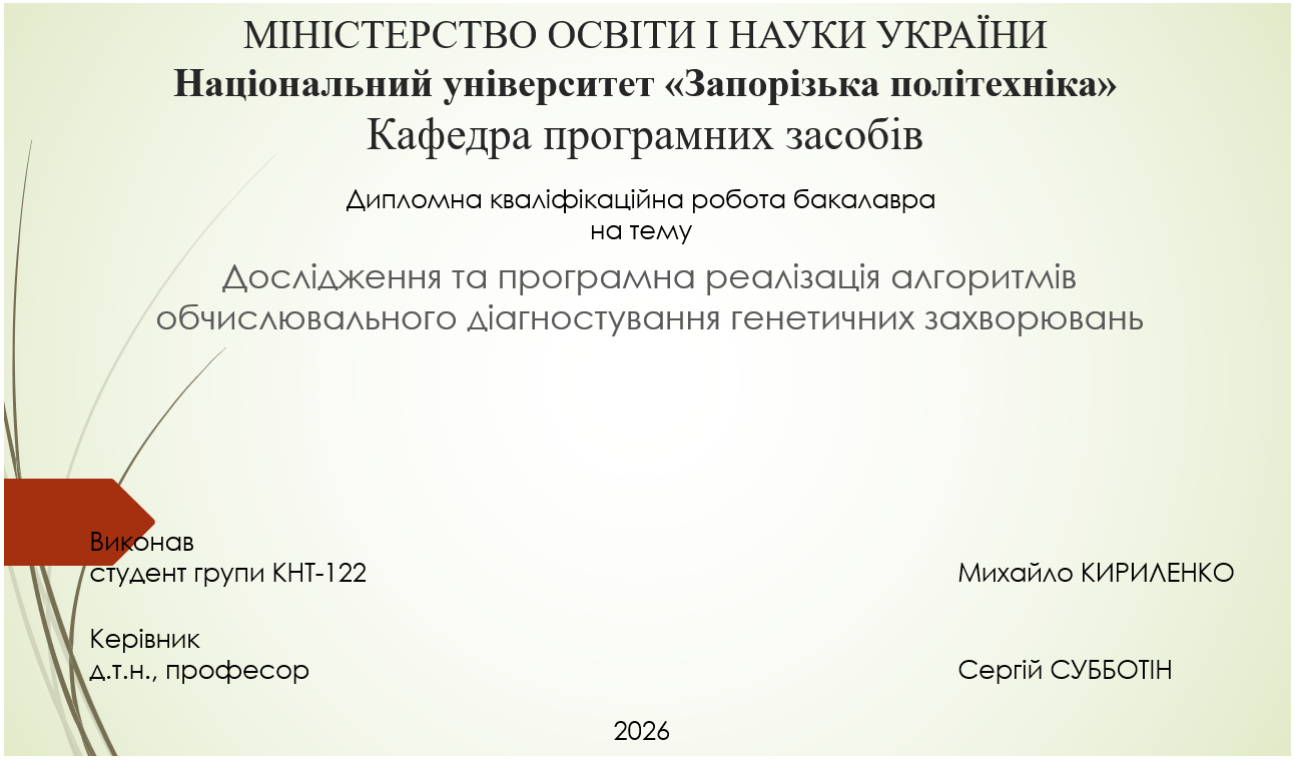

Рисунок Б1 – Слайд 1

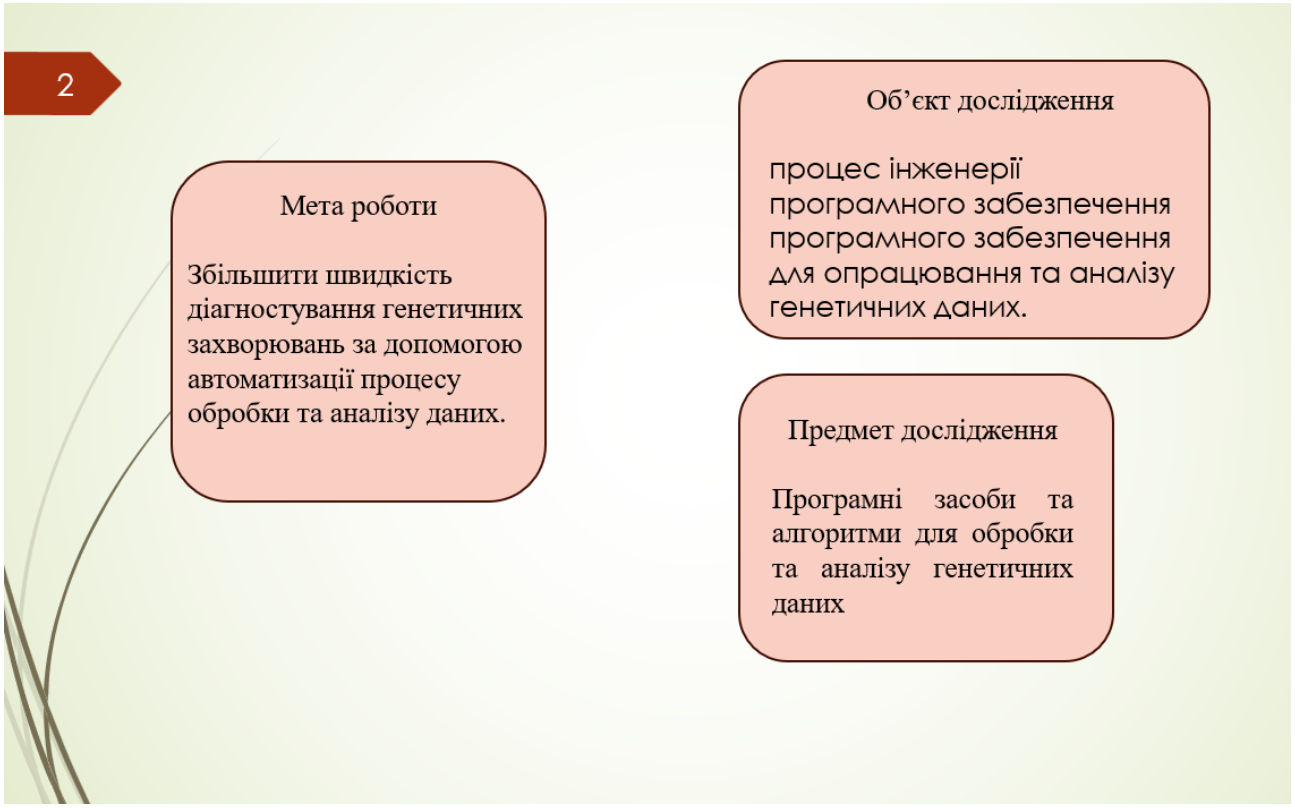

Рисунок Б2 – Слайд 2

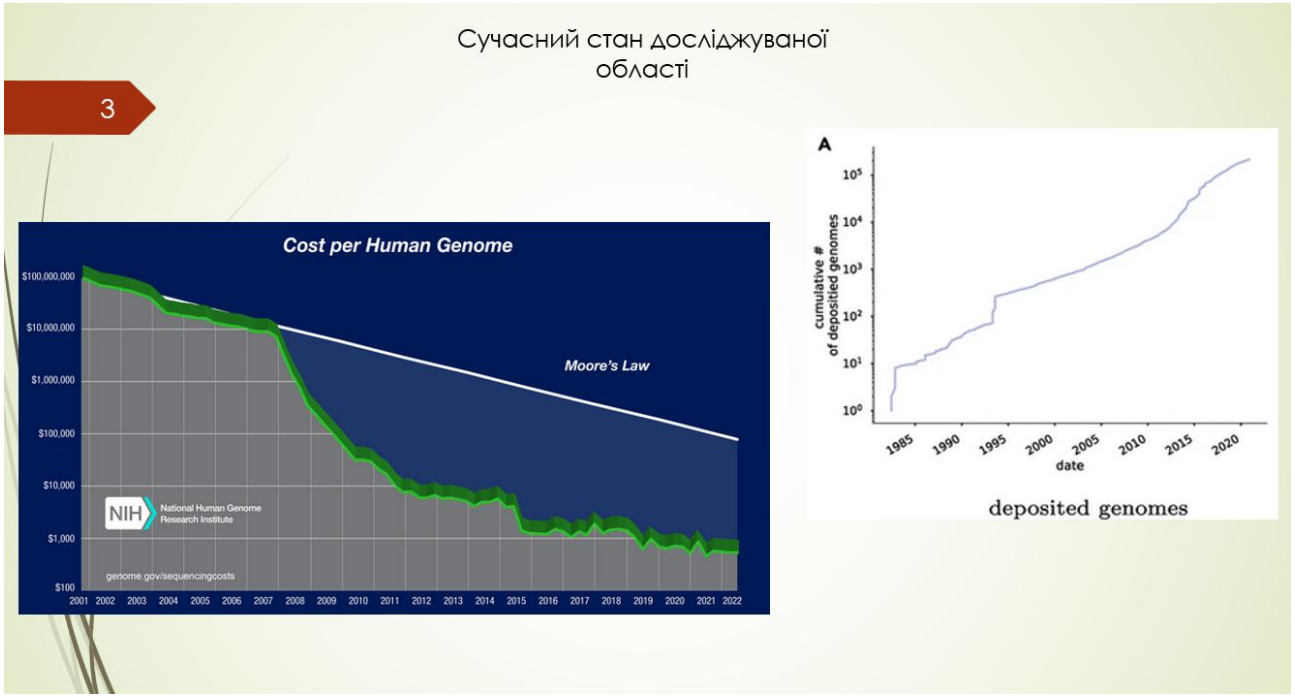

Рисунок Б3 – Слайд 3

Критерій	Fastq2vcf	SeqMule
Автоматизація	Часткова	Висока
Складність використання	Середня	Середня
Кількість алгоритмів	Невелика	Значна
Моніторинг процесів	Відсутній	Логи
Вимоги до ресурсів	Помірні	Високі
Налаштування	Середнє	Високе

Рисунок Б4 – Слайд 4

Рисунок Б5 – Слайд 5

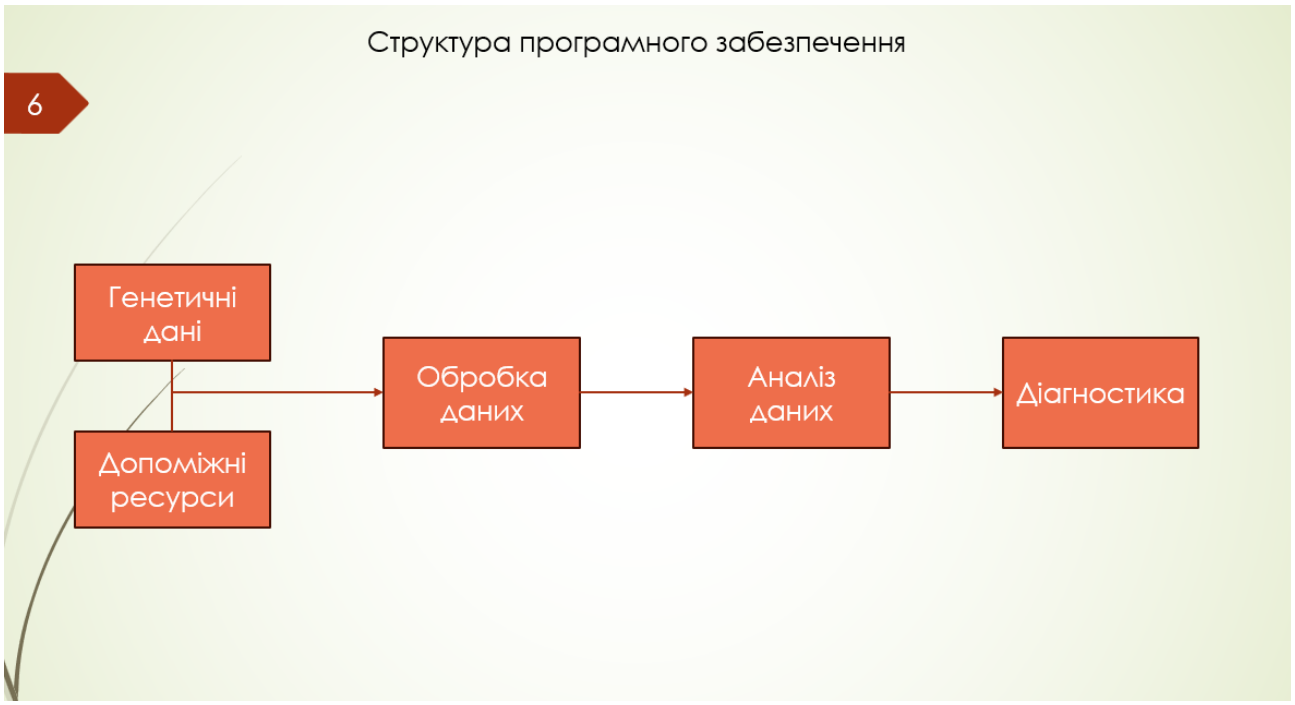

Рисунок Б6 – Слайд 6

Рисунок Б7 – Слайд 7

Рисунок Б8 – Слайд 8

Рисунок Б9 – Слайд 9

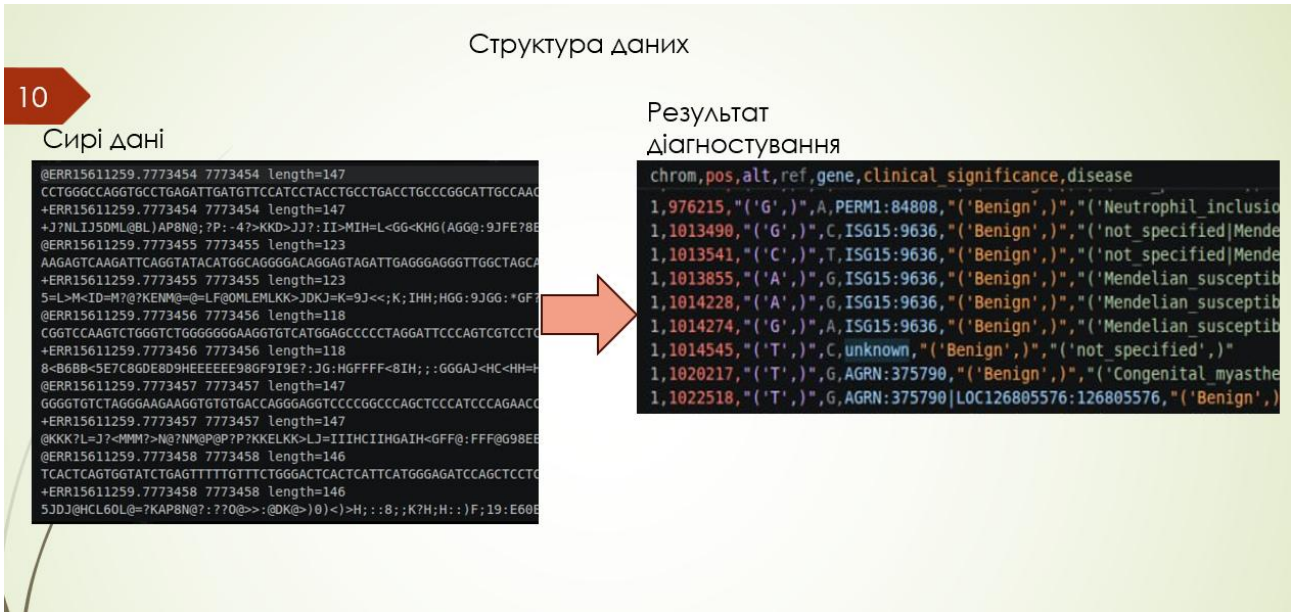

Рисунок Б10 – Слайд 10

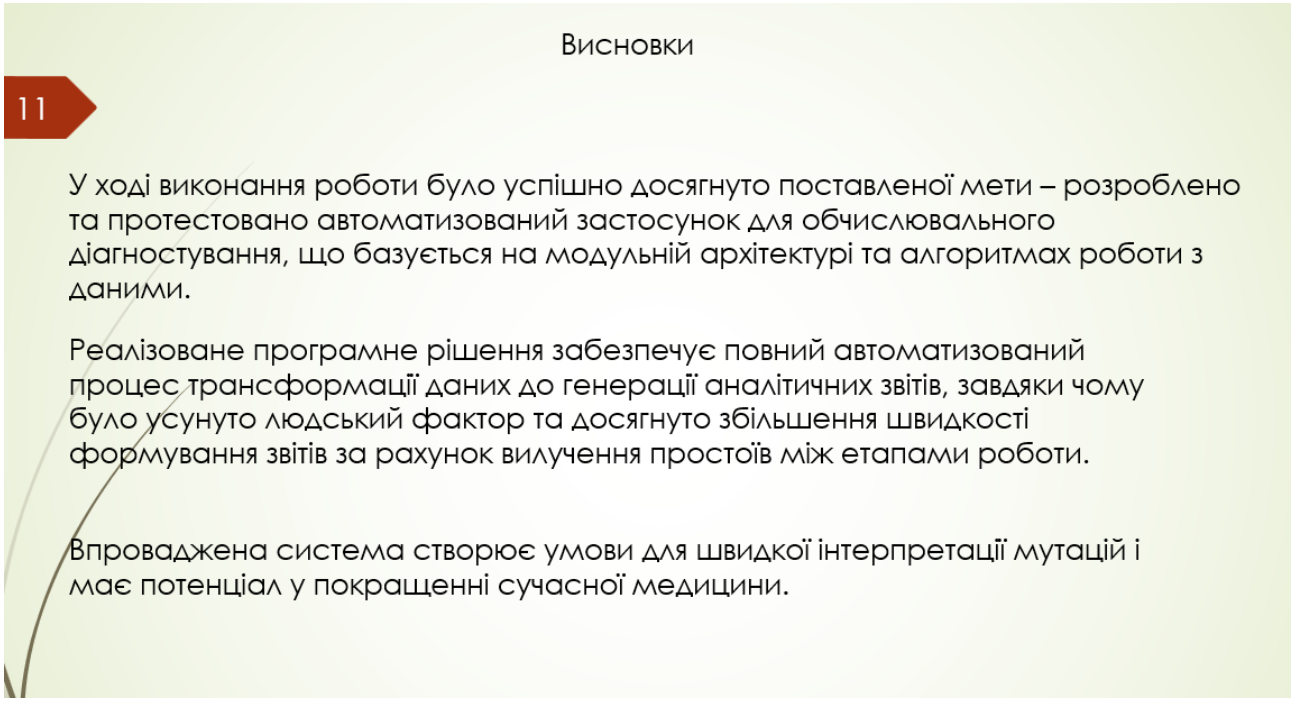

Рисунок Б11 – Слайд 11