Форма № Н-9.02

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ
імені адмірала Макарова

Херсонський навчально-науковий інститут

Кафедра інформаційних технологій та фізико-математичних дисциплін

"Допущений до захисту"
Завідувач кафедри
д.т.н., доцент Гучек П.Й.

"16" грудня 2025 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи
на здобуття ступеня вищої освіти "магістр"

на тему: "Дослідження технології та розробка програмної платформи "Цифровий двійник" дизельного двигуна на базі моделі повного робочого циклу"

Виконав: студент VI курсу, групи 6157зм
спеціальності
122 - "Комп'ютерні науки", ОПП -
(шифр і назва спеціальності та освітньо-професійної програми)
"Інформаційні управляючі системи та технології"
Шалапко Д.О.
(прізвище та ініціали)

Керівник к.ф.-м.н., доц. Латанська Л.О.
(прізвище та ініціали)

Рецензент к.т.н., доц. Л.М.Макарова
(прізвище та ініціали)

Херсон - 2025 р.

Форма № Н-9.01

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ
імені адмірала Макарова
ХЕРСОНСЬКИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ

Кафедра інформаційних технологій та фізико-математичних дисциплін
Освітній ступень магістр
Галузь знань 12 – "Інформаційні технології"
(шифр і назва)
Спеціальність 122 – " Комп'ютерні науки"
(шифр і назва)
Освітньо-професійна програма "Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ
Гарант освітньої програми
Гучек П.Й.
"23" жовтня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
на здобуття ступеня вищої освіти «магістр»

Студенту Шалапку Денису Олеговичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікацій роботи "Дослідження технології та розробка програмної платформи "Цифровий двійник" дизельного двигуна на базі моделі повного робочого циклу"

Керівник роботи Латанська Людмила Олексіївна, к.ф.-м.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Тема затверджена розпорядженням по інституту від "17" жовтня 2025 року № 20

2. Термін подання студентом роботи 12.12.2025 р.

3. Вихідні дані до роботи:

4. Зміст та структура розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

4.1 Титульний аркуш, завдання на магістерську роботу, анотація (українською, російською, англійською), зміст, перелік умовних означень, символів, одиниць та термінів (при необхідності).

4.2 Вступ.

4.3 Дослідження технологій проектування інформаційних систем та аналіз їх особливостей.

4.4 Формування вимог до інформаційної системи, розробка її концепції і постановка задачі.

4.5 Розробка проектних рішень по інформаційній системі

4.6 Реалізація проекту інформаційної системи (технічна документація по проекту).

4.7 Розділ з економіки

4.8 Розділ з охорони праці та безпеки в надзвичайних ситуаціях (охорони оточуючого середовища).

4.9 Висновки.

4.10 Список використаної літератури

4.11 Додатки (технічне завдання, загальний опис ІС, інструкція користувача, текст програми).

5. Перелік презентаційних матеріалів

5.1 Схема організаційної структури підприємства.

5.2 Архітектура автоматизованої ІС.

5.3 Організаційно-функціональна структура автоматизованої ІС.

5.4 Інформаційна модель програмного забезпечення автоматизованої ІС.

5.5 Модель переходів станів програмного забезпечення автоматизованої ІС.

5.6 Модель технологічного забезпечення автоматизованої ІС.

5.7 Інтерфейс програми

5.8

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
4.7	Ломоносов Дмитро Анатолійович		
4.8	Щедролосєв Олександр Вікторович		

7. Дата видачі завдання 27 жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів роботи	Примітка
1.	Вивчення і аналіз предметної галузі	28.10.2025	
2.	Розробка концепції ІС	30.10.2025	
3.	Розробка проектних рішень по ІС	31.10.2025	
4.	Реалізація проекту, розробка робочої документації	03.11.2025	
5.	Вирішення питань з економіки	06.11.2025	
6.	Вирішення питань охорони праці та безпеки в надзвичайних ситуаціях (охорони оточуючого середовища)	12.11.2025	
7.	Виконання графічної частини роботи	22.11.2025	
8.	Оформлення пояснювальної записки	06.12.2025	
9.	Подання роботи на попередній захист	12.12.2025	
10.	Подання роботи рецензенту	20.12.2025	
11.	Подання роботи на державний захист	29.12.2025	

Студент _____ Шалапко Д.О.
Підпис Прізвище та ініціали

Керівник роботи _____ Латанська Л.О.
Підпис Прізвище та ініціали

Анотація

У магістерській роботі виконано дослідження технологічних підходів та розроблено програмну платформу класу “цифровий двійник” дизельного двигуна на базі моделі повного робочого циклу. Реалізовано обчислювальне ядро на Python, що забезпечує послідовний розрахунок процесів наддуву, наповнення, стискання, згоряння та розширення, а також формування індикаторних залежностей і енергетичних показників. Вхідні дані організовано у форматі конфігураційних файлів (JSON) з можливістю накопичення результатів у структурованому вигляді для подальшого аналізу та візуалізації. Передбачено інтерактивне представлення результатів у вигляді таблиць та графіків і підготовлено основу для розширення платформи до веб-сервісу з API для інтеграції з зовнішніми системами моніторингу або цифровими стендами. Практична цінність роботи полягає у створенні відтворюваного програмного інструменту для інженерних розрахунків, навчальних задач та попередньої оцінки впливу параметрів паливоподачі й умов роботи на показники двигуна.

Робота представлена на 126 сторінках друкованого тексту та містить 12 рисунків, 1 таблицю, 5 додатків та список використаної літератури з 51найменування.

Ключові слова: цифровий двійник, дизельний двигун, робочий цикл, індикаторна діаграма, моделювання, Python, візуалізація, API.

Abstract

This master's thesis explores key technological approaches and presents a software platform of a diesel-engine "digital twin" based on a full-cycle simulation model. A Python-based computational core is implemented to perform a consistent calculation of boosting, intake, compression, combustion, and expansion processes, as well as to generate indicator trends and key performance metrics. Input data are organized as configuration files (JSON) with structured persistence of results for further analysis and visualization. The platform provides interactive output in the form of tables and plots, establishing a foundation for future extension into a web service with an API to integrate external monitoring systems or digital test benches. The practical value of the thesis lies in providing a reproducible software tool for engineering calculations, educational purposes, and preliminary assessment of how fuel-injection settings and operating conditions impact diesel-engine performance.

The work is presented on 126 pages of printed text and contains 12 figures, 1 table, 5 appendices, and a list of 51 references.

Keywords: digital twin, diesel engine, complete full cycle model, indicator diagram, simulation, Python, visualization, API.

Зміст

Перелік умовних означень, символів, одиниць та термінів

Перелік умовних означень і скорочень:

- **ДВЗ** – двигун внутрішнього згоряння
- **ККД** – коефіцієнт корисної дії
- **ЦТ** – цифровий двійник
- **API** – Application Programming Interface
- **JSON** – JavaScript Object Notation
- **IMEP** – indicated mean effective pressure (середній індикаторний тиск)
- **BMEP** – brake mean effective pressure (середній ефективний тиск)

Основні позначення (приклад для роботи):

- D – діаметр циліндра, м
- S – хід поршня, м
- ε – ступінь стискання
- n – частота обертання колінчастого вала, об/хв
- P – тиск, Па або кПа (залежно від розділу)
- T – температура, К
- V – об’єм, м^3
- α – коефіцієнт надлишку повітря
- η_i – індикаторний ККД
- η_m – механічний ККД
- g_e – питома ефективна витрата палива, г/(кВт·год)

Вступ

Сучасні вимоги до енергоефективності, економічності та екологічності дизельних двигунів формують потребу в інструментах, які дозволяють швидко оцінювати вплив конструктивних і режимних параметрів на показники робочого процесу. У промисловості та на транспорті дедалі ширше застосовуються концепції цифрових двійників, що поєднують математичну модель об’єкта, дані вимірювань і програмні засоби аналізу для прогнозування, оптимізації та підтримки прийняття рішень. Для дизельного двигуна цифровий двійник є особливо актуальним, оскільки робочий цикл визначається взаємодією багатьох фізичних підпроцесів: газообміну, стискання, впорскування та сумішоутворення, згоряння, розширення, теплопередачі та гідродинамічних втрат.

У навчальній і прикладній практиці використовуються різні підходи до моделювання дизельного циклу – від аналітичних наближених розрахунків до деталізованих 1D/3D моделей. Водночас високоточні комерційні пакети часто мають обмеження щодо вартості, закритості алгоритмів та складності інтеграції з власними програмними рішеннями. Для задач спеціальності «Комп’ютерні науки» ключовими стають відтворюваність розрахунку, прозорість структури даних, можливість розширення, автоматизації експериментів і побудови інтерактивних інтерфейсів та сервісів.

У межах даної магістерської роботи розглядається розробка програмної платформи «Цифровий двійник» дизельного двигуна на базі Python-моделі повного робочого циклу. Платформа орієнтована на послідовний алгоритмічний конвеєр обчислень, де вхідні параметри (геометрія, умови наддуву, характеристики палива, параметри впорскування та коефіцієнти моделі) задаються у конфігураційних файлах, а результати формуються у вигляді структурованих наборів даних і візуальних залежностей. Модель забезпечує розрахунок термодинамічних параметрів у характерних точках циклу та інтегрування параметрів у функції кута повороту колінчастого вала, що дозволяє будувати індикаторні діаграми, визначати роботу циклу та отримувати інтегральні показники ефективності.

Практична цінність роботи полягає у створенні відкритої та розширюваної програмної основи, придатної для інженерних розрахунків і навчальних задач, а також як бази для подальшого розвитку у напрямі веб-сервісу з API та інтерактивною візуалізацією. Це дає можливість використовувати розроблений інструмент як ядро цифрового двійника для порівняння режимів, проведення серій параметричних досліджень та інтеграції з зовнішніми джерелами даних у перспективних системах моніторингу й оптимізації роботи дизельних установок.

1 Дослідження технологій проектування інформаційних систем та аналіз їх особливостей

Проєктування програмної платформи класу «цифровий двійник» для дизельного двигуна доцільно розглядати як проєктування інформаційної системи (ІС), а не лише «програми для розрахунку». Це пояснюється тим, що цифровий двійник має забезпечити не одноразове отримання чисел, а цілісний цикл роботи з моделлю та результатами: від введення параметрів і запуску сценарію до накопичення, відтворення, аналізу, порівняння та експорту даних. У межах такої ІС логічно виділити чотири взаємопов'язані складові.

(1) Ядро обчислювальної моделі.

Це центральний компонент, який реалізує фізико-математичну модель повного робочого циклу дизеля та алгоритмічний конвеєр розрахунку. Ядро повинно приймати структурований набір вхідних параметрів (геометрія циліндра, ступінь стиснення, частота обертання, параметри наддуву, характеристики палива, налаштування впорскування, коефіцієнти моделей теплообміну/газообміну тощо) і формувати вихідні масиви станів та інтегральні показники. Практично це означає, що ядро має генерувати профілі величин у функції кута колінчастого вала φ: $p(\varphi), T(\varphi), V(\varphi), x(\varphi), dx/d\varphi$, а також узагальнені метрики циклу (індикаторна робота, середні тиски, ККД, витрати). Важливим є те, що ядро має бути детермінованим: за однакових входів воно повинно давати однакові виходи, і не залежати від того, чи запускається розрахунок із консолі, GUI чи через веб-API.

(2) Підсистема керування даними.

У цифровому двійнику дані – це не «додаток», а основа відтворюваності. Підсистема даних повинна вирішувати кілька задач:

- зберігання вхідних параметрів (конфігурації двигуна, режиму, палива, налаштувань моделей);
- ведення довідників (типи палива, емпіричні коефіцієнти, типові конфігурації);
- збереження результатів розрахунку у стандартизованому форматі (масиви, таблиці, метрики);
- підтримка версіонування запусків (ідентифікатор розрахунку, дата/час, версія моделі, коментарі користувача).

У практичній реалізації на Python це зазвичай починається з JSON/CSV як простих, прозорих форматів. Однак при переході до платформи логічним кроком є використання БД (навіть локальної SQLite), щоб забезпечити пошук і

порівняння серій запусків, зберігати історію експериментів і будувати звітність без ручного «перетягування файлів».

(3) Інтерфейс взаємодії користувача та/або зовнішніх сервісів.

Цифровий двійник має бути не тільки «інтерактивним», а й програмно керованим. Це означає підтримку двох типів взаємодії:

- людина ↔ система (GUI/веб-інтерфейс/консоль): вибір конфігурації, запуск сценарію, перегляд графіків, експорт таблиць;
- система ↔ система (веб-API): передача параметрів як структури даних, запуск розрахунку, отримання результатів у форматі JSON/CSV для інтеграції з іншими компонентами (наприклад, з фронтендом, оптимізатором режимів, модулем калібрування, навчальними симуляторами).

Тобто «інтерфейс» у цифровому двійнику – це набір правил і форматів взаємодії, які гарантують, що обчислення можна запускати повторно, автоматизовано і в складі більшого процесу (серії експериментів, параметричні дослідження, порівняння альтернативних палив).

(4) Підсистема візуалізації та звітності.

Оскільки модель генерує великі масиви даних, потрібні засоби перетворення чисел у форму, зручну для інженерного аналізу. Для дизельного циклу це насамперед: індикаторна діаграма $p(\varphi)$, залежності $T(\varphi)$, профілі тепловиділення $x(\varphi)$ і $dx/d\varphi$, графіки газообміну, а також таблиці інтегральних показників (робота, $p_i, p_e, \eta_i, \eta_e, g_i, g_e, N_i, N_e$). Звітність може включати формування підсумкових таблиць, порівняльних графіків «база/варіант», експорт у формати для включення у документи (наприклад, таблиці та рисунки для LaTeX).

Саме ця підсистема робить цифровий двійник зручним як інструмент дослідження: дозволяє швидко виявляти відхилення, оцінювати вплив параметрів, перевіряти фізичну узгодженість результатів і документувати серії запусків.

Таким чином, на відміну від «звичайної» інженерної програми, цифровий двійник має працювати як керований сервіс. Під керованим сервісом у даному контексті мається на увазі, що система:

- приймає параметри в стандартизованому вигляді;
- виконує розрахунок за визначеним сценарієм;
- фіксує (зберігає) як входи, так і виходи;
- забезпечує можливість повторити запуск і отримати ті самі результати;
- надає результати у формах, зручних для аналізу, порівняння та інтеграції.

1.1 Загальні підходи до проєктування ІС

У сучасній практиці застосовують декілька базових підходів до проєктування інформаційних систем. Для платформи «цифровий двійник» корисно розглянути їх крізь призму того, як саме організується логіка, дані та взаємодія компонентів.

Функціонально-орієнтований підхід.

Система описується як послідовність функцій і потоків даних за схемою «вхід → обробка → вихід». У задачі моделювання дизельного циклу цей підхід природно відповідає алгоритмічному конвеєру: зчитування параметрів → розрахунок базових величин → моделювання фаз циклу → формування масивів і метрик → виведення та збереження. Його головна перевага – простота: легко побачити, що за чим виконується, та швидко отримати працюючий прототип. Однак зі зростанням проєкту (додавання альтернативних моделей тепловиділення, різних палив, режимів наддуву, сценаріїв інтеграції, форматів експорту) виникає проблема масштабування: збільшується кількість умовних гілок, залежностей між функціями, повторів коду. У результаті «лінійний» пайплайн перетворюється на важкокеровану структуру, де складно вносити зміни без ризику порушити інші сценарії.

Об’єктно-орієнтований підхід.

Предметна область подається як набір сутностей, кожна з яких має стан (атрибути) та поведінку (методи). Для цифрового двійника дизельного двигуна такими сутностями є: двигун і його геометрія, паливо та його властивості, система впорскування, робочий цикл, підмоделі (теплообмін, згоряння, газообмін), результати та метрики. Перевага підходу – кероване розширення: можна замінити або доповнити підмодель, не переписуючи весь код, за умови збереження контрактів (вхід/вихід об’єктів). Це важливо для магістерської роботи, бо дозволяє показати «комп’ютерно-науковий» аспект: коректне проектування структури, розмежування відповідальності, інкапсуляція даних та повторне використання компонентів. Недолік полягає в тому, що потрібне більш суворе початкове проектування: слід визначити, які класи є ключовими, які дані де зберігаються, які методи є публічними, як стандартизуються результати.

Компонентний/сервісний підхід.

Система розбивається на модулі (компоненти), які взаємодіють через інтерфейси. У розподілених архітектурах це може розвиватися у мікросервіси, але навіть у межах одного Python-проєкту компонентність є критичною. Для цифрового двійника це означає відокремлення:

- обчислювального ядра (модель);

- керування даними (формати/БД/версії);
- API (контракти запит/відповідь);
- UI/візуалізації (графіки/таблиці/звіти).

Перевага – незалежний розвиток частин: можна удосконалювати модель, не змінюючи API; змінити спосіб збереження (JSON → БД), не торкаючись обчислень; підключити новий веб-інтерфейс, не переписуючи фізику. Саме цей підхід робить цифровий двійник «платформою», а не одноразовим застосунком.

З огляду на тематику магістерської роботи та перспективу розвитку у формат «платформи», доцільно комбінувати об’єктно-орієнтоване моделювання предметної області (для структури й розширюваності) з компонентною архітектурою (для незалежності ядра, даних, API та візуалізації). При цьому функціонально-орієнтований опис залишається корисним для документування пайплайна на верхньому рівні: він чітко показує логіку «параметри → розрахунок → результат», що важливо для пояснення системи в тексті магістерської роботи.

1.2 Життєвий цикл проєктування та артефакти

Життєвий цикл розробки інформаційної системи традиційно описують через послідовність етапів: аналіз вимог → проєктування → реалізація → тестування → впровадження → супровід. Для програмної платформи класу «цифровий двійник» (тобто системи, яка не просто «рахує», а забезпечує кероване відтворюване моделювання) критично важливо, щоб кожен етап породжував формалізовані артефакти, які роблять продукт перевірюваним, повторюваним і придатним до розвитку.

Етап 1. Аналіз вимог.

На цьому етапі формулюється *що саме* повинна робити система і *в яких межах*. Для цифрового двійника дизельного двигуна вимоги мають два рівні: інженерний (які фізичні процеси та показники моделюються) і програмний (як організований запуск, збереження, доступ до результатів). Результатом етапу є артефакти:

- вимоги (функціональні та нефункціональні);
- сценарії використання (користувач запускає розрахунок режиму; порівнює два палива; формує звіт; робить серію параметричних прогонів);
- обмеження (діапазони параметрів, допустимі одиниці вимірювання, крок інтегрування по φ, контроль чисельної стійкості).

Етап 2. Проєктування.

Тут вимоги перетворюються на структуру системи: як виглядають дані, як вони проходять через модулі, де зберігаються результати, як забезпечується відтворюваність. Для платформи «цифровий двійник» ключовими артефактами є:

- модель даних: які параметри є обов'язковими (геометрія, $\varepsilon, n, p_0, T_0, \pi_k$, параметри впорскування, властивості палива), які факультативні (коефіцієнти підмоделей, налаштування теплообміну/газообміну), і які результати повинні зберігатися завжди (масиви $p(\varphi), T(\varphi)$, інтегральні метрики);
- архітектура модулів: що є «ядром» (обчислювальна модель), що є підсистемою даних (JSON/CSV/БД), що є API/інтерфейсом, що є візуалізацією;
- контракти взаємодії: які структури даних передаються між компонентами, які ключі/поля є стандартними, як кодуються одиниці вимірювання.

Етап 3. Реалізація.

Реалізація – це не «написання всього в одному файлі», а побудова коду так, щоб він відповідав запроєктованим контрактам. Для інженерно-обчислювальної системи важливо, щоб реалізація одразу підтримувала:

- стандартизований ввід параметрів (наприклад, конфіг-файли JSON);
- автоматичне формування структури результатів (таблиці + масиви);
- механізми логування (ідентифікатор запуску, дата/час, версія моделі, ключові налаштування);
- експорт для інтеграції та звітності (CSV/JSON і підготовка даних для LaTeX-графіків/таблиць).

Етап 4. Тестування та валідація.

Для цифрового двійника тестування не обмежується unit-тестами функцій. Потрібні також перевірки фізичної коректності та регресійні набори, які гарантують, що зміни в коді не «ламають» результат. Типові артефакти:

- план тестування: що саме перевіряємо, які критерії приймання;
- контроль одиниць (узгодженість кПа/МПа/Па, градуси/радіани, кДж/Дж тощо);
- регресійні кейси: набір параметрів, для якого «еталонні» результати збережено, і нові версії мають давати ті ж значення в межах допуску;
- перевірки фізичної узгодженості (монотонність окремих ділянок, допустимі межі T, p, відсутність невалідних значень тощо).

Етап 5. Впровадження.

Для платформи це означає визначення способу запуску: локально (для налагодження і навчальних задач) або як сервіс (сервер з API). Артефакти:

- інструкція розгортання;
- конфігурації середовища;
- опис версійності (версія моделі, сумісність форматів даних).

Етап 6. Супровід і розвиток.

Цифровий двійник – система, яка еволюціонує: додаються нові підмоделі, калібрування коефіцієнтів, нові типи палива, інтеграція з вимірювальними даними. Тому підтримка документації та тестів стає частиною продукту, а не «додатком».

Отже, проєктування ІС для цифрового двійника – це не лише програмування, а створення описаної, документованої і перевірюваної системи, де артефакти (вимоги, моделі даних, архітектура, протоколи, тести) забезпечують відтворюваність та інженерну довіру до результатів.

1.3 Технології архітектурного проєктування

Платформа «цифровий двійник» поєднує обчислення, зберігання та візуалізацію, тому архітектурний стиль визначає, наскільки система буде розширюваною, тестованою та придатною до інтеграції. Для подібних систем найчастіше використовують такі підходи.

Монолітна архітектура.

У моноліті всі підсистеми (обчислювальне ядро, робота з файлами/БД, UI/графіки) зосереджені в одному застосунку. Це зручно на ранньому етапі: простий запуск, швидке налагодження, мінімальні накладні витрати. Однак зі зростанням функціоналу моноліт починає ускладнюватися: важче замінювати окремі частини (наприклад, перейти з локальних графіків Matplotlib до веб-візуалізації), складніше розмежовувати відповідальність модулів, з’являється ризик «взаємопроникнення» логіки моделі і логіки інтерфейсу.

Шарова (layered) архітектура.

Шаровий підхід пропонує чітко виділити рівні:

- рівень представлення (UI або API);
- рівень прикладної логіки (сценарії запуску, керування розрахунком, формування звітів);
- рівень доступу до даних (завантаження/збереження JSON/CSV/БД);
- рівень обчислювального ядра (математична модель, чисельні методи, підмоделі процесів).

Перевага цього підходу для магістерської роботи в галузі «комп’ютерні науки» – у тому, що він забезпечує формальну структуру, полегшує тестування (ядро можна тестувати незалежно від UI/API), спрощує супровід і робить систему масштабованою за функціями. Саме шарова архітектура найкраще підтримує ідею «цифрового двійника як платформи».

Клієнт–сервер.

У цьому підході обчислювальне ядро працює як серверний сервіс, який надає API для запуску розрахунків і отримання результатів. Клієнтська частина

(браузерний інтерфейс) відповідає за введення параметрів, запуск сценаріїв та інтерактивну візуалізацію. Для цифрового двійника це природно, тому що:

- розрахунки можуть запускатися з різних пристроїв і середовищ;
- результати зручно представити у веб-форматі (інтерактивні графіки, порівняння прогонів);
- API відкриває шлях до інтеграції з іншими сервісами (оптимізація, ML-калібрування, підключення експериментальних даних).

1.4 Технології проєктування моделі даних та інформаційного забезпечення

У цифровому двійнику «DieselCycleUa» дані – це не просто “вхід і вихід”. Фактично система живе даними: приймає конфігурацію режиму та двигуна, формує проміжні стани по всіх підмоделях, накопичує профілі по куту колінчастого вала і в кінці видає як інтегральні показники, так і масиви для графіків та звітів. Тому модель даних потрібно будувати так, щоб вона була одночасно: зрозумілою, відтворюваною, придатною до автоматичного аналізу і стабільною при розвитку платформи.

У практичній реалізації доцільно розділяти дані на три логічні групи.

1) Вхідні дані.

Це все, що користувач (або зовнішній сервіс через API) задає перед запуском розрахунку. Для дизельного робочого циклу типовий набір включає:

- геометрію двигуна: D, S, ε, а також співвідношення кривошип–шатун λ;
- параметри наддуву і газообміну: $П_k$, коефіцієнти втрат/ефективності, параметри охолоджувача наддувочного повітря, граничні припущення для тисків/температур;
- умови середовища: T_0, P_0, відносна вологість φ;
- параметри впорскування: кут початку впорскування, тривалість φ_{vp}, а також налаштування для підмоделей затримки займання та випаровування;
- дані палива: елементний склад C, H, O, S, нижча теплота згоряння Q_n, а також фізичні властивості (густина, в’язкість) і коефіцієнти, які потрібні для підмоделей;
- коефіцієнти емпіричних залежностей (те, що у коді задається як константи або параметри – щоб у майбутньому їх можна було калібрувати).

Ключова вимога до вхідних даних для цифрового двійника – однозначність. Кожен параметр має бути або з одиницями, або з чітко зафіксованим стандартом (наприклад: тиски зберігаються в кПа у файлах, а всередині ядра переводяться в Па). Якщо цього не зробити, помилки виникають не там, де “падає програма”, а там, де з’являється фізично неправильний результат.

2) Проміжні дані.

Це те, що цифровий двійник “будує всередині” у процесі моделювання. У «DieselCycleUa» таких даних багато, і вони мають різну природу:

- параметри характерних точок циклу (на кшталт $T_a, P_c, T_z, P_z, T_b, P_b$);
- профілі по куту: $P(\varphi), T(\varphi)$, а також допоміжні функції, які у вашому алгоритмі реально існують – наприклад частка випарованого палива, швидкість випаровування, частка згорілого палива $x(\varphi), dx(\varphi)$;
- внутрішні розрахункові величини, без яких профілі не порахувати: поточний об’єм $V(\varphi)$, похідна $dV/d\varphi$, масові витрати на впуск/випуск, коефіцієнти теплообміну тощо.

Проміжні дані важливі не тільки для “цікавості”. Вони потрібні для відладки, валідації і пояснюваності моделі. Цифровий двійник відрізняється від “чорної скриньки” тим, що користувач може подивитись, *чому* змінилися p_{max} або g_e: через фазу впорскування, через наддув, через інші теплові втрати.

3) Вихідні дані.

Це результати, які споживає користувач або інші модулі платформи:

- індикаторна робота за цикл;
- середні індикаторні та ефективні тиски;
- η_i, η_e;
- питомі витрати палива;
- масиви для побудови індикаторної діаграми й допоміжних графіків;
- табличне представлення “всі параметри” для звіту.

Щоб ці три групи даних “жили” у системі без хаосу, потрібні нормальні формати і правила зберігання.

Формати зберігання і обміну.

Для «DieselCycleUa» природно використовувати комбінацію:

- JSON – для конфігурацій і результатів, які мають ієрархію (вхідні параметри + блоки проміжних величин + метадані запуску). Це дає акуратну структуру й просту інтеграцію з веб-API.
- CSV – для профілів і таблиць (масиви $P(\varphi)$, $T(\varphi)$, $x(\varphi)$), які зручно відкривати в Excel/LibreOffice або підтягувати в Pandas для аналізу.
- SQLite (за потреби) – коли з’являється режим “серії прогонів”: багато запусків з різними параметрами, пошук, порівняння, історія, версії. Це саме той випадок, коли файлова система вже починає заважати.

Принцип відтворюваності.

Для цифрового двійника варто зафіксувати правило: один запуск = один набір входів + один набір результатів + метадані. Тобто, збережений результат повинен містити:

- повний дамп вхідних параметрів;
- версію моделі/коду (хоча б вручну задану);
- налаштування кроку інтегрування;
- ідентифікатор запуску та час.

Тоді будь-який графік або таблиця стають не “картинкою”, а відтворюваним експериментом.

1.5 Технології проєктування обчислювального ядра та алгоритмічного конвеєра

Модель “повного робочого циклу” дизеля – це не один розрахунок за формулою. Це послідовність підзадач, де кожна наступна використовує результати попередньої. У вашому коді це видно дуже чітко: спочатку формується базова термодинаміка й характерні точки, далі – блоки впорскування та розпилу, потім – згоряння у кутовій області, і вже після цього – інтегрування по φ для отримання $P(\varphi), T(\varphi)$ та енергетики циклу.

Тому обчислювальне ядро треба проєктувати як алгоритмічний конвеєр з чіткими межами відповідальності.

1) Аналітичні та “точкові” розрахунки.

Це етапи, де результат – окремі значення характерних станів: наддув, температура перед циліндром, залишкові гази, параметри кінця стискання, оцінка максимумів тиску/температури тощо. Саме тут формуються величини типу T_a, P_c, T_z, P_z. Перевага цього блоку – швидкість і прозорість; він задає опорні точки, які далі використовуються у кутових розрахунках.

2) Побудова функцій у кутовій області.

Далі система переходить до того, що безпосередньо формує “цифровий двійник” як процес у часі/куті:

- кінематика циліндра: $V(\varphi)$ та $dV/d\varphi$;
- підмодель затримки займання та моменту займання;
- профілі випаровування і згоряння, включаючи $x(\varphi)$ і $dx(\varphi)$.

Цей рівень важливий тим, що він визначає форму індикаторної діаграми, а не лише інтегральні значення.

3) Чисельне інтегрування циклу.

Фінальний етап – інтегрування по φ, де розв’язуються рівняння балансу маси й енергії та формується масив $P(\varphi), T(\varphi)$. Саме цей результат потім використовується для:

- обчислення індикаторної роботи як інтегралу $\int P\ dV$;
- розрахунку p_i, p_e, η, g;
- побудови графіків.

Що це означає з точки зору проєктування ІС.

Щоб ядро не перетворилося на “великий скрипт”, для цифрового двійника потрібні три речі.

1. Ізоляція підмоделей.

Термодинаміка, впорскування/розпил, згоряння, теплообмін, газообмін, інтегратор – це різні домени. Навіть якщо на початку все знаходиться в одному файлі, логічно вони мають існувати як окремі блоки з чіткими входами/виходами.

2. Стандартизовані структури обміну.

Потрібно мати домовленість, у якому вигляді підмоделі передають дані: словник/структура з фіксованими ключами, де значення з одиницями або з відомим стандартом. Це особливо важливо, коли з’являється веб-API: API по суті “оголює” внутрішні структури назовні.

3. Контроль одиниць і діапазонів.

Для фізичних розрахунків найнебезпечніші помилки – ті, що не дають винятку. Наприклад, переплутати кПа і Па, градуси і радіани, або дати параметр поза фізично допустимим діапазоном. Тому ядро має містити перевірки типу: тиск > 0, температура в адекватних межах, $\varepsilon > 1$, крок інтегрування не нульовий, маса заряду додатна, відсутність NaN/inf у профілях тощо.

У підсумку, обчислювальне ядро «DieselCycleUa» доцільно будувати як послідовний конвеєр: вхідні параметри → характерні точки → кутові профілі → інтегрування циклу → метрики та візуалізація, де кожен крок дає не тільки число “на виході”, а і проміжні дані, що дозволяють перевірити правильність розрахунку та пояснити отримані результати.

1.6 Технології проєктування інтерфейсу та візуалізації

Для програмної платформи класу «цифровий двійник» недостатньо отримати числові значення на виході. Користувачеві потрібно бачити поведінку процесу і розуміти, за рахунок чого змінюються інтегральні показники. У випадку дизельного робочого циклу це означає, що візуалізація має виконувати дві функції одночасно: (1) демонструвати фізичну картину циклу, (2) служити інструментом перевірки моделі (чи виглядають криві та співвідношення так, як повинні виглядати у реальному процесі).

Найбільш інформативними для «DieselCycleUa» є графіки, які прямо відповідають внутрішнім масивам і функціям моделі:

- індикаторна діаграма $P(\varphi)$ – центральний графік, за яким оцінюють характер згоряння, жорсткість процесу, фазові особливості та опосередковано коректність інтегрування;

- кінематичний профіль $V(\varphi)$ та, за потреби, похідна $dV/d\varphi$ – як базова перевірка правильності геометрії та кінематики;
- профілі згоряння $x(\varphi)$ і $dx/d\varphi$ – вони показують, де саме (в яких фазах) формується основне тепловиділення, і чи не виникають нефізичні "скачки";
- допоміжні профілі, які корисні для аналізу впорскування та випаровування: частка випарованого палива, швидкість випаровування, моменти $\varphi_{inj}, \varphi_v, \varphi_{kv}, \varphi_{kr}$ як вертикальні мітки на графіках.

Разом із графіками потрібна таблична частина, оскільки у звітах і при порівняннях зазвичай працюють з інтегральними величинами. У табличному вигляді доцільно формувати щонайменше: роботу циклу, середні тиски p_i, p_e, коефіцієнти корисної дії, питомі витрати, індикаторну та ефективну потужність, а також контрольні величини (наприклад, перевірка циклової подачі палива).

Окреме питання – порівняння запусків. Для цифрового двійника важливо мати сценарій «базовий режим → зміна параметра → порівняння». Найпростіший практичний варіант – зберігати результати кожного прогону з метаданими, після чого будувати накладені графіки $P(\varphi), x(\varphi)$ та порівняльні таблиці різниць по g_e, η_e, p_{max} тощо. Саме порівняння робить платформу інструментом дослідження, а не одноразовим калькулятором.

З технологічної точки зору візуалізацію доцільно організовувати у два рівні.

Локальний рівень.

На цьому етапі логічно використовувати Matplotlib для графіків і Pandas для звітних таблиць. Перевага в тому, що результат з'являється одразу після розрахунку, а налагодження можна вести в одному середовищі. Такий підхід добре підходить для перевірки підмоделей (впорскування, випаровування, інтегратор циклу), оскільки будь-яка помилка швидко "видна на графіку".

Веб-рівень.

Для платформи «цифровий двійник» природною є схема, коли ядро працює як сервіс: сервер приймає параметри, виконує розрахунок і повертає дані у структурованому вигляді, а клієнт (браузер) будує інтерактивні графіки. Важливо, що навіть при локальній реалізації варто одразу формувати результати так, ніби вони будуть передані через API: тобто зберігати масиви, серії, ідентифікатори запуску, одиниці, ключові події циклу. Це знімає проблему "переписувати" формат даних під веб на пізніших етапах.

Таким чином, інтерфейс і візуалізація в «DieselCycleUa» – це не допоміжний модуль "намалювати графік", а повноцінна підсистема, яка робить модель придатною для аналізу, пояснення та порівняння режимів.

1.7 Технології забезпечення якості: тестування та валідація

Якість інженерної інформаційної системи визначається двома речами: чи стабільно працює код і чи фізично коректний результат. Для цифрового двійника друге часто важливіше: програма може “не падати”, але давати нефізичні значення – і це буде критичною помилкою. Тому систему контролю якості потрібно будувати так, щоб вона перевіряла і програмну, і фізичну узгодженість.

Перший рівень – unit-тести для базових функцій. Тут перевіряються ті елементи, які є фундаментом усього пайплайна:

- розрахунок об’єму циліндра $V(\varphi)$ і похідної $dV/d\varphi$ (включно з перевірками крайніх положень поршня);
- перетворення одиниць і узгодження форматів (кПа ↔ Па, градуси ↔ радіани, кДж ↔ Дж);
- функції, що формують проміжні параметри (маса заряду, тиск/температура в точці впорскування, оцінка затримки займання), де помилка в одному місці далі “роз’їжджає” весь цикл.

Другий рівень – регресійні тести. Для платформи моделювання це практично обов’язково: береться еталонний набір параметрів (типовий режим), зберігаються ключові результати (наприклад, $p_{max}, T_{max}, p_i, g_e$, форма $P(\varphi)$ в контрольних точках), і при кожній зміні коду результати порівнюються з еталоном у заданому допуску. Це дозволяє розвивати модель.

Третій рівень – контроль фізичної узгодженості (інваріанти і обмеження). Це набір перевірок, які повинні спрацьовувати автоматично під час виконання моделі або після розрахунку:

- невід’ємність мас, тисків і температур у всіх кроках інтегрування;
- контроль допустимих діапазонів (наприклад, надто високі або надто низькі $T(\varphi)$ як маркер помилки в одиницях або балансі);
- перевірка відсутності NaN/inf у масивах профілів;
- перевірки “очікуваних” властивостей кривих там, де це фізично обґрунтовано (наприклад, безпідставні стрибки $x(\varphi)$, або некоректна поведінка $V(\varphi)$).

У підсумку, застосування технологій тестування та валідації для «DieselCycleUa» зводиться до простого принципу: код повинен бути перевірюваним, а результат – фізично контрольованим. Такий підхід робить цифровий двійник не разовою програмою для розрахунку, а керованою платформою, яку можна розвивати, інтегрувати з веб-API, доповнювати новими підмоделями та використовувати для серійних досліджень із накопиченням і порівнянням результатів.

2 Формування вимог до інформаційної системи, розробка її концепції і постановка задачі

2.1 Призначення інформаційної системи та межі проєкту

Інформаційна система, яка розробляється в межах магістерської роботи, призначена для створення програмної платформи класу «Цифровий двійник» дизельного двигуна на основі моделі повного робочого циклу. По суті, це інструмент, що дозволяє в одному середовищі задати вхідні параметри двигуна та умов роботи, виконати розрахунок циклу за обраним сценарієм, отримати як інтегральні показники, так і деталізовані профілі процесу, а також зберегти результат у відтворюваному форматі.

На рівні обчислювальної логіки система має охоплювати ключові етапи формування робочого циклу дизеля: наддув і підготовку заряду, процеси наповнення та стискання, опис згоряння (включно з фазовими характеристиками у кутовій області), подальше розширення, газообмін та теплообмін. Важливо, що цифровий двійник не обмежується розрахунком «характерних точок» – принциповою вимогою є формування масивів параметрів у функції кута повороту колінчастого вала, тобто отримання профілів $P(\varphi), T(\varphi), V(\varphi), x(\varphi)$ та похідних величин, які безпосередньо використовуються для побудови індикаторної діаграми та аналізу якості згоряння.

У межах проєкту магістерської роботи приймаються такі рамки (межі) реалізації:

- модель орієнтована на розрахунок повного робочого циклу з можливістю варіювання як режимних параметрів (частота обертання, коефіцієнт надлишку повітря, параметри наддуву, температура/тиск середовища), так і конструктивних (геометрія циліндра, ступінь стискання, параметри впорскування та ін.);
- підтримується сценарний підхід: користувач може виконувати серію запусків із різними наборами параметрів, щоб порівнювати отримані результати між режимами (наприклад, базовий режим проти зміненого кута випередження впорскування або зміни наддуву);
- вихідні дані формуються у форматі, придатному для подальшого аналізу й візуалізації, а також для інтеграції із зовнішніми компонентами. На рівні магістерської роботи це фіксується у вигляді концепції та технічної постановки задачі: структура даних і результати мають бути сумісними з майбутнім веб-інтерфейсом та web-API (тобто, ідеологія «модель як сервіс» закладається одразу).

Таким чином, межі проєкту визначають цифровий двійник як інженерно-обчислювальну інформаційну систему, де ключовими є відтворюваність запусків, прозорість даних та готовність до подальшого розширення у напрямку сервісної (веб) архітектури.

2.2 Зацікавлені сторони (stakeholders) та користувачі

Оскільки цифровий двійник у межах магістерської роботи розглядається як інформаційна система, важливо окремо визначити коло зацікавлених сторін і їхні потреби. Це дозволяє правильно сформувати вимоги: система має бути не лише «працюючою», а й зручною для навчальної, дослідницької та розробницької діяльності.

Основними користувачами та стейкхолдерами є:

Інженер-дослідник / студент.

Це ключовий користувач, для якого система є інструментом розрахунку та аналізу. Йому потрібна можливість швидко задавати параметри двигуна і режиму, запускати розрахунок, отримувати профілі процесу і інтегральні показники, будувати індикаторні діаграми, а також порівнювати різні сценарії. Для цього важливими є зрозуміла структура вхідних параметрів, наочний вихід (графіки + таблиці) і можливість збереження результатів, щоб повернутися до них пізніше.

Викладач / науковий керівник.

Для цієї групи основний акцент – не стільки на швидкості запуску, скільки на коректності постановки задачі, адекватності використаних моделей і відтворюваності результатів. Важливо, щоб система була достатньо прозорою: які саме параметри приймаються, які моделі застосовуються, як формується результат, чи можна повторити розрахунок за тим самим набором даних. У контексті магістерської роботи викладач також оцінює рівень системного проєктування: наскільки чітко визначені вимоги, межі проєкту, структура даних, підхід до тестування.

Розробник системи.

Це роль, яка відповідає за підтримку й розвиток коду. Розробнику важливі модульність і структурованість рішення, чіткі контракти між частинами системи, мінімізація дублювання, контроль одиниць, наявність тестів і можливість швидко локалізувати помилки. Для цифрового двійника дизеля ця роль особливо критична, оскільки система потенційно буде розширюватися (додавання нових підмоделей, альтернативних законів тепловиділення, інших палив, варіантів газообміну, режимів тощо), і без правильно сформованих вимог та архітектурних обмежень розвиток швидко стає некерованим.

Отже, визначення стейкхолдерів у даному проєкті прямо впливає на вимоги до системи: вона має одночасно бути корисною для користувача-практика, прозорою для контролю наукової коректності та зручною для розвитку як програмний продукт.

2.3 Концепція системи «Цифровий двійник» дизельного двигуна

У межах цієї магістерської роботи «цифровий двійник» дизельного двигуна я розглядаю не як окрему програму “порахувати цикл”, а як платформу, яка дозволяє відтворити робочий процес двигуна в розрахунковому вигляді, зберегти результат і дати можливість його нормально проаналізувати (а не просто вивести кілька чисел у консоль). Тобто система має поводити себе як керований інструмент: я задаю параметри двигуна та режиму, запускаю розрахунок за заданим сценарієм і отримую стандартизований набір вихідних даних, який можна порівнювати між запусками.

Концептуально систему доцільно поділити на чотири взаємопов’язані підсистеми.

1) Обчислювальне ядро (Core Engine).

Це центральна частина цифрового двійника, де виконується фізико-математичне моделювання повного робочого циклу. Саме тут реалізуються:

- геометрія циліндра та кінематика кривошипно-шатунного механізму (функція об’єму $V(\varphi)$ і її похідна);
- термодинамічний розрахунок для характерних точок циклу (параметри заряду, стискання, оцінка параметрів згоряння та розширення);
- підмодель впорскування та розпилу (визначення швидкостей, чисел подібності, характеристик факела, оцінка середнього діаметра крапель і констант випаровування);
- підмодель випаровування та підготовки палива до згоряння (формування частки випареного палива як функції кута);
- модель згоряння у кутовій області – формування закону тепловиділення та/або частки згорілого палива $x(\varphi)$, а також швидкості $dx/d\varphi$;
- теплообмін (оцінка коефіцієнтів тепловіддачі та теплопередачі, корекція процесу через втрати тепла);
- газообмін (впуск/випуск, масові потоки через клапани, баланс мас);
- чисельне інтегрування параметрів по куту φ для отримання масивів $P(\varphi), T(\varphi), M(\varphi)$ тощо.

Якщо пояснювати простіше: ядро робить “важку” фізику і математику та формує весь набір профілів параметрів по циклу, а не тільки пару значень в точках.

2) Підсистема керування даними (Data Layer).

Цифровий двійник без нормальної роботи з даними швидко перетворюється на “скрипт для одного запуску”. Тому потрібен окремий шар, який відповідає за:

- завантаження вхідних параметрів (геометрія, режим, паливо, емпіричні коефіцієнти);
- збереження результатів так, щоб їх можна було повторити (тобто «один запуск = один набір вхідних даних + один набір вихідних даних»);
- ведення історії запусків і можливість робити серійні експерименти.

Базові формати для цієї задачі логічні: JSON – для конфігурації та структурованих результатів; CSV – для профілів і таблиць, які зручно відкривати в Excel або аналізувати окремо. Якщо серій розрахунків стає багато, доцільно підключати SQLite як легку базу для збереження запусків і порівняння сценаріїв.

3) Підсистема інтерфейсу взаємодії (Interface Layer).

Навіть якщо ядро працює ідеально, користувачу потрібен зручний спосіб запустити розрахунок і отримати результат. На старті це може бути найпростіший варіант (CLI або невелике GUI), але для платформи «цифровий двійник» логічно одразу закласти ідею роботи через сервісний інтерфейс.

Тому на рівні концепції ця підсистема включає:

- механізм запуску розрахунку з передачею параметрів;
- механізм отримання результатів у стандартизованій структурі;
- підготовку логіки до роботи через web-API (тобто щоб “запит” на розрахунок і “відповідь” із результатами могли бути легко винесені у серверний режим).

4) Підсистема візуалізації та звітності (Visualization/Reporting).

В інженерній задачі недостатньо “вивести числа”: результат потрібно побачити і порівняти. Для цього потрібні:

- індикаторні діаграми та графіки $P(\varphi), V(\varphi), T(\varphi)$;
- графіки процесів згоряння та випаровування: $x(\varphi), dx/d$, інші допоміжні профілі;
- підсумкові таблиці інтегральних показників (робота циклу, p_i, p_e, ККД, питомі витрати, потужність);
- порівняльні графіки для кількох запусків (наприклад, базовий режим і режим зі зміщенням кута впорскування або зі зміною наддуву).

На ранньому етапі це може бути реалізовано локально (Matplotlib), але важливо одразу формувати результати так, щоб їх можна було віддати у веб-візуалізацію без “переписування всього з нуля”.

У підсумку, цифровий двійник у даному проєкті можна описати як програмний об’єкт/сервіс, який:

- приймає параметри “реального” двигуна та режиму роботи;
- відтворює робочий процес розрахунковим шляхом;
- повертає результати у стандартизованому вигляді для аналізу, порівняння та інтеграції.

2.4 Функціональні вимоги

Нижче наведені основні функціональні вимоги, які прямо випливають з концепції системи і потрібні, щоб платформа працювала як цифровий двійник, а не як «разовий калькулятор».

FR1. Завантаження вхідних даних.

Система повинна завантажувати параметри двигуна, режиму та палива з конфігураційних файлів (передусім JSON), а також підтримувати введення параметрів через інтерфейс (форма, CLI або GUI). Вхідні дані мають включати як основні параметри, так і коефіцієнти емпіричних моделей, які впливають на результат.

FR2. Розрахунок характерних параметрів циклу.

Система повинна виконувати розрахунок параметрів на основних етапах циклу (наддув, наповнення, стискання, згоряння, розширення) з формуванням ключових станів: P_c, T_c, P_z, T_z, а також інших проміжних величин, що потрібні для подальшого інтегрування і розрахунку показників.

FR3. Побудова профілів параметрів у функції кута φ.

Система повинна формувати масиви $P(\varphi), T(\varphi), V(\varphi)$ і пов’язані проміжні функції процесу: частку випареного/згорілого палива $x(\varphi)$, швидкість $dx/d\varphi$, а також допоміжні профілі (теплообмін, масові потоки при газообміні тощо). Саме цей пункт відрізняє «повний цикл» від спрощених методик “по точках”.

FR4. Обчислення інтегральних енергетичних показників.

Система повинна визначати індикаторну роботу циклу:

$$L_i = \oint P \ dV,$$

а також похідні інтегральні показники: середній індикаторний тиск p_i, середній ефективний тиск p_e, індикаторний та ефективний ККД, питомі витрати палива, індикаторну й ефективну потужність тощо.

FR5. Візуалізація результатів.

Система повинна будувати індикаторні діаграми, графіки профілів процесу та формувати підсумкові таблиці. Бажано, щоб набір графіків був достатнім для пояснення, “чому” змінився результат, а не лише “на скільки”.

FR6. Збереження і відтворення результатів.

Система повинна зберігати результати кожного запуску разом із вхідними параметрами, щоб забезпечити відтворюваність (можливість повторити розрахунок) і порівняння різних сценаріїв. Тобто результат має бути «прив’язаний» до конфігурації, а не існувати окремо.

FR7. Підготовка до інтеграції через API.

Система повинна мати визначені структури обміну: «вхідні параметри → вихідні показники». Навіть якщо web-API не реалізується повністю в першій версії, формат запиту/відповіді (логіка та структура даних) має бути закладений одразу, щоб потім не переробляти всю архітектуру.

2.5 Нефункціональні вимоги

Окрім функцій, які система повинна виконувати (завантажити дані, порахувати цикл, побудувати графіки тощо), для цифрового двійника важливо визначити нефункціональні вимоги. Вони описують не “що саме робить програма”, а “якою вона має бути”: наскільки результатам можна довіряти, чи можна повторити розрахунок, як система поводиться при розширенні, чи не перетворюється вона на складний у користуванні набір скриптів.

NFR1. Достовірність і фізична узгодженість результатів.

Оскільки мова йде про інженерне моделювання, система має “не дозволяти” отримати абсурдні результати без попередження. На практиці це означає, що під час розрахунку і формування вихідних даних потрібно контролювати фізичні обмеження: тиск, температура та маса не повинні ставати від’ємними; параметри мають залишатися у реалістичних межах; одиниці вимірювання повинні бути узгоджені. Якщо вхідні дані задані неправильно (наприклад, тиск в кПа замість Па, або температура в °C замість K), система має або явно виконати перетворення, або видати помилку, щоб уникнути “тихої” помилки, яку потім складно відловити.

NFR2. Відтворюваність.

Один і той самий набір вхідних даних має давати один і той самий результат за умови, що версія моделі та налаштування не змінювалися. Для цього важливо зберігати конфігурацію запуску разом із результатом і фіксувати версійність (принаймні версію коду/модуля та дату запуску). Це особливо актуально для серій розрахунків, коли потрібно через певний час повернутись до результатів і чесно повторити експеримент.

NFR3. Масштабованість і розширюваність.

Архітектура повинна дозволяти додавати нові можливості без “переписування всього проєкту”. Для цієї теми це виглядає реальною потребою, бо модель може розвиватися: з’являться інші підходи до опису згоряння, нові види палива/сумішей, додаткові метрики, або альтернативні методи валідації. Тому важливо, щоб модулі обчислювального ядра, збереження даних і візуалізації були достатньо ізольовані, а обмін між ними мав зрозумілу структуру.

NFR4. Зручність використання.

Система повинна мати логічний і простий сценарій роботи, який зрозумілий навіть без довгого “вчитування” у код. Базова послідовність виглядає так: підготовка параметрів → запуск розрахунку → перегляд графіків і таблиць → експорт результатів. Якщо користувачеві потрібно робити багато ручних кроків або “підправляти код під кожен запуск”, то це вже не платформа, а набір окремих заготовок.

NFR5. Продуктивність.

Час одного розрахунку має бути прийнятним для навчально-дослідних задач. Тобто модель повинна працювати в межах секунд–хвилин, а не десятків хвилин або годин, і не вимагати надмірних витрат оперативної пам’яті. Це важливо ще й тому, що у платформі передбачені серійні запуски: якщо один прогін повільний, то десятки варіантів стануть практично нереальними для аналізу.

2.6 Обмеження та припущення

Під час постановки задачі потрібно одразу зафіксувати, що саме обмежує точність і межі застосування цифрового двійника.

По-перше, модель реалізується на Python, а точність результатів залежить не тільки від правильності коду, а й від прийнятих емпіричних коефіцієнтів та спрощень. Це нормальна ситуація для інженерних моделей: навіть при правильній математичній реалізації результат може “плисти”, якщо коефіцієнти не підлаштовані під конкретний тип двигуна або режим.

По-друге, вхідні дані мають задаватися в узгоджених одиницях вимірювання, або з явними перетвореннями. На практиці це треба закріпити як правило: або система приймає тільки один стандарт (наприклад, SI), або вона робить перетворення через задані параметри конфігурації.

По-третє, на етапі магістерської роботи web-API може бути реалізоване частково або розглядатися як технічне рішення/концепція без промислового розгортання. Тобто головний акцент – працездатний прототип і правильна архітектурна основа, а не “готовий комерційний сервіс”.

2.7 Постановка задачі на розробку

Мета розробки полягає у створенні програмної платформи класу «Цифровий двійник» дизельного двигуна, яка за заданими параметрами двигуна, режиму та палива виконує моделювання повного робочого циклу й формує комплекс вихідних показників та візуалізацій для подальшого аналізу.

По суті, система повинна реалізувати замкнений цикл роботи: користувач задає параметри, отримує розрахунковий результат, має можливість його зберегти, відтворити і порівняти з іншими режимами.

Вхідні дані доцільно групувати, щоб користувачеві було зрозуміло, що саме він задає і за що це відповідає.

1. Геометричні параметри двигуна.

Сюди входять діаметр циліндра D, хід поршня S, ступінь стиску ε, а також параметри, які потрібні для кінематики КШМ (для розрахунку $V(\varphi)$ і похідних).

2. Режимні та зовнішні умови.

Частота обертання n, параметри навколишнього середовища P_0, T_0, вологість (за потреби), параметри наддуву, а також налаштування режиму, які впливають на наповнення та газообмін.

3. Параметри паливоподачі та форсунки.

Кут початку впорскування φ_{inj}, тривалість впорскування φ_{vp}, дані форсунки (геометрія отворів, кількість, витратні характеристики), а також параметри, які впливають на розпил і початкові умови паливного факела.

4. Характеристики палива.

Елементний склад (C, H, O, S), нижча теплота згоряння Q_n, густина, в’язкість та інші властивості, які потрібні для підмоделей випаровування/згоряння (за потреби можуть бути додані параметри типу енергії активації або спрощені коефіцієнти, якщо використовується емпіричний підхід).

5. Налаштування моделей (коефіцієнти).

Це набір емпіричних коефіцієнтів і перемикачів, які визначають, які підмоделі використовуються (наприклад, варіант теплообміну, закон тепловиділення тощо).

Вихідні дані система повинна формувати у двох рівнях: профілі процесу і підсумкові інтегральні показники.

1. Профілі у функції кута φ.

Головні результати повного циклу – це масиви:

$P(\varphi), T(\varphi), V(\varphi)$, а також функції процесу згоряння $x(\varphi)$ і $dx/d\varphi$. Саме вони дозволяють будувати індикаторні діаграми та аналізувати “форму” процесу, а не тільки підсумкове число.

2. Інтегральні показники та метрики.

Система повинна визначати індикаторну роботу L_i, середні тиски p_i та p_e, індикаторний і ефективний ККД (η_i, η_e), питомі витрати (g_i, g_e), а також індикаторну й ефективну потужність (N_i, N_e).

3. Візуальні та табличні матеріали.

Результатом мають бути графіки (індикаторна діаграма і допоміжні профілі), а також таблиці підсумкових значень. У ідеалі – з можливістю порівняння кількох запусків.

4. Експорт результатів.

Система повинна формувати файли експорту: JSON (структурований результат і конфігурація запуску) та CSV (профілі і табличні набори даних).

Очікуваний результат роботи – працездатний прототип, який забезпечує повний ланцюжок “параметри → розрахунок → збереження → візуалізація → аналіз” і має нормальні архітектурні передумови для подальшого підключення web-API та інтерактивної веб-візуалізації (навіть якщо ці компоненти в магістерській реалізації виконані не в промисловому масштабі).

3 Розробка проектних рішень по інформаційній системі

У цьому розділі подано проєктні рішення, які забезпечують реалізацію програмної платформи класу «Цифровий двійник» дизельного двигуна на базі моделі повного робочого циклу. Логіка проєктування тут така: модель має не просто “порахувати цикл”, а працювати як керована інформаційна система – з чіткими правилами підготовки параметрів, запуску, збереження результатів, повторення експериментів і подальшого аналізу. Тому всі рішення формуються з орієнтацією на відтворюваність (щоб один і той самий запуск можна було повторити), розширюваність (щоб додавати модулі без “перелому” структури) та зручність інтеграції з інтерфейсом, зокрема з веб-рівнем у перспективі.

3.1 Загальносистемна архітектура

Для платформи цифрового двійника доцільно одразу прийняти багаторівневу архітектуру, де обчислювальне ядро ізольоване від “обгортки” у вигляді інтерфейсу та сховища даних. Це важливо з практичної точки зору: ядро тоді не залежить від того, як саме запускається програма (через консоль, кнопку у GUI чи HTTP-запит), а також не “прив’язується” до конкретного формату зберігання. У результаті одна й та сама модель може використовуватися і як локальний інструмент для навчальних робіт, і як сервіс для більш складного сценарію (серійні розрахунки, робота через браузер, обмін через API).

У роботі прийнято поділ на чотири основні підсистеми.

1) Core – обчислювальне ядро.

Це центральний модуль, де реалізовано модель повного робочого циклу. Його задача – отримати підготовлений набір параметрів і повернути результат у стандартній формі. В ядро входять функції геометрії та кінематики КШМ (розрахунок $V(\varphi)$ і пов’язаних залежностей), термодинамічні розрахунки основних процесів, блоки, що описують впорскування, розпил та випаровування палива, модель згоряння, газообмін і теплообмін, а також інтегратор циклу, який формує профілі $P(\varphi)$ і $T(\varphi)$ та підсумкові метрики.

Ключова вимога до Core – не знати “нічого зайвого” про те, де будуть показані графіки або як саме зберігатимуться результати. Ядро має працювати як чиста функція: параметри → розрахунок → результат.

2) Data – інформаційний рівень (керування даними).

Цей рівень відповідає за збереження і відтворення запусків. Він реалізує читання параметрів, запис результатів, експорт профілів, а також ведення структури “історії запусків”, якщо робиться серія розрахунків. Для прототипу достатньо JSON/CSV, але в архітектурі передбачається можливість підключення

локальної бази (наприклад, SQLite), якщо потрібно накопичувати десятки/сотні запусків, робити пошук і порівняння по параметрах.

3) Interface – рівень взаємодії.

Це частина, яка організує сценарій роботи: завантаження конфігурації, виклик ядра, отримання результату, передача його на збереження і візуалізацію. Важливий момент: інтерфейс не повинен містити фізики чи "важких" формул – він лише керує процесом і валідністю вводу. На початковому етапі це може бути CLI (консольний запуск) або простий GUI. У перспективі той самий рівень стає основою для web-API, де замість локального виклику використовується HTTP-запит, але внутрішня логіка запуску залишається однакова.

4) Visualization/Reporting – візуалізація та звітність.

Платформа цифрового двійника має пояснювати результат, а не тільки видавати числові значення. Тому окремим модулем виділяється побудова графіків і підготовка підсумкових таблиць: індикаторна діаграма, профілі $P(\varphi), T(\varphi), V(\varphi)$, а також порівняння кількох запусків. На рівні локальної реалізації це може бути Matplotlib і табличний вивід у файл, але дані формуються так, щоб їх можна було в майбутньому показувати й у веб-інтерфейсі.

У підсумку архітектура будується за принципом "ядро незалежне – обгортки змінні". Це дозволяє розвивати систему без конфліктів: наприклад, додати новий тип графіків, не торкаючись термодинаміки, або підключити web-API, не переписуючи моделі згоряння.

3.1.1 Вибір засобів моделювання для ІС «Цифровий двійник» дизельного двигуна

Під час розробки інформаційної системи типу «цифровий двійник» дизельного двигуна ключовим етапом є обґрунтований вибір засобів моделювання та програмної реалізації. На відміну від окремого розрахункового алгоритму, цифровий двійник повинен поєднувати фізичну модель робочого процесу, програмні засоби її реалізації та елементи системного проєктування, які забезпечують керованість, відтворюваність і можливість подальшого розвитку платформи. З цієї причини при виборі засобів доцільно розглядати їх у трьох взаємопов'язаних групах: засоби фізико-математичного моделювання, засоби інженерного програмування та засоби системного проєктування інформаційної системи.

1. Засоби моделювання робочого циклу (0D/1D підхід)

Основою цифрового двійника є модель повного робочого циклу дизельного двигуна. У межах даної роботи як базовий приймається 0D/1D-підхід із дискретизацією процесів за кутом повороту колінчастого вала (φ) у діапазоні 0–

720°. Такий підхід є класичним для інженерних теплотехнічних розрахунків і широко застосовується у навчальній та проєктній практиці.

Моделювання по куту колінчастого вала дозволяє відтворити послідовність основних підпроцесів робочого циклу: газообмін, стискання, згоряння та розширення. При цьому всі величини (тиск, температура, об'єм, тепловиділення) розглядаються як функції (φ), що дає можливість аналізувати не лише кінцеві результати, а й форму перебігу процесу. Саме така інформація необхідна для побудови індикаторних діаграм, оцінки фаз згоряння та впливу окремих параметрів на ефективність циклу.

Обраний підхід забезпечує розумний компроміс між точністю і швидкодією. На відміну від CFD-моделей, 0D/1D-модель не потребує значних обчислювальних ресурсів і може бути багаторазово використана для серійних розрахунків, що особливо важливо в умовах навчального або дослідного експерименту. Водночас така модель зберігає фізичний зміст і дозволяє коректно оцінювати інтегральні показники циклу, такі як індикаторна робота, середні тиски, коефіцієнти корисної дії та питомі витрати палива.

2. Засоби інженерного програмування

Для програмної реалізації обчислювального ядра цифрового двійника обґрунтовано використання мови програмування Python. Вибір саме цієї екосистеми зумовлений її широким застосуванням у задачах інженерних розрахунків, зручністю роботи з числовими масивами та можливістю швидкого створення прототипів.

У межах проєкту Python використовується як універсальна R&D-платформа, що поєднує чисельні обчислення, обробку даних та візуалізацію результатів. Масиви параметрів і профілів по куту (φ) зручно реалізовувати у вигляді числових структур, що дозволяє виконувати інтегрування, диференціювання та постобробку результатів без ускладнення алгоритму. Побудова графіків і індикаторних діаграм дає змогу одразу інтерпретувати результати розрахунку в наочній формі.

Для збереження та обміну даними в проєкті використовуються формати JSON і CSV. Формат JSON застосовується для зберігання вхідних параметрів і підсумкових результатів у структурованому вигляді, що полегшує повторення запусків і контроль параметрів. Формат CSV доцільний для збереження профілів і подальшого аналізу в табличних редакторах або інших програмних засобах. Для накопичення серій розрахунків і збереження історії експериментів використовується локальна база даних SQLite, яка не потребує окремого сервера і добре підходить для навчальних і дослідних проєктів невеликого масштабу.

3. Засоби системного проєктування

Оскільки розроблювана платформа розглядається саме як інформаційна система, а не як окремий скрипт для розрахунку, важливу роль відіграють засоби системного проєктування. У роботі застосовується стандартний набір UML-моделей, які дозволяють формалізувати структуру та поведінку системи.

Діаграма варіантів використання (Use Case) дає уявлення про основні сценарії взаємодії користувача з цифровим двійником: підготовку параметрів, запуск розрахунку, збереження та аналіз результатів. Концептуальна та логічна моделі даних описують склад і взаємозв'язок інформаційних об'єктів, таких як параметри двигуна, режими роботи, паливо та результати розрахунків. Модель класів дозволяє структуризувати програмні компоненти та їх відповідальності, а моделі взаємодії (Sequence, State, Activity) формалізують послідовність виконання операцій під час роботи системи.

Використання таких засобів системного проєктування дозволяє представити цифровий двійник як цілісну інформаційну систему з чіткою логікою роботи та можливістю подальшого розвитку, наприклад, у напрямку інтеграції з web-інтерфейсом або розширення набору фізичних моделей.

3.1.2 Розробка діаграми варіантів використання ІС «Цифровий двійник» дизельного двигуна

Для формалізації взаємодії користувачів із програмною платформою «Цифровий двійник» дизельного двигуна на етапі проєктування доцільно використати діаграму варіантів використання (Use Case). Така діаграма дозволяє наочно описати, які саме дії може виконувати система, хто є ініціатором цих дій і в яких межах відбувається взаємодія, не заглиблюючись при цьому в деталі реалізації алгоритмів. Це особливо важливо для навчального проєкту, де потрібно показати, що модель розглядається як повноцінна інформаційна система, а не як окремий обчислювальний модуль.

Актори інформаційної системи

У межах розроблюваної ІС виділено три основні актори, які відображають реальні ролі користувачів і сценарії експлуатації системи.

Основним актором є користувач-дослідник. Саме він безпосередньо працює з цифровим двійником: задає параметри двигуна і режиму роботи, обирає паливо, запускає розрахунки та аналізує отримані результати. Цей актор відповідає типовому користувачу системи в навчальному або дослідному процесі – студентові, аспіранту або інженеру, який проводить серію розрахункових експериментів і порівнює різні варіанти налаштувань.

Другим актором є розробник або адміністратор системи. Його роль полягає не у виконанні розрахунків, а в підтримці працездатності та актуальності

платформи. Зокрема, цей актор відповідає за ведення довідника палив, актуалізацію або коригування еталонних конфігурацій, а також контроль версій обчислювального ядра. Виділення такого актора в діаграмі варіантів використання дозволяє чітко відмежувати функції користувача і функції супроводу системи.

Третім актором, який закладається на перспективу розвитку платформи, є зовнішній клієнт. Під цим актором розуміється web-клієнт або програмний скрипт, що взаємодіє з цифровим двійником через програмний інтерфейс (API). У поточній реалізації ця взаємодія може бути відсутня або обмежена, однак її включення до діаграми варіантів використання демонструє архітектурну готовність системи до масштабування та інтеграції у клієнт–серверне середовище.

Ключові варіанти використання

На основі аналізу функціональності платформи та реальних сценаріїв її використання сформовано набір основних варіантів використання, які відображають повний життєвий цикл розрахункового експерименту.

Першим і базовим варіантом використання є створення або редагування конфігурації запуску. У межах цього сценарію користувач формує набір вхідних параметрів, що описують геометрію двигуна, режим роботи, властивості палива та налаштування фізичних підмоделей. Саме на цьому етапі закладається відтворюваність експерименту, оскільки конфігурація зберігається у структурованому вигляді та може бути використана повторно.

Другий варіант використання пов’язаний із вибором палива з довідника або додаванням нового палива. Наявність окремого довідника дозволяє відокремити властивості палива від коду обчислювального ядра, що спрощує дослідження впливу альтернативних палив або їх домішок без зміни структури моделі.

Центральним варіантом використання є виконання одиночного розрахунку повного робочого циклу. У цьому сценарії система, отримавши підготовлену конфігурацію, виконує розрахунок усіх підпроцесів циклу та формує як інтегральні показники, так і профілі параметрів у функції кута колінчастого вала. Саме цей варіант використання реалізує основне призначення цифрового двійника.

Для проведення досліджень і аналізу чутливості передбачено варіант використання серійного розрахунку, або параметричного прогона (sweep). У межах цього сценарію система автоматично виконує кілька розрахунків зі зміною одного або декількох параметрів, що дозволяє досліджувати вплив окремих факторів на показники циклу без ручного втручання в кожен запуск.

Наступний варіант використання стосується перегляду та аналізу результатів розрахунку. Користувач отримує доступ не лише до числових значень інтегральних показників, але й до графічних залежностей ($P(\varphi)$), ($T(\varphi)$), а також індикаторної діаграми, що є основним інструментом аналізу робочого процесу двигуна.

Важливим елементом функціональності системи є збереження результатів і їх експорт. У межах цього варіанту використання результати розрахунку фіксуються у вигляді окремого запуску (run), який містить як вхідні параметри, так і вихідні дані. За потреби результати можуть експортуватися у форматах CSV або JSON для подальшого аналізу в зовнішніх інструментах.

Окремий варіант використання пов’язаний із порівнянням двох або більше запусків. Цей сценарій дозволяє накладати профілі параметрів, порівнювати індикаторні діаграми та формувати таблиці відмінностей інтегральних показників, що є особливо корисним при дослідженні впливу зміни одного параметра або типу палива.

Завершальним варіантом використання є запуск перевірок коректності та регресійних тестів. Цей сценарій орієнтований насамперед на розробника або адміністратора системи та дозволяє контролювати стабільність результатів при зміні коду або налаштувань моделі. Його включення до діаграми варіантів використання підкреслює, що цифровий двійник розглядається як система, яка потребує контролю якості та супроводу.

Таким чином, діаграма варіантів використання охоплює всі основні сценарії роботи з цифровим двійником – від підготовки параметрів до аналізу й збереження результатів – і забезпечує зрозумілу основу для подальшого проєктування структури програмних компонентів та моделей взаємодії.

3.1.3 Специфікації варіантів використання ІС «Цифровий двійник» дизельного двигуна

Після визначення акторів та загального переліку варіантів використання доцільно деталізувати ключові сценарії у вигляді специфікацій. Специфікація варіанта використання дозволяє чітко зафіксувати, які дані надходять на вхід системи, за яких умов виконується сценарій, яку послідовність дій реалізує інформаційна система та який результат отримує користувач. Такий підхід полегшує як подальшу реалізацію програмної частини, так і перевірку коректності роботи системи.

У межах даної роботи розглянуто три основні варіанти використання, які охоплюють типовий цикл роботи з цифровим двійником дизельного двигуна.

UC-1. «Одиночний розрахунок повного робочого циклу»

Даний варіант використання є базовим і реалізує основну функцію цифрового двійника – моделювання повного робочого циклу дизельного двигуна для одного набору параметрів.

Актор.

Ініціатором сценарію є користувач-дослідник, який виконує розрахунок з метою аналізу робочого процесу або оцінки впливу окремих параметрів.

Вхідні дані.

На вхід системи подається конфігурація запуску, що включає параметри двигуна, режим роботи, обране паливо та налаштування фізичних підмоделей. Окремо задається крок дискретизації по куту колінчастого вала ($\Delta\varphi$), який визначає точність і трудомісткість розрахунку.

Передумови виконання.

Перед запуском сценарію передбачається, що всі параметри є валідними з точки зору фізичного змісту та одиниць вимірювання, обране паливо існує у довіднику, а режим роботи характеризується фізично припустимими значеннями початкового тиску (P_0), температури (T_0) та частоти обертання (n).

Основний потік подій.

Сценарій починається із завантаження конфігурації запуску та її перевірки на коректність. Після успішної валідації система викликає обчислювальне ядро, яке виконує чисельне моделювання робочого циклу. У процесі розрахунку формуються профілі основних параметрів у функції кута (φ). Далі на основі цих профілів обчислюються інтегральні показники циклу. Отримані результати передаються на візуалізацію у вигляді графіків і таблиць, після чого зберігаються як окремий запуск (*run*) разом із вхідними даними.

Вихідні дані.

Результатом виконання сценарію є набір інтегральних метрик (середні тиски, ККД, потужність, питомі витрати), профілі параметрів по (φ), а також метадані запуску, зокрема час виконання, версія моделі та використаний крок дискретизації.

Виключні ситуації.

Сценарій може бути перерваний у разі виявлення некоректних одиниць або значень параметрів, порушення фізичних обмежень (наприклад, вихід функцій ($x(\varphi)$) або ($\sigma(\varphi)$) за межі [0;1]), а також при нестійкій роботі чисельного інтегратора.

UC-2. «Серія розрахунків (параметричний прогін)»

Цей варіант використання орієнтований на проведення досліджень і аналізу чутливості параметрів робочого циклу.

Актор.

Ініціатором сценарію також є користувач-дослідник.

Вхідні дані.

На вхід подається базова конфігурація запуску та перелік значень одного або кількох параметрів, що змінюються. Це можуть бути кут початку впорскування (φ_{inj}), тривалість впорскування, крок ($\Delta\varphi$) або коефіцієнти теплообміну.

Основний потік подій.

Система на основі заданих діапазонів автоматично формує набір конфігурацій. Для кожної з них послідовно виконується сценарій UC-1 з повним циклом розрахунку та збереженням результатів. Після завершення серії розрахунків результати агрегуються, а користувачу надається можливість перегляду порівняльних графіків і таблиць.

Вихідні дані.

Результатом є набір окремих запусків (run), а також зведений підсумок у вигляді таблиці чутливості або графіків трендів, що відображають вплив змінюваного параметра на основні показники циклу.

UC-3. «Порівняння запусків»

Даний варіант використання призначений для аналітичної обробки вже виконаних розрахунків.

Актор.

Ініціатором сценарію є користувач-дослідник.

Вхідні дані.

На вхід подається два або більше збережених запусків, виконаних раніше та доступних у сховищі результатів.

Основний потік подій.

Користувач обирає необхідні запуски, після чого система виконує нормалізацію осі кута (φ) та, за потреби, кроку дискретизації. Далі здійснюється накладання профілів параметрів, формуються порівняльні графіки та обчислюються відмінності інтегральних показників.

Вихідні дані.

Результатом сценарію є набір порівняльних графіків та таблиць відмінностей, які дозволяють зробити висновки щодо впливу зміни параметрів або режимів роботи на показники робочого циклу.

Таким чином, наведені специфікації варіантів використання описують основні сценарії роботи з цифровим двійником дизельного двигуна та формують логічний міст між загальною діаграмою варіантів використання і подальшим проєктуванням алгоритмічного ядра та моделей взаємодії.

3.1.4 Сценарії основних варіантів використання ІС «Цифровий двійник» дизельного двигуна

Для більш детального опису роботи інформаційної системи доцільно розглянути сценарії виконання основних варіантів використання. На відміну від формальних специфікацій, сценарії дозволяють описати реальний перебіг роботи системи крок за кроком, з урахуванням як штатних, так і нештатних ситуацій. Це дає уявлення про логіку керування процесом розрахунку та про механізми контролю коректності результатів.

У межах проєкту розглянуто три характерні сценарії: успішний локальний запуск, обробка помилки введення параметрів та реакція системи на нестійку роботу чисельної схеми.

Сценарій S-1. Успішний локальний запуск розрахунку

Даний сценарій відповідає типовому випадку використання цифрового двійника в навчальній або дослідній практиці. Користувач на початковому етапі формує файл *parameters.json*, у якому задає параметри двигуна, режим роботи, обране паливо та налаштування моделей. Такий підхід дозволяє чітко відокремити підготовку вхідних даних від самого процесу розрахунку.

Після запуску розрахунку система виконує валідацію введених параметрів. На цьому етапі перевіряються допустимі межі значень, правильність одиниць вимірювання та логічна узгодженість параметрів між собою. Якщо помилок не виявлено, керуючий модуль (Controller) передає підготовлені дані до обчислювального ядра (Core).

Обчислювальне ядро виконує чисельне моделювання робочого циклу та формує профілі основних параметрів ($P(\varphi)$), ($T(\varphi)$), а також службові масиви, необхідні для подальшої обробки. На основі отриманих профілів модуль metrics обчислює інтегральні показники циклу, зокрема середні тиски, коефіцієнти корисної дії, потужність та питомі витрати.

Далі модуль візуалізації (*viz*) будує стандартний набір графіків і таблиць, які використовуються для аналізу результатів. Завершальним кроком є збереження конфігурації та результатів розрахунку на рівні даних (Data-рівень) у вигляді одного запуску (*run*). Це забезпечує відтворюваність експерименту та можливість його подальшого порівняння з іншими режимами.

Сценарій S-2. Помилка введення параметрів

Другий сценарій описує ситуацію, коли на етапі підготовки або зчитування вхідних даних задано некоректні параметри. Прикладами таких помилок можуть бути значення ступеня стиску ($\varepsilon \leq 1$), від’ємний або нульовий початковий тиск ($P_0 \leq 0$), а також некоректні одиниці вимірювання.

У цьому випадку система не переходить до виконання розрахунку. Помилка фіксується на етапі обробки конфігурації (рівень io_config), після чого користувачу повертається повідомлення з поясненням, який саме параметр є некоректним, який діапазон допустимих значень очікується та в яких одиницях він має бути заданий. Такий підхід дозволяє уникнути виконання фізично безглуздих розрахунків і спрощує виправлення помилок користувачем.

Сценарій S-3. Нестійкість чисельного інтегрування

Третій сценарій стосується випадків, коли вхідні параметри формально є коректними, однак у процесі чисельного інтегрування виникають нефізичні значення. Це може проявлятися у вигляді від'ємних температур, виходу частки згорілого палива ($x(\varphi)$) за межі інтервалу [0;1] або різких стрибків параметрів, що свідчать про нестійкість чисельної схеми.

У такій ситуації система фіксує помилку вже під час розрахунку, зупиняє поточний запуск та позначає відповідний run як "failed". Користувачу надається рекомендація зменшити крок дискретизації ($\Delta\varphi$) або переглянути значення коефіцієнтів підмоделей, які можуть впливати на стійкість інтегрування. При цьому результати нестабільного запуску не включаються до серійних вибірок і не змішуються з коректними результатами, що підвищує надійність подальшого аналізу.

Розглянуті сценарії демонструють, що інформаційна система «Цифровий двійник» дизельного двигуна керує процесом розрахунку не лише в штатному режимі, але й у разі помилок введення або чисельних проблем. Це дозволяє забезпечити контроль коректності результатів і формує основу для подальшого розвитку платформи як інструменту навчальних та дослідних розрахунків.

3.2 Проєктування даних та форматів обміну

У платформі прийнято логіку роботи "конфігурація → розрахунок → результат". Такий підхід добре підходить саме для навчально-дослідних задач: конфігурація фіксує, що саме було задано, а результат зберігається разом із цими даними, тому запуск можна повторити або порівняти з іншими режимами.

Вхідні дані зберігаються у конфігураційному файлі (наприклад, parameters.json). Файл доцільно будувати так, щоб його можна було читати не тільки програмі, а й людині: структура розділена на логічні блоки, а назви параметрів не двозначні. Типовий набір блоків:

- параметри двигуна (геометрія, кількість циліндрів, \varepsilon, кінематичні параметри для розрахунку об'єму);

- режимні параметри (частота n, P_0, T_0, наддувні параметри, коефіцієнти наповнення чи пов'язані величини);
- параметри паливоподачі (кут початку впорскування, тривалість, характеристики форсунки);
- налаштування моделей (коефіцієнти політропи, параметри теплообміну, коефіцієнти повноти діаграми, механічний ККД тощо).

Окремо доцільно винести дані палива у "довідник" (наприклад, fuel_db.json або модуль fuel_data.py). Це дає дві переваги: по-перше, різні проєкти/запуски можуть використовувати один і той самий набір еталонних палив; по-друге, додавання нового палива зводиться до додавання одного запису, а не зміни коду ядра. Важливо одразу зафіксувати єдину структуру опису палива: елементний склад (C, H, O, S), Q_n, густина, в'язкість, а також параметри, які потребують підмоделі (якщо вони використовуються).

Вихідні дані логічно поділяються на дві групи.

1. Інтегральні показники циклу.

Це підсумкові значення: p_i, p_e, ККД, питомі витрати, потужності та інші метрики. Для них зручно застосувати структуру JSON, де крім числа зберігається одиниця вимірювання і короткий опис. Такий формат практичний, бо зменшує ризик плутанини одиниць, а також полегшує вивід у звіт.

2. Профілі по куту \varphi.

Сюди входять масиви $P(\varphi), T(\varphi), V(\varphi), x(\varphi), dx/d$. Їх зручно зберігати як дані з однаковим кроком по φ: або у JSON (як масиви), або у CSV (стовпцями). CSV практичний для швидкого перегляду в табличних редакторах та для імпорту в інші інструменти, а JSON – для передачі через API та для збереження разом із метаданими (крок, діапазон, одиниці).

Окремо варто зафіксувати правило, яке прямо впливає на відтворюваність: один запуск моделі = один набір вхідних параметрів + один набір результатів, і обидва ці набори повинні зберігатися разом (наприклад, у папці запуску з унікальним ідентифікатором або міткою часу). Тоді серія експериментів не перетворюється на "хаос файлів", а має структуру, з якою реально працювати.

3.2.1 Концептуальна модель даних ІС «Цифровий двійник» дизельного двигуна

Концептуальна модель даних інформаційної системи «Цифровий двійник» дизельного двигуна визначає набір основних сутностей, з якими працює система, а також їхні логічні взаємозв'язки. Її призначення полягає не у відображенні конкретної реалізації збереження даних, а у формуванні узгодженого уявлення про структуру інформації, що використовується під час моделювання робочого

процесу двигуна. Саме на цьому рівні закладається логіка відтворюваності розрахунків, можливості порівняння режимів та накопичення результатів.

Вихідним положенням при побудові концептуальної моделі є те, що цифровий двійник повинен відображати фізичну реальність двигуна і процесів у ньому, але у формі інформаційних об’єктів, зручних для обробки, збереження і аналізу.

Однією з базових сутностей є Engine. Вона описує конструктивні та геометричні характеристики дизельного двигуна, які в межах одного об’єкта моделювання вважаються незмінними. До цієї сутності належать основні паспортні параметри, геометрія циліндро-поршневої групи, співвідношення радіуса кривошипа і довжини шатуна, кількість циліндрів, а також ступінь стиску (ε). Окреме місце займають параметри, необхідні для визначення миттєвого об’єму циліндра ($V(\varphi)$) та його похідних за кутом колінчастого вала. Таким чином, сутність Engine формує геометричну і кінематичну основу для всіх подальших розрахунків і може повторно використовуватися у великій кількості запусків.

Сутність OperatingMode відображає умови, в яких експлуатується двигун у конкретному розрахунку. Вона включає частоту обертання колінчастого вала (n), початкові параметри навколишнього середовища (тиск (P_0) та температуру (T_0)), параметри наддуву, а також коефіцієнти, що характеризують наповнення циліндра і процес газообміну. На відміну від геометрії двигуна, режим роботи є змінною складовою, і саме його варіювання дозволяє досліджувати поведінку двигуна при різних навантаженнях, швидкісних режимах або умовах зовнішнього середовища.

Окремою сутністю виділено Fuel, яка описує паливо, що використовується у розрахунку. До цієї сутності входять фізико-хімічні властивості палива: елементний склад, нижча теплота згоряння (Q_n), густина, в’язкість та інші характеристики, необхідні для моделювання процесу згоряння. Винесення палива в окрему сутність та зберігання його у вигляді довідника має принципове значення. Це дозволяє легко змінювати тип палива або додавати нові види (наприклад, альтернативні або сумішеві палива) без втручання в структуру обчислювального ядра і без дублювання даних у кожному запуску.

Сутність ModelSettings об’єднує параметри та коефіцієнти, що визначають конкретну реалізацію фізичних підмоделей у цифровому двійнику. До неї належать показники політропи стискання і розширення, коефіцієнти теплообміну, коригувальні множники, коефіцієнт механічних втрат та інші налаштування, які не є прямими фізичними параметрами двигуна, але суттєво впливають на результати розрахунку. Виділення цієї сутності дозволяє чітко розмежувати

фізичну модель як таку і її параметризацію, що є важливим при аналізі чутливості та порівнянні різних підходів до моделювання.

Центральним елементом концептуальної моделі даних є сутність Run. Вона представляє собою окремий розрахунковий експеримент і є ключовою для забезпечення відтворюваності результатів. Один об'єкт Run об'єднує в собі всі вхідні дані (посилання на Engine, OperatingMode, Fuel та ModelSettings), результати розрахунку, а також службові метадані. До метаданих належать дата і час виконання, використаний крок дискретизації ($\Delta\varphi$), версія обчислювального ядра та статус виконання. Саме через сутність Run система дозволяє повторити розрахунок у майбутньому або порівняти його з іншими запусками на повністю однакових умовах.

Результати кожного запуску логічно поділяються на дві групи, що відображено у двох окремих сутностях. Сутність Profiles містить профільні дані у функції кута колінчастого вала (φ). До неї входять масиви значень тиску ($P(\varphi)$), температури ($T(\varphi)$), об'єму ($V(\varphi)$), частки згорілого палива ($x(\varphi)$), допоміжної функції ($\sigma(\varphi)$) та похідної ($\mathrm{d}x/\mathrm{d}$). Ці дані є найбільш інформативними з точки зору аналізу робочого процесу, оскільки дозволяють оцінити форму перебігу згоряння, положення максимумів параметрів та характер фаз циклу.

Сутність Metrics, у свою чергу, містить інтегральні результати розрахунку. До неї належать індикаторна робота (L_i), середні індикаторний і ефективний тиски (p_i) та (p_e), коефіцієнти корисної дії, питома витрата палива, ефективна потужність та інші узагальнені показники. Ці дані використовуються для швидкого порівняння різних режимів і варіантів розрахунків, а також для формування підсумкових таблиць і висновків.

Взаємозв'язки між сутностями у концептуальній моделі мають чітко визначену логіку. Один об'єкт Engine може бути пов'язаний із багатьма об'єктами Run, оскільки для одного й того самого двигуна можливе моделювання різних режимів роботи. Кожен об'єкт Run використовує одне конкретне паливо (*Fuel*) та один режим роботи (*OperatingMode*), а також один набір налаштувань моделей (*ModelSettings*). Водночас кожен Run має рівно один набір профільних даних (Profiles) і один набір інтегральних показників (*Metrics*), що забезпечує цілісність і несуперечливість результатів.

Таким чином, концептуальна модель даних відображає логіку роботи цифрового двійника дизельного двигуна як інформаційної системи, орієнтованої на відтворювані розрахункові експерименти. Вона створює основу для подальшого переходу до логічної та фізичної моделей даних і забезпечує структуроване представлення результатів моделювання.

3.2.2 Логічна модель даних ІС «Цифровий двійник» дизельного двигуна

Логічна модель даних є наступним рівнем деталізації після концептуальної моделі та визначає, як саме абстрактні сутності та їхні взаємозв’язки представлені у структурованому вигляді, зручному для збереження, обробки та передачі між компонентами інформаційної системи. На цьому рівні ще не фіксується конкретна реалізація у вигляді файлів або таблиць бази даних, однак уже визначається структура даних, склад атрибутів і принципи їх організації.

Ключовим принципом побудови логічної моделі для цифрового двійника дизельного двигуна є підхід «один розрахунковий запуск = одна логічна одиниця даних». Саме тому центральним елементом логічної моделі залишається сутність Run, навколо якої групуються всі інші дані.

У логічній моделі сутність Engine представляється як структурований набір параметрів, що описують геометрію та кінематику двигуна. До неї входять ідентифікатор двигуна, базові паспортні параметри, геометричні розміри циліндра, характеристики кривошипно-шатунного механізму, кількість циліндрів і ступінь стиску. Логічно Engine розглядається як відносно незалежний об’єкт, на який можуть посилатися кілька запусків Run, що дозволяє уникнути дублювання однакових геометричних даних.

Сутність OperatingMode у логічній моделі оформлюється як окрема структура, що містить набір режимних параметрів. До неї входять частота обертання колінчастого вала, початкові значення тиску і температури, параметри наддуву та коефіцієнти, що характеризують газообмін. У логічній моделі OperatingMode може зберігатися як частина конфігурації запуску або як окремий об’єкт із власним ідентифікатором, якщо один і той самий режим використовується у кількох серіях розрахунків.

Сутність Fuel у логічній моделі реалізується у вигляді запису довідника палив. Кожен запис має унікальне ім’я або код палива та набір атрибутів, що описують його фізико-хімічні властивості. Логічно Fuel не входить безпосередньо до складу Run, а зв’язується з ним через посилання, що забезпечує централізоване управління властивостями палива та їх узгодженість у різних розрахунках.

Сутність ModelSettings у логічній моделі описується як структурований блок налаштувань, що включає параметри політроп, коефіцієнти теплообміну, коригувальні множники та інші налаштування фізичних підмоделей. Ця сутність може зберігатися як частина конфігурації запуску або як окремий шаблон налаштувань, який використовується у кількох розрахунках. Такий підхід спрощує порівняння результатів при зміні лише одного параметра або підмоделі.

Центральна сутність Run у логічній моделі містить посилання на Engine, OperatingMode, Fuel та ModelSettings, а також власні атрибути. До них належать

ідентифікатор запуску, дата і час виконання, крок дискретизації по куту колінчастого вала ($\Delta\varphi$), версія обчислювального ядра та статус виконання (успішний або помилковий). Таким чином, Run виступає контейнером, що логічно об'єднує всі дані, пов'язані з одним розрахунковим експериментом.

Результати розрахунку у логічній моделі поділяються на дві взаємопов'язані структури. Сутність Profiles представляється як набір масивів значень, індексованих по куту (φ). Логічно всі профілі мають спільну вісь дискретизації та зберігаються як узгоджений набір даних, що дозволяє коректно будувати графіки та індикаторні діаграми. До складу Profiles входять масиви тиску, температури, об'єму, частки згорілого палива та допоміжних функцій.

Сутність Metrics у логічній моделі містить агреговані числові значення результатів розрахунку. Кожен показник зберігається разом з одиницями вимірювання та, за потреби, коротким описом. Логічно Metrics жорстко пов'язана з конкретним Run і не має сенсу поза контекстом відповідного запуску.

Взаємозв'язки між сутностями у логічній моделі зберігають кардинальності, визначені на концептуальному рівні. Один Engine може бути пов'язаний з багатьма Run, тоді як кожен Run має рівно один набір Profiles і один набір Metrics. Fuel та OperatingMode також мають зв'язок типу «один до багатьох» із Run, що відповідає реальній практиці проведення серій розрахунків.

Таким чином, логічна модель даних формує чітке уявлення про структуру інформації в ІС «Цифровий двійник» дизельного двигуна та слугує основою для реалізації фізичної моделі даних у вигляді файлів конфігурацій, результатів і, за потреби, локальної бази даних.

3.3 Проєктування програмного ядра (алгоритмічний конвеєр)

Розрахунок повного циклу організовується у вигляді послідовного конвеєра:

1. Завантаження параметрів (двигун/режим/паливо).
2. Початковий термодинамічний розрахунок: наддув, наповнення, стискання, визначення T_a, P_c, T_c, коефіцієнтів (наприклад, γ_r, η_H, n_1).
3. Розрахунок горіння: визначення параметрів суміші, коефіцієнтів молекулярної зміни, оцінка P_z, T_z та параметрів розширення.
4. Моделі впорскування/розпилу/випаровування: розрахунок затримки займання, параметрів факела, функцій $\sigma(\varphi), x(\varphi), dx/d\varphi$.
5. Чисельна інтеграція циклу по φ у межах 0–720° з урахуванням фаз горіння та газообміну.
6. Постобробка: індикаторна робота, середні тиски, ККД, питомі витрати, потужність; формування графіків.

7. Збереження результатів та формування звіту (таблиця + графіки).

Ключовий індикаторний показник обчислюється інтегруванням:

$$L_i = \oint P \, dV.$$

3.3.1 Вибір мови програмування та технологічного стеку

Під час розробки інформаційної системи «Цифровий двійник» дизельного двигуна вибір мови програмування та технологічного стеку є принциповим проєктним рішенням, оскільки він безпосередньо впливає на швидкість розробки, гнучкість моделі, можливість проведення серійних розрахунків та подальше масштабування системи. На відміну від прикладних інформаційних систем, у даному випадку пріоритет надається чисельному моделюванню фізичних процесів, а не максимальній продуктивності інтерфейсної частини.

Для реалізації обчислювального ядра та інструментів серійних експериментів у межах роботи обґрунтовано використання мови програмування Python. Основною причиною такого вибору є те, що Python оптимально підходить для задач 0D/1D-моделювання робочого циклу двигунів внутрішнього згоряння, де ключову роль відіграють чисельні методи, робота з масивами та постобробка результатів.

При виборі мови програмування розглядалися також альтернативні варіанти, зокрема C/C++, MATLAB та Java. Мова C/C++ забезпечує високу продуктивність і широко застосовується у промислових симуляторах, однак її використання значно ускладнює швидке прототипування, модифікацію моделей та експериментальну зміну алгоритмів. Для магістерської роботи, де модель активно доопрацьовується і розширюється, такий підхід є менш доцільним через високу вартість розробки й налагодження.

MATLAB є потужним інструментом для інженерних розрахунків і має зручні засоби візуалізації, проте його використання обмежене ліцензійними умовами та менш придатне для створення відкритих або масштабованих інформаційних систем. Крім того, інтеграція MATLAB-рішень у web-сервіси або клієнт–серверні архітектури потребує додаткових інструментів і ускладнює подальший розвиток платформи.

Java та подібні до неї мови більше орієнтовані на розробку корпоративних інформаційних систем і вебзастосунків. Хоча вони дозволяють створювати стабільні та масштабовані рішення, їх застосування для чисельного моделювання робочих процесів двигунів є менш зручним через складнішу роботу з математичними моделями та більший обсяг допоміжного коду.

На цьому тлі Python поєднує достатню обчислювальну ефективність з високою гнучкістю. Його синтаксис дозволяє безпосередньо реалізовувати математичні залежності, близькі до аналітичного запису, що спрощує перевірку формул і підвищує прозорість коду. Це особливо важливо для навчально-дослідної моделі, де коректність фізичної інтерпретації важливіша за максимальну швидкодію.

Типова структура технологічного стеку платформи сформована відповідно до функціональних рівнів інформаційної системи. Для чисельних обчислень і роботи з масивами даних використовуються стандартні засоби роботи з числовими структурами, що дозволяє ефективно обробляти профілі параметрів по куту колінчастого вала (\varphi), виконувати інтегрування та формувати індикаторні діаграми. Такий підхід є значно простішим і наочнішим порівняно з реалізацією аналогічних алгоритмів у низькорівневих мовах.

Для збереження вхідних конфігурацій і результатів розрахунків застосовуються формати JSON та CSV. Порівняно з використанням складних форматів або відразу повноцінних СУБД, ці формати забезпечують простоту, прозорість і зручність ручної перевірки даних. Формат JSON добре підходить для опису конфігурацій і метаданих запуску, тоді як CSV є зручним для збереження профільних масивів і їх подальшого аналізу. За потреби індексації та накопичення великої кількості запусків використовується SQLite, яка є компромісом між файловими рішеннями та повноцінними серверними базами даних.

Візуалізація результатів реалізується безпосередньо в середовищі Python, що дозволяє одразу будувати графіки залежностей (P(\varphi)), (T(\varphi)), індикаторні діаграми та порівняльні графіки для серійних розрахунків. На відміну від зовнішніх графічних інструментів, такий підхід забезпечує повну узгодженість графіків з даними розрахунку і не потребує ручного перенесення результатів.

Інтерфейсний шар системи на початковому етапі реалізується у вигляді локального сценарію запуску (CLI або простого GUI). Порівняно з повноцінним вебінтерфейсом, такий підхід є простішим і достатнім для навчальних та дослідних задач. Водночас архітектура системи одразу орієнтована на можливість переходу до web-API, де взаємодія з обчислювальним ядром будується за принципом:

Input (JSON) → Compute → Output (JSON).

Ключовим і критичним принципом обраного технологічного стеку є незалежність обчислювального ядра від інтерфейсу користувача та механізмів збереження даних. Це означає, що фізична модель робочого процесу дизельного двигуна не змінюється при переході від локального запуску до клієнт–серверного

режиму роботи. Саме така властивість є визначальною для цифрового двійника, оскільки дозволяє розглядати модель як стабільну основу, навколо якої можуть змінюватися та розвиватися інтерфейсні та сервісні компоненти.

3.3.2 Фізична модель даних та організація сховища результатів

Фізична модель даних визначає конкретний спосіб збереження інформації, що формується під час роботи інформаційної системи «Цифровий двійник» дизельного двигуна. Якщо концептуальна та логічна моделі описують що саме зберігається і як логічно пов’язані дані, то фізична модель відповідає на питання, де і в якій формі ці дані реально існують у файловій системі або базі даних.

Ключовим принципом фізичної організації сховища є чітке розділення різних типів даних, що використовуються в системі. Це дозволяє спростити доступ до результатів, підвищити відтворюваність експериментів і уникнути плутанини при роботі з серіями запусків. У межах проєкту прийнято розділяти дані на чотири основні групи: конфігураційні дані, результати метрик, профільні масиви та метадані запуску.

До конфігураційних даних належать усі вхідні параметри, які повністю визначають умови конкретного розрахунку. Вони включають параметри двигуна, режиму роботи, властивості палива та налаштування фізичних підмоделей. Фізично ці дані зберігаються у вигляді текстового файлу формату JSON, що дозволяє легко відновити умови запуску, перевірити їх вручну або використати як шаблон для наступних експериментів. Важливо, що конфігураційні дані зберігаються разом із результатами, а не окремо, що забезпечує повну відтворюваність кожного запуску.

Результати метрик є вихідними інтегральними показниками розрахунку. До них належать індикаторна робота, середні тиски, коефіцієнти корисної дії, питомі витрати та ефективна потужність. Фізично ці дані також зберігаються у структурованому форматі JSON, де кожен показник має значення, одиницю вимірювання та, за потреби, короткий опис. Такий формат є зручним як для автоматичної обробки, так і для формування звітів або порівняльних таблиць.

Профільні дані становлять найбільш об’ємну частину результатів. Вони містять масиви значень параметрів у функції кута колінчастого вала (φ), зокрема ($P(\varphi)$), ($T(\varphi)$), ($V(\varphi)$), ($x(\varphi)$), ($\sigma(\varphi)$) та похідні величини. Фізично ці дані доцільно зберігати у форматі CSV, де кожен стовпець відповідає окремому параметру, а рядки – дискретним значенням кута (φ). Такий підхід дозволяє швидко переглядати профілі, будувати графіки у зовнішніх інструментах та використовувати дані для подальшого аналізу без складних перетворень.

Окрему групу становлять метадані запуску, які не належать безпосередньо ні до вхідних параметрів, ні до результатів моделювання, але є критично важливими для організації сховища. До метаданих належать дата і час виконання запуску, використаний крок дискретизації (\Delta\varphi), версія обчислювального ядра, а також статус виконання (успішний або помилковий). Метадані дозволяють швидко орієнтуватися у великій кількості запусків і відокремлювати коректні результати від тестових або аварійно завершених.

На фізичному рівні для прототипу цифрового двійника достатньо використання файлової структури run-папок. Кожен запуск зберігається у власній директорії, яка містить файл конфігурації, файл з інтегральними результатами, файл або набір файлів з профільними даними та файл метаданих. Така організація є простою, наочною та зручною для ручної перевірки результатів, що є важливим на етапі розробки та налагодження моделі.

Однак при виконанні масових серій розрахунків або параметричних прогонів файловий підхід стає менш зручним, оскільки пошук потрібних запусків і порівняння результатів потребують перегляду великої кількості файлів. Тому у фізичній моделі даних передбачена можливість додавання локальної бази даних SQLite, яка використовується для індексації запусків і збереження ключових метаданих та інтегральних показників. У цьому випадку файлове сховище використовується для зберігання повних даних, а база даних – для швидкого пошуку, фільтрації та формування вибірок без «ручного» аналізу десятків або сотень файлів.

Таким чином, фізична модель даних поєднує простоту файлової організації з можливістю масштабування за рахунок використання легкої локальної бази даних. Такий підхід є оптимальним для навчально-дослідної платформи цифрового двійника, оскільки забезпечує прозорість, відтворюваність і зручність роботи з результатами як при одиночних, так і при серійних розрахунках.

3.3.3 Модель класів ІС «Цифровий двійник» дизельного двигуна

Модель класів потрібна тут не «для галочки», а щоб показати, що платформа цифрового двійника має структуру інформаційної системи: є вхідні дані, контроль коректності, обчислювальне ядро, постобробка, сховище результатів і представлення результатів користувачу. На рівні UML це найзручніше відобразити через набір логічних класів/компонентів із чіткими відповідальностями. Така формалізація також допомагає уникнути типової проблеми розрахункових проєктів, коли все «змішується» в одному файлі: вхід, фізика, графіки й збереження, і потім будь-які зміни починають ламати весь ланцюжок.

У даній ІС доцільно виділити такі основні класи (компоненти), які напряму відповідають вашій модульній декомпозиції (*io_config, thermo_cycle, spray_evap, combustion, heat_transfer, integrator, metrics, viz* тощо), але подані у вигляді UML-структури.

1) Класи конфігурації та доступу до вхідних даних

Config (EngineParams, ModeParams, FuelRef, ModelSettings)

Це центральна структура вхідних параметрів запуску. Логічно вона складається з чотирьох підоб'єктів:

- EngineParams – геометрія та кінематика: (D), (S), (l/r), кількість циліндрів, (ε), дані для розрахунку ($V(\varphi)$).
- ModeParams – режимні умови: (n), (P_0), (T_0), наддув, параметри газообміну/наповнення.
- FuelRef – посилання на паливо (ключ або назва запису у довіднику), щоб не дублювати властивості палива в кожному запуску.
- ModelSettings – параметри підмоделей: політропи, теплообмін, корекції, механічний ККД, крок ($\Delta\varphi$) тощо.

Головна ідея: Config – це «контракт» між користувачем і ядром. Якщо Config повний і коректний, то ядро має видати повний результат без ручних правок у коді.

ConfigLoader

Відповідає за завантаження конфігурації з JSON і приведення до внутрішнього представлення. Типова задача цього класу – нормалізація одиниць (наприклад, бар → Па, °C → K) і приведення типів. Важливий момент: все, що стосується «читання/перетворення», має бути локалізовано тут, щоб фізичні модулі не займалися форматами і «випадковими» переведеннями.

FuelRepository

Забезпечує доступ до довідника палив: пошук за ключем, повернення структури Fuel, перевірка наявності запису. На рівні ІС це окремий клас, тому що паливо – це типовий довідниковий об'єкт, який оновлюється рідше і використовується багаторазово. У перспективі саме цей клас зручно розширювати (наприклад, додати версіювання властивостей палива або облік сумішей).

2) Клас контролю коректності

Validator

Ключовий клас для інженерної системи. Його задача – відсікти помилки до того, як користувач отримає «красивий, але неправильний» результат.

На практиці Validator виконує три рівні перевірок:

1. Перевірка меж і знаків: ($\varepsilon > 1$), (P_0 > 0), (T_0>0), геометрія (>0) тощо.

2. Логічна узгодженість: параметри впорскування не виходять за межі циклу, крок (\Delta\varphi) має адекватне значення, не задані взаємовиключні режими/опції.
3. Одиниці вимірювання: контроль того, що перетворення одиниць виконано однозначно (щоб не було ситуацій «частина вже в Па, частина ще в бар»).

Цей клас «знімає» основний ризик інженерного моделювання: коли система працює без помилок, але дає фізично абсурдні числа через одну неправильну одиницю.

3) Класи обчислювального ядра

CycleCore

Це «серце» системи: реалізація моделі повного робочого циклу. Логічно CycleCore агрегує підмодулі:

- кінематика КШМ та ($V(\varphi)$);
- термодинамічні процеси стискання/розширення;
- підмоделі газообміну, теплообміну;
- моделі впорскування/розпилу/випаровування;
- модель тепловиділення/згоряння;
- передача даних інтегратору та формування службових масивів.

На рівні UML CycleCore доцільно показувати як клас-компонент, який приймає Config + Fuel і повертає «сирий» результат інтегрування (профілі та службові дані).

Integrator

Відповідає за реалізацію чисельної схеми інтегрування по (φ). Це окремий клас, бо інтегратор – типове місце, де виникають проблеми стійкості та точності. Integrator має:

- працювати з фіксованим ($\Delta\varphi$);
- контролювати стабільність (відсікати нефізичні стани типу (T<0));
- повертати не лише ($P(\varphi)$), ($T(\varphi)$), а й ознаки збою/попередження (наприклад, «step too large», «non-physical state»).

Практично це дозволяє систему зробити «керованою»: замість хаотичних “nan” користувач отримує нормальне повідомлення, що саме пішло не так.

4) Класи постобробки та формування результату

PostProcessor / Metrics

Цей компонент приймає профілі та обчислює інтегральні показники. Логічно він не повинен знати нічого про те, як було проведено інтегрування – він працює тільки з масивами.

Типовий набір задач:

індикаторна робота:

$$L_i = \oint P\,, dV;$$

розрахунок (p_i), (p_e), ККД, питомих витрат, потужності;

формування структури Metrics з одиницями вимірювання.

Важливо, що відділення Metrics від CycleCore дає дуже практичний плюс: якщо змінюється модель згоряння або теплообміну, але профілі формуються, то блок метрик лишається стабільним і перевіреним.

5) Класи збереження та управління історією запусків

RunStore

Відповідає за збереження «одиниці відтворюваності» – Run. На фізичному рівні це може бути файлове сховище run-папок, а для серій – додаткова індексація у SQLite. Але в UML це один логічний клас із методами:

- *save_run(config, profiles, metrics, metadata);*
- *load_run(run_id);*
- *list_runs(filters) (для вибірки за режимом, паливом, датою);*
- *mark_failed(run_id, reason).*

Саме цей клас реалізує дисципліну даних: щоб будь-який запуск потім можна було знайти, повторити і порівняти.

6) Класи представлення результатів

Visualizer / Reporter

Має будувати стандартні графіки й таблиці: $(P(\varphi)), (T(\varphi))$, індикаторну діаграму, порівняльні накладання, підсумкові таблиці метрик. Важливо, що цей клас працює лише з готовими даними (Profiles, Metrics) і не втручається у фізику. Інакше з'являється типова помилка: «візуалізація починає щось перераховувати», і тоді вже незрозуміло, де істинний результат.

7) Клас керування сценарієм (оркестрація)

Controller

Це компонент, який реалізує сценарій:

load → validate → compute → postprocess → save → visualize.

Controller не містить термодинаміки і не займається графіками. Він просто «склеює» компоненти в один робочий процес. Саме через Controller легко реалізувати і UC-1 (одиночний запуск), і UC-2 (серійний прогін), і UC-3 (порівняння запусків), змінюючи лише логіку керування, а не ядро.

Відповідність модульній декомпозиції проєкту

Описана модель класів напряму відповідає вашій модульній структурі:

- *io_config → ConfigLoader + Validator*
- *fuel_data / fuel_types → FuelRepository*
- *thermo_cycle, spray_evap, combustion, gas_exchange, heat_transfer →* внутрішні підмодулі CycleCore
- *integrator → Integrator*
- *metrics → PostProcessor/Metrics*
- *viz → Visualizer/Reporter*
- *data layer (JSON/CSV/SQLite) → RunStore*
- *сценарії запусків → Controller*

Таким чином, на UML-рівні платформа подається як інформаційна система зі зрозумілими компонентами та межами відповідальності. Це підкреслює ключовий принцип цифрового двійника: Core ізольований від UI та сховища, а тому зміна інтерфейсу або способу збереження не призводить до переписування фізики моделі.

3.3.4 Моделі взаємодії ІС «Цифровий двійник» дизельного двигуна

Модель класів показує, з яких компонентів складається система, але не пояснює, як саме вони взаємодіють у часі під час виконання розрахунку. Тому для ІС «Цифровий двійник» доцільно додатково використати моделі взаємодії. Вони дають два практичні ефекти: по-перше, фіксують керовану логіку запуску (що за чим іде), по-друге, дозволяють явно описати, як система поводиться при помилках введення або при збоях чисельного інтегрування. У підсумку цифровий двійник виглядає не як «ланцюжок формул», а як керована ІС з формалізованими сценаріями.

1) Діаграма послідовностей (Sequence) для основного запуску (UC-1)

Діаграма послідовностей описує типову взаємодію компонентів у випадку одиночного локального розрахунку. На практиці це найчастіший сценарій: користувач має файл конфігурації, запускає модель і отримує метрики, профілі та графіки.

Учасники взаємодії (lifelines) та їх роль:

- User – ініціатор запуску, формує конфігурацію і запускає моделювання.
- Interface – приймає команду запуску, показує повідомлення/результати.
- Controller – керує процесом “load → validate → compute → save → visualize”.
- ConfigLoader/Validator – завантажує JSON, приводить одиниці, перевіряє коректність параметрів.
- FuelRepository – надає властивості палива з довідника.

- CycleCore/Integrator – виконує модель циклу й інтегрування по (φ), повертає профілі.
- Metrics – обчислює інтегральні показники, включаючи
- RunStore – зберігає результати як окремий run з метаданими.
- Visualizer – формує графіки/таблиці та повертає їх у інтерфейс.

Типова послідовність повідомлень (словесна форма, відповідно до вашої схеми):

1. User → Interface: команда запуску + шлях до parameters.json (або вибір профілю конфігурації).
2. Interface → Controller: start_run(config_path, options)
3. Controller → ConfigLoader: load(config_path)
4. ConfigLoader → Validator: validate(config)
    - якщо помилка введення – повернення помилки з описом параметра, діапазону й одиниць (див. альтернативний фрагмент нижче).
5. Controller → FuelRepository: get_fuel(config.fuel_ref)
    - якщо палива немає – помилка “fuel not found”.
6. Controller → CycleCore: compute(config, fuel)
7. CycleCore → Integrator: integrate_over_phi(...)
    - Integrator повертає profiles + warnings/errors.
8. CycleCore → Controller: повернення профілів $\big(P(\varphi)\big), \big(T(\varphi)\big), \big(V(\varphi)\big)$, службових масивів $(x, (\sigma)$, тощо).
9. Controller → Metrics: calc_metrics(profiles, config)
10. Controller → RunStore: save_run(config, profiles, metrics, metadata)
11. Controller → Visualizer: render(profiles, metrics, style)
12. Visualizer → Interface → User: показ графіків, таблиць, підсумків.

Альтернативні гілки (alt fragments), які варто закласти в Sequence:

- alt: ValidationError
- Якщо Validator виявляє некоректний параметр (наприклад, (ε ≤ 1), (P_0 \le 0), конфлікт одиниць), Controller завершує сценарій без запуску ядра, а Interface виводить повідомлення з причиною і підказкою.
- alt: IntegrationUnstable
- Якщо Integrator фіксує нефізичні значення (наприклад (T<0), (x>1), $(\sigma <$ 0)), Controller формує run.status="failed", додає reason, зберігає мінімальні метадані та (за потреби) часткові профілі для діагностики, але не включає цей запуск до “успішних” порівнянь.

Ці альтернативи важливі методично: вони показують, що система не просто «рахує», а контролює коректність і керує помилками.

2) Модель переходів станів (State machine) для життєвого циклу запуску (Run lifecycle)

Модель станів описує, як змінюється стан системи (або конкретного run) у процесі виконання. Для цифрового двійника це доцільно формулювати саме через стан run, бо run є одиницею відтворюваності і має статус.

Основний ланцюжок станів:

- Idle – система готова, активних розрахунків немає.
- ConfigLoaded – конфігурація завантажена, одиниці приведено до внутрішнього формату.
- Validated – параметри пройшли перевірки меж і логіки.
- Running – виконується інтегрування циклу по (φ) у ядрі.
- PostProcessing – обчислюються метрики, формуються підсумкові структури даних.
- Saved – run збережено у сховищі (файли/SQLite-індекс).
- Completed – результати візуалізовані та доступні користувачу.

Гілка помилки:

- На будь-якому етапі (зазвичай на Validated або Running) можливий перехід у стан Error.
- Для стану Error обов’язково фіксуються:
  - reason (тип помилки: validation/integration/storage),
  - where (модуль/етап),
  - timestamp,
  - run.status = "failed".

Практичний сенс такого стан-машину – дисципліна. Наприклад, якщо run перейшов у Error, він не повинен “висіти” як успішний і підмішуватися в серійні вибірки. А якщо run дійшов до Saved, то його можна гарантовано використовувати для порівняння та повторного запуску.

3) Діаграма діяльності (Activity) для серійного прогона (UC-2)

Діаграма діяльності потрібна там, де є цикли та розгалуження. Серійний прогін – якраз такий випадок: система автоматично запускає багато конфігурацій, при цьому має відсіювати помилкові результати і формувати підсумкові таблиці.

Логіка Activity-сценарію (словесно):

1. Старт → завантаження базової конфігурації.
2. Формування плану експерименту:
   - вибір параметра(ів) для sweep;
   - перелік значень;
   - генерація набору конфігурацій (множина Config).
3. Цикл запусків (for each Config):

   - load/validate
   - якщо validation fail → позначити як “skipped/failed”, записати причину, перейти до наступної конфігурації;
   - compute (CycleCore + Integrator)
   - якщо integration fail → run.failed, записати причину, перейти до наступної;
   - metrics
   - save_run (і, за потреби, індексація SQLite).
4. Агрегування результатів:
   - збір Metrics в одну таблицю;
   - розрахунок різниць/трендів, мін/макс, чутливостей.
5. Порівняльна візуалізація/звіт:
   - накладання графіків;
   - таблиця залежностей “параметр → метрика”;
   - формування підсумкового звіту.
6. Фініш.

Ключова ідея activity-моделі: серійний прогін – це не «запустити багато разів», а керований експеримент, де система сама:

- генерує конфігурації,
- виконує типову процедуру UC-1,
- контролює коректність,
- накопичує метрики,
- формує порівняння.

Узагальнення ролі моделей взаємодії

У підсумку діаграма послідовностей, модель станів і activity-діаграма доповнюють модель класів і дозволяють описати роботу цифрового двійника як керованої інформаційної системи. Саме ці моделі показують, що платформа має формалізовані сценарії, контроль якості результатів та логіку обробки помилок, тобто відповідає ідеології «цифрового двійника», а не є просто «скриптом з формулами».

3.4 Проєктні рішення щодо модульності та підтримуваності

На практиці на старті розробки часто виходить так, що робочий прототип швидше зібрати в одному файлі: це простіше для налагодження, легше бачити весь ланцюжок обчислень і швидко перевіряти формули. Але для «платформи цифрового двійника» та ще й у межах магістерської роботи важливо показати, що система не є одноразовим скриптом. Вона має бути такою, щоб її можна було нормально супроводжувати: додавати нові підмоделі, змінювати формат вхідних

даних, робити серійні запуски, розширювати візуалізацію, і при цьому не ламати вже перевірені частини.

Тому проєктне рішення формулюється так: навіть якщо зараз реалізація зібрана в одному файлі, архітектурно код розбивається на логічні компоненти з чіткими відповідальностями. Це дає три практичні переваги:

1. Зменшення зв'язності (coupling). Термодинаміка не залежить від способу читання JSON чи від побудови графіків.
2. Керовані зміни. Якщо треба замінити модель теплообміну або додати інший закон згоряння – змінюється один модуль, а не весь проєкт.
3. Тестованість. Модульні функції легше перевіряти юніт-тестами: простіше знайти помилку й не отримувати "дивні" результати через одну неправильну формулу.

У межах проєкту доцільна така раціональна декомпозиція (назви модулів наведено як орієнтир, їх можна реалізувати як окремі файли або як розділи/класи в одному файлі з наступним винесенням):

- io_config – завантаження вхідних параметрів, перевірка на повноту, контроль допустимих меж, приведення одиниць. Саме тут логічно зосередити "захист від дурня": наприклад, якщо переплутали бар і Па, або задали ступінь стиску як 0.18 замість 18.
- thermo_cycle – базові процеси циклу: наддув, наповнення, стискання, розширення, розрахунок характерних точок. Тут концентруються основні термодинамічні залежності та розрахункові параметри циклу.
- spray_evap – впорскування, розпил і випаровування. У цьому модулі зручно формувати допоміжні функції, зокрема залежності типу $\sigma(\varphi)$, які потім використовуються в моделі згоряння.
- combustion – розрахунок тепловиділення та закону згоряння: функція $x(\varphi)$, похідна $dx/d\varphi$, фази горіння, підбір/задання коефіцієнтів. Важливо, щоб модуль повертав не лише "підсумок", а й профільні масиви для аналізу.
- gas_exchange – моделі впуску/випуску, витрати через клапани, оцінка впливу газообміну на крайові умови циклу (особливо якщо вводиться змінність по режимах).
- heat_transfer – теплообмін, коефіцієнти типу α_g та перехідні коефіцієнти на кшталт $k_{\text{об}}$, якщо вони у вас присутні в моделі. Сюди ж логічно поміщати розрахунок втрат теплоти по куту φ.
- integrator – «збірка» всього циклу: інтегрування по куту колінчастого вала, формування профілів $P(\varphi), T(\varphi)$, службові масиви, контроль стабільності чисельної схеми.

- metrics – обчислення інтегральних показників: робота циклу, середні тиски, ККД, питомі витрати, потужність. Наприклад, індикаторна робота:

Важливо, щоб цей модуль приймав уже готові профілі (масиви) і повертав метрики в структурованому вигляді.

- viz – побудова графіків і підсумкових таблиць. Рішення принципове: візуалізація не втручається у фізику, вона тільки “пояснює” результати.

Таке проєктне рішення прямо підтримує ключову ідею цифрового двійника: обчислювальне ядро незалежне від інтерфейсів, а отже, його можна буде безболісно використати як основу для web-API (рис. 3.1).

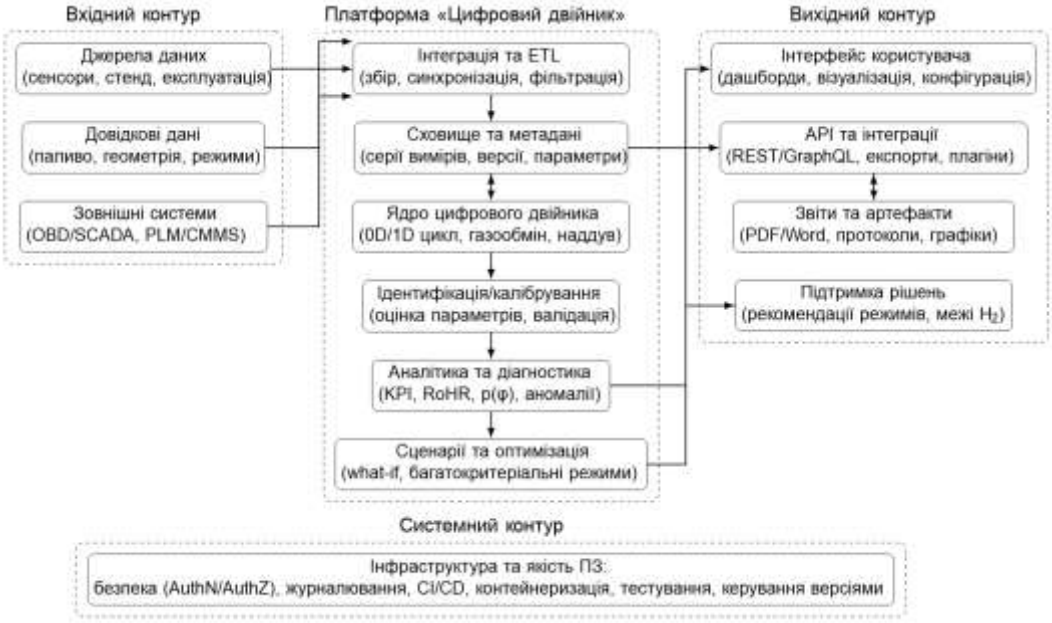


Рисунок 3.1 – Загальна схема модульної структури платформи «Цифровий двійник» дизельного двигуна (логічна декомпозиція на компоненти).

3.5 Проєктні рішення щодо інтерфейсів

Інтерфейс у такій системі не є “красивою оболонкою”. Він визначає, як користувач реально працює з цифровим двійником: як задає параметри, як запускає розрахунок, як порівнює результати та як зберігає експерименти. Тому проєктні рішення щодо інтерфейсів доцільно описувати у двох площинах: що є вже зараз (локально) і як це має масштабуватися до сервісу (web-API).

Локальний рівень (поточна реалізація/прототип)

Для прототипу достатньо локального сценарію, який закриває базові задачі навчального та дослідного використання:

- Запуск моделювання з файлу параметрів. Користувач готує конфігурацію (JSON), запускає модель, отримує результат. Це найпростіший і найнадійніший варіант для відтворюваності.
- Вибір палива з довідника. Паливо не “зашивається” в коді, а береться з окремого опису (довідника), щоб легко додавати альтернативні варіанти або змінювати властивості без правок у розрахункових модулях.
- Автоматичне формування графіків і таблиць. Після запуску користувач отримує не лише числа, а й стандартний набір графіків, які реально використовуються для аналізу циклу.

На цьому етапі найважливіше, щоб інтерфейс не створював “ручних залежностей”, коли користувач щось вводить у коді або змінює формули прямо перед запуском. Якщо параметри завжди приходять з конфігурації, а результат завжди зберігається разом із нею – тоді серійні експерименти стають нормальними експериментами, а не набором випадкових запусків (рис. 3.2).

Рисунок 3.2 – Локальний сценарій роботи платформи: конфігурація → запуск моделювання → формування результатів → збереження → побудова графіків.

Перспектива web-API (архітектурна готовність)

Оскільки тема роботи передбачає платформу, логічно закласти рішення, яке дозволяє “без переробки фізики” перейти до клієнт–серверного формату. Тут ключовий принцип такий:

- ядро оформлюється як функція або клас, що приймає структуру параметрів і повертає структуру результатів;
- візуалізація і UI – лише надбудова, яка працює з результатом, але не є частиною обчислень (рис. 3.3).

Тоді web-API стає природним продовженням локальної логіки. Формально це описується як:

$$\text{Input (JSON)} \rightarrow \text{Compute} \rightarrow \text{Output (JSON)}$$

де:

- Input містить ті самі параметри, що й локальний parameters.json (можливо, з невеликими додатковими полями типу “id експерименту” або “назва сценарію”);
- Output містить:
  - інтегральні показники (ККД, тиски, витрати, потужність тощо);
  - профілі (масиви по φ) для графіків і порівнянь;
  - метадані (крок по φ, одиниці, версія моделі, час запуску).

Практична користь такого рішення в тому, що веб-частина може змінюватися як завгодно (інший дизайн, інша бібліотека графіків, навіть інший клієнт), але формат даних і саме ядро залишаються стабільними. Для цифрового двійника це важливо: ядро – це «модель двигуна», а інтерфейс – лише спосіб з нею працювати

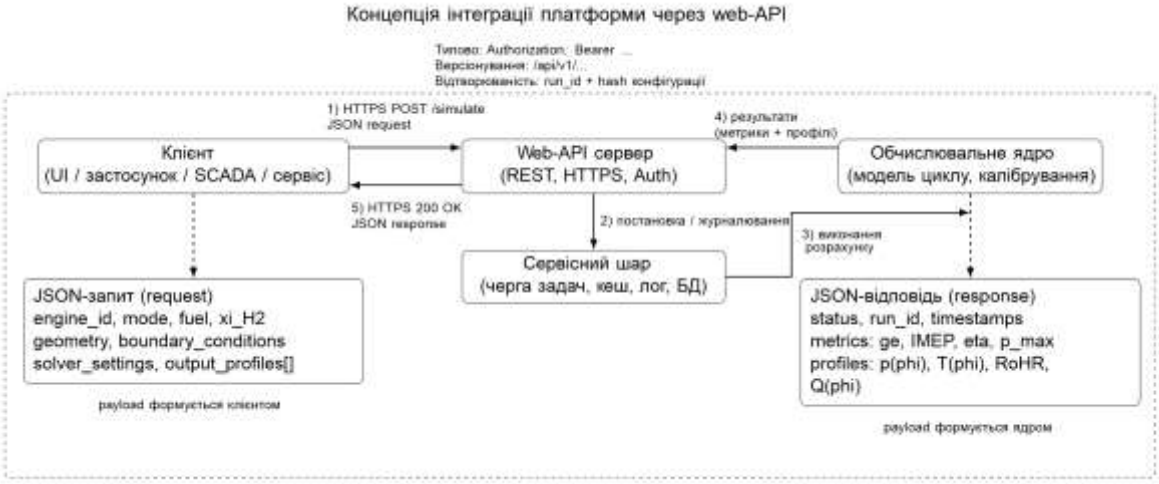


Рисунок 3.3 – Концепція інтеграції через web-API: клієнт формує JSON-запит → сервер виконує розрахунок → повертає JSON-відповідь з метриками та профілями.

3.6 Проєктні рішення щодо тестування та контролю коректності

Для платформи, яка моделює повний робочий цикл дизеля, питання якості – це не тільки «щоб не падала програма». Тут головне, щоб результат не був формально отриманим, але фізично безглуздим. Найгірша ситуація в інженерних розрахунках – коли код працює, графіки будуються, числа виходять, але десь «тихо» закралася помилка в одиницях, у знаку, у кроці інтегрування або в параметрах моделі. Саме тому в проєкті закладається мінімальний, але обов’язковий набір перевірок, які відсікають типові помилки ще до того, як користувач почне інтерпретувати результати (рис. 3.4).

По-перше, виконується валідація вхідних даних. Вхідні параметри перевіряються на допустимі межі та елементарну логіку. Наприклад, геометричні розміри не можуть бути від'ємними, ступінь стиску має бути більшим за 1, тиск і температура навколишнього середовища мають бути фізично реальними, а параметри впорскування не повинні виходити за межі одного циклу. Окремо контролюється узгодженість одиниць: якщо вхід задається в бар, а ядро працює в Паскалях, перетворення має бути однозначним і виконуватися в одному місці (на рівні io_config). Так простіше уникати ситуацій, коли одна змінна вже переведена, а інша – ні.

По-друге, після розрахунку вводяться перевірки фізичної узгодженості. Їхня задача – ловити «неможливі» стани, які часто виникають через помилки в формулах або нестабільність чисельної схеми. Базовий набір таких перевірок включає:

- тиск і температура на всіх кроках мають бути додатними (не допускаються стрибки в нуль або «мінус»);
- маса заряду, масові витрати та масові частки не можуть ставати від'ємними;
- функція $\sigma(\varphi)$ повинна залишатися в межах [0;1];
- частка згорілого палива x(\varphi) також повинна бути в межах [0;1].

Ці обмеження на практиці дуже корисні: якщо десь з'являється $x(\varphi) > 1$ або $\sigma(\varphi) < 0$, це означає, що або неправильно задані коефіцієнти моделі, або некоректно описана форма функції, або збився крок інтегрування. Користувач отримує не «дивний графік», а чіткий сигнал, що саме порушено.

По-третє, у проєкті передбачаються регресійні тести. Ідея проста: задається один або кілька «контрольних» наборів параметрів (еталонні конфігурації), і після будь-яких змін у коді система порівнює, чи збігаються ключові вихідні показники з попередньо зафіксованими. Для моделі циклу це може бути набір на кшталт: $p_i, p_e, L_i,$, піковий тиск, положення максимуму тиску за кутом, а також окремі контрольні значення профілю $P(\varphi)$ у кількох точках. Такий підхід дисциплінує розробку: якщо після «невинного» рефакторингу раптом змінюється p_i – значить, десь зміни торкнулися логіки обчислень.

Рисунок 3.4 – Схема контролю коректності: вхідна валідація → розрахунок → фізичні перевірки → регресійне порівняння → збереження результатів.

По-четверте, обов'язково контролюється вплив кроку інтегрування Δφ. У розрахунках по куту колінчастого вала дрібні деталі (особливо в зоні згоряння) сильно залежать від дискретизації. Тому в проєктному рішенні закладено простий тест чутливості: один і той самий режим рахується при двох-трьох значеннях Δφ (наприклад, грубіше і точніше), і порівнюється, як змінюються інтегральні метрики. Якщо розкид перевищує задану межу, це означає, що крок треба зменшити або стабілізувати чисельну схему. Такий контроль корисний ще й тим, що дозволяє обґрунтовано вибрати «розумний» крок: достатньо точний, але без зайвого часу рахування.

3.6.1 Тестування ІС «Цифровий двійник» дизельного двигуна

Тестування у даному проєкті розглядається як обов'язкова частина розробки, оскільки система виконує інженерні чисельні розрахунки, де типова проблема – не "помилка виконання", а тиха помилка, коли програма працює, але результат фізично некоректний через одиниці, знак, крок інтегрування або некоректні межі параметрів. Тому тестування тут спрямоване на дві речі: (1) контроль правильності окремих функцій і вузлів, (2) фіксацію стабільності результатів після змін у коді.

У проєкті використовується реальна структура даних і файлів платформи: *parameters.json* як джерело вхідних параметрів, *calculation_results.json* (та альтернативний файл результатів на іншому середовищі) як еталонні значення для регресійного контролю, а також *engine_calculations.db* як сховище для серійних запусків і накопичення обчислень. Це дозволяє тестувати не "в абстракції", а саме той сценарій, який виконується під час роботи цифрового двійника.

Тестування охоплює:

1) Модульні тести (unit tests).

Мета модульних тестів – перевірити коректність окремих функцій і мінімізувати ризик, що одна помилка в низькорівневій формулі "потягне" за собою спотворення всього циклу.

Доцільно виділяти такі групи модульних тестів (на прикладі функцій і файлів проєкту):

Геометрія/кінематика ($V(\varphi)$) та похідні.

Перевіряються функції calc_volume(phi_{deg}, V_s, epsilon, lambda_ratio) та calc_dvolume (phi_{deg}, V_s, lambda_ratio):

при ($\varphi = 0°$) об'єм має дорівнювати об'єму камери згоряння ($V_c = \frac{V_s}{\varepsilon - 1}$) (тобто мінімум об'єму);

при ($\varphi = 180°$) об'єм має дорівнювати ($V_c + V_s$) (максимум об'єму);

похідна ($dV/d\varphi$) на крайніх положеннях (0°, 180°) повинна бути близькою до нуля;

додатково корисно перевіряти узгодженість *calc_dvolume* через чисельну апроксимацію (скінченні різниці) на кількох значеннях (φ), щоб переконатися, що аналітична похідна не "з'їхала" після редагування формули.

Перетворення одиниць і завантаження даних.

Для функцій *load_parameters()* та *load_calc_results()* (читання *parameters.json* і *calculation_results.json*) перевіряється:

коректне зчитування числових значень і структура словника;

реакція на помилки (відсутній файл / некоректний JSON);

однозначність одиниць: наприклад, тиск з *parameters.json* заданий у кПа (*environment_pressure*: 101.0), тоді як в обчисленнях частина величин переводиться у Па або МПа – відповідно, тест має "ловити" ситуації, коли параметр випадково вже був переведений раніше або, навпаки, не переведений.

Контроль фізичних меж для ($x(\varphi)$) і ($\sigma(\varphi)$).

Навіть якщо самі функції $\big(x(\varphi)\big), \big(\sigma(\varphi)\big)$ формуються в модулі згоряння, тестовий принцип простий: будь-яка модель згоряння зобов'язана повертати значення у межах ([0;1]).

Тому створюється набір перевірок типу:

min(x) >= 0, max(x) <= 1;

min(sigma) >= 0, max(sigma) <= 1;

відсутність NaN/Inf у профілях.

Перевірка інтегральних показників (напр. (L_i)).

Мінімальний тест – коректність знаку та масштабу: якщо тиск всюди додатній, а цикл "замкнений" правильно, то (L_i) не повинен виходити від'ємним або мати порядок величини, що суперечить типовим значенням для заданого робочого об'єму.

Робота довідника палив.

Для *fuel_data.py* і функції *add_fuel_type()* доцільні тести:

додавання нового типу палива з повним набором полів (C,H,O,S, (Q_n), Cetane, Density, Viscosity, Activation_Energy);

відмова при дублюванні ключа;

відмова при відсутності обов'язкового параметра.

Файлова/БД-частина (RunStore).

Для *database.py* (SQLite) перевіряється, що:

create_database() створює таблиці *basic_calculations* і *detailed_calculations*;

save_*_calculation_results() реально додає запис, і поля зберігаються в числовому форматі без “зсуву” одиниць.

Результат модульного тестування: отримується базова впевненість, що “цеглинки” цифрового двійника працюють правильно окремо.

2) Регресійні тести (regression tests).

Мета регресійних тестів – зафіксувати, що після змін у коді система не почала давати інші результати на тому самому режимі, якщо змін у фізичній моделі не планувалося.

У проєкті регресійна база може формуватися на основі:

- еталонної конфігурації *parameters.json*;
- еталонних розрахункових результатів *calculation_results.json* (як контрольний “знімок” ключових величин).

Практично регресійний тест виглядає так:

- запуск моделі з еталонною конфігурацією;
- порівняння ключових показників з еталоном у допустимій похибці (зазвичай задається відносна похибка або абсолютна – залежно від величини).

До списку контрольних величин доцільно включати (саме те, що в проєкті вже є і зберігається):

$$(\varepsilon), (V_s);$$
$$(P_k, T_k, P_s, T_s);$$
$$(T_a, \gamma_r, \eta_H);$$

індикаторні/ефективні показники (за наявності в розрахунках): P_i, P_e, η_i, η_e;

P_{max} та кутове положення максимуму тиску (як якісно дуже чутлива характеристика).

Окремо, якщо в подальшій версії ядра з’являються повні профілі $P(\varphi)$, то для регресії зручно фіксувати контрольні точки (наприклад, $P(0°), P(180°), P(\varphi_{inj}), P(\varphi_{max})$, щоб не зберігати весь масив як еталон.

Результат регресійного тестування: після будь-якого рефакторингу або дрібних змін у коді одразу видно, чи “попливли” ключові результати.

3) Тести чутливості до ($\Delta\varphi$) (чисельна збіжність).

Оскільки цифровий двійник виконує інтегрування по куту колінчастого вала, крок ($\Delta\varphi$) є одним з найважливіших чисельних параметрів. Занадто великий крок може давати:

зміщення піка тиску;

завищення/заниження індикаторної роботи;

нефізичні значення (T) або ($x(\varphi)$) через нестійкість.

Тому у тестовому наборі задається одна й та сама конфігурація, яка рахується при кількох кроках, наприклад:

$$(\Delta\varphi = 1^\circ),$$
$$(\Delta\varphi = 0.5^\circ),$$
$$(\Delta\varphi = 0.25^\circ).$$

Далі порівнюються інтегральні показники (насамперед L_i, p_i, P_{max}, положення P_{max}). Якщо при зменшенні кроку зміни стають малими і стабілізуються, це є практичним підтвердженням збіжності та коректності дискретизації. Такий тест корисний ще й для обґрунтування "робочого" кроку: достатньо точного, але без зайвого часу рахування серій.

Результат тестів ($\Delta\varphi$): підтверджується, що отримані показники не є артефактом грубої дискретизації.

Підсумок і мета тестування

Метою тестування є фіксація того, що після змін у коді платформа зберігає узгодженість і відтворюваність результатів:

модульні тести гарантують правильність ключових формул і обробки даних;

регресійні тести "страхують" від непомітних змін результатів;

тести чутливості до ($\Delta\varphi$) підтверджують чисельну стійкість і збіжність.

У сукупності це дозволяє використовувати цифровий двійник як інструмент для серійних досліджень (порівняння режимів, прогін параметрів, аналіз впливів), не боячись, що результати "гуляють" через випадкові зміни в реалізації.

3.6.2 Випробування ІС «Цифровий двійник» дизельного двигуна

На відміну від тестування, яке перевіряє окремі функції та контрольні значення, випробування розглядаються як перевірка системи в цілому – тобто у форматі реальної експлуатації за типовими сценаріями користувача. Мета випробувань у межах даного проєкту – підтвердити, що платформа як інформаційна система працює стабільно, відтворювано, коректно зберігає дані та забезпечує користувачу результати в тому вигляді, в якому їх можна аналізувати (метрики + профілі + візуалізація).

Випробування виконуються у двох режимах:

Функціональні випробування (перевірка повного ланцюжка "конфігурація → розрахунок → збереження → візуалізація").

Експлуатаційні випробування (серійні прогони, накопичення запусків, пошук/порівняння, контроль стійкості та продуктивності).

1) Функціональні випробування за сценаріями використання

Випробування В-1. Одиночний запуск (аналог UC-1)

Мета: перевірити, що система повністю виконує одиночний розрахунок і формує стандартний пакет результатів.

Вхідні дані: еталонна конфігурація parameters.json (двигун/режим/паливо/налаштування).

Порядок виконання: –

Завантажити конфігурацію через ConfigLoader.

Виконати валідацію параметрів через Validator.

Отримати властивості палива через FuelRepository.

Запустити CycleCore та інтегрування по (\varphi) (Integrator).

Виконати постобробку (Metrics) і розрахунок інтегральних показників. Зберегти запуск як Run у сховищі (RunStore) разом із метаданими.

Побудувати графіки/таблиці (Visualizer).

Критерії успішності:

- run має статус Completed;
- присутні файли/записи: конфігурація, метрики, профілі (або їх еквівалент), метадані;
- відсутні NaN/Inf у результатах;
- базові фізичні перевірки виконані (тиск/температура додатні, $(x(\varphi) \in [0; 1])$, $(\sigma(\varphi) \in [0; 1])$).

Випробування В-2. Запуск із помилкою введення (аналог сценарію S-2)

Мета: підтвердити, що система коректно зупиняє запуск на етапі валідації та не формує “псевдорезультатів”.

Вхідні дані: модифікована конфігурація (наприклад, $(\varepsilon \leq 1)$, $(P_0 \leq 0)$ або некоректна одиниця/формат).

Порядок виконання: запуск як у В-1 до етапу Validator.

Очікувана реакція системи:

- запуск зупиняється до входу в CycleCore;
- користувачу повертається повідомлення з вказанням параметра, діапазону та одиниць;
- run, якщо створюється, має статус Error/Failed і збережену причину помилки;
- система не “підмішує” такий запуск в історію успішних експериментів.

Критерій успішності: помилка локалізована і пояснена, відсутні частково сформовані “неповні” результати, які можуть вводити в оману.

Випробування В-3. Реакція на нестійкість інтегрування (аналог сценарію S-3)

Мета: перевірити контроль нефізичних значень і правильне завершення запуску при збоях чисельної схеми.

Вхідні дані: конфігурація з навмисно грубим кроком (Δφ) або з "жорсткими" параметрами підмоделей, що можуть спричинити нестійкість.

Очікувана реакція системи:

- при появі (T<0), (x>1), ($\sigma < 0$) або інших нефізичних станів інтегратор фіксує збій;
- запуск завершується зі статусом Failed, причина записується в метадані;
- користувач отримує рекомендацію (наприклад, зменшити (Δφ), переглянути коефіцієнти теплообміну/згоряння).

Критерій успішності: система не "мовчить" і не видає некоректний графік як результат; run маркується і не потрапляє у вибірки успішних.

2) Експлуатаційні випробування (серійні прогони та робота зі сховищем)

Випробування В-4. Серійний прогін (аналог UC-2)

Мета: перевірити працездатність серійних експериментів і накопичення результатів без ручної обробки файлів.

Вхідні дані: базова конфігурація + перелік значень одного параметра (наприклад, (φ_{inj}), (Δφ), коефіцієнт теплообміну або інший налаштувальний параметр).

Порядок виконання:

Генерація набору конфігурацій (пакет run'ів).

Послідовний автоматичний запуск кожної конфігурації.

Накопичення Metrics у зведену таблицю.

Побудова порівняльних графіків і/або звіту.

Критерії успішності:

- кожен run має власний ідентифікатор та метадані;
- успішні й помилкові запуски відокремлені;
- зведена таблиця формується автоматично;
- система не вимагає ручного "перенесення" значень між файлами.

Випробування В-5. Перевірка збереження та пошуку результатів у сховищі

Мета: підтвердити, що результати після серії запусків не перетворюються на "хаос файлів", а доступні для відбору й порівняння.

Реалізація:

- для прототипу: перевіряється цілісність структури run-папок (наявність конфігурації, результатів, профілів, метаданих);
- для серій: використовується індексація в SQLite (таблиці на кшталт basic_calculations, detailed_calculations) для відбору run'ів за режимом/паливом/датою або параметром експерименту.

Критерії успішності:

- за заданим фільтром система знаходить потрібні запуски без ручного перегляду десятків файлів;
- дані в БД відповідають файлам (відсутні розбіжності типу "в БД одне, у файлі інше").

Випробування В-6. Продуктивність і стабільність серійних запусків

Мета: оцінити, чи система витримує серійні прогони без збоїв пам'яті та неконтрольованих накопичень даних.

Перевіряється:

- чи очищуються проміжні масиви після кожного run;
- чи не росте час виконання "без причин" при довгій серії;
- чи коректно завершується серія, якщо один запуск дав помилку (серія не повинна падати цілком).

Критерії успішності: серія завершується, run'и коректно маркуються, результати підсумовуються.

Підсумок випробувань

У підсумку випробування підтверджують, що ІС «Цифровий двійник» дизельного двигуна працює як керована платформа, придатна до практичного використання: вона виконує одиночні та серійні розрахунки, контролює коректність, коректно обробляє помилки, зберігає результати у структурованому вигляді та забезпечує можливість подальшого аналізу і порівняння запусків. Це є необхідною умовою для позиціонування розробки як інформаційної системи цифрового двійника, а не як набору розрізнених розрахункових скриптів.

3.7 Результат проєктування

У підсумку прийняті проєктні рішення дозволили сформувати цілісну структуру інформаційної системи, яка відповідає логіці «цифрового двійника», а не просто набору розрахункових формул. По-перше, система має обчислювальне ядро, що реалізує модель повного робочого циклу дизельного двигуна і видає не лише окремі кінцеві числа, а й профілі параметрів у функції кута колінчастого вала – тобто ті дані, з якими реально працюють при аналізі робочого процесу.

По-друге, передбачено стандартизоване збереження параметрів і результатів: вхідні дані фіксуються разом із виходом, що робить кожен запуск відтворюваним. Це важливо для серійних експериментів, коли потрібно порівнювати режими або змінювати один параметр і бачити наслідок без плутанини «які саме налаштування були в минулий раз».

По-третє, система одразу орієнтується на аналітичне використання результатів: формуються індикаторні діаграми та супутні графіки, а також таблиці інтегральних показників. У такому вигляді результати не потрібно «витягувати вручну» – вони доступні для пояснення та порівняння одразу після розрахунку.

І нарешті, архітектура спроєктована так, щоб у майбутньому її можна було розширити до формату сервісу: обчислювальне ядро відокремлене від інтерфейсної частини, а результати формуються у структурі, яку можна напряму віддавати через web-API. Тобто перехід від локального прототипу до клієнт–серверного режиму не потребує переписування фізики моделі – потрібно буде лише додати інтерфейсний шар (рис. 3.5).

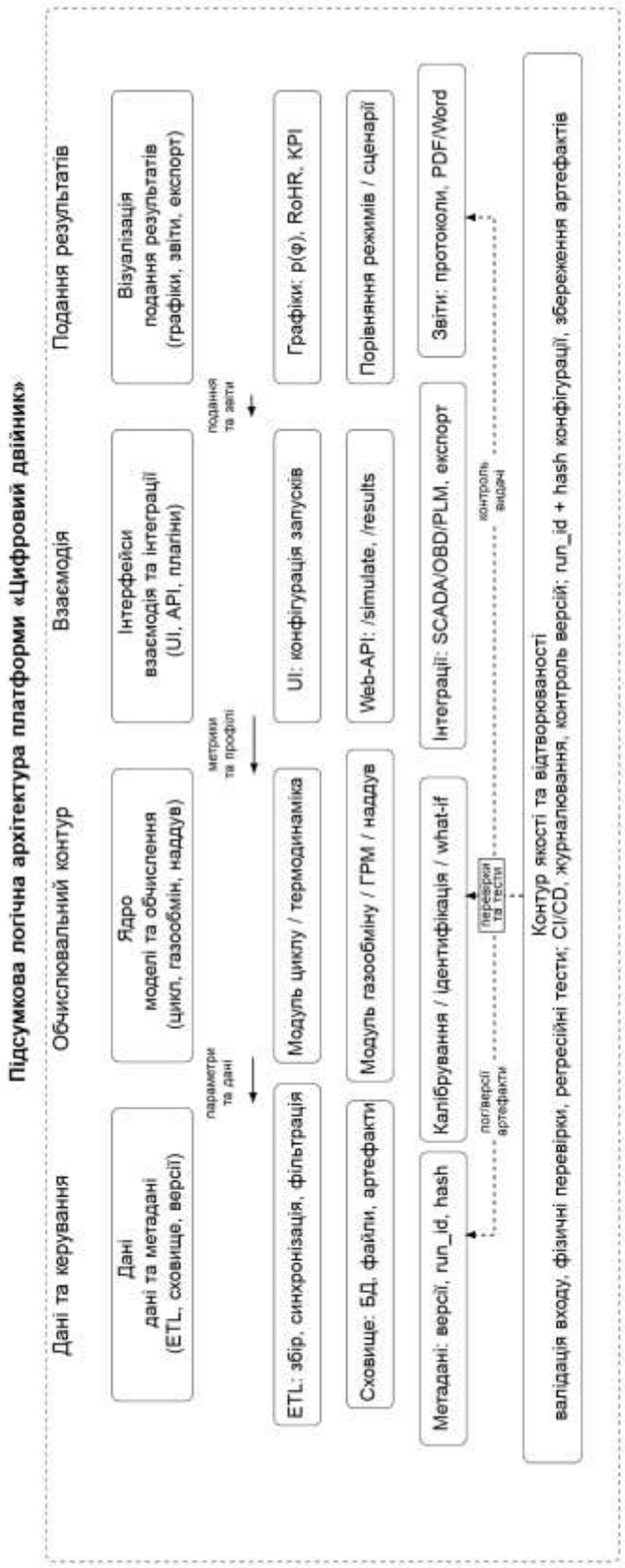


Рисунок 3.5 – Підсумкова логічна архітектура платформи: Core + Data + Interface + Visualization, з перевірками якості та відтворюваністю запусків.

4 Програмна реалізація розрахункової моделі робочого процесу ДВЗ

Розрахункова модель реалізована у вигляді Python-скрипта, який працює як послідовний конвеєр обчислень і забезпечує відтворюваний розрахунок робочого циклу дизельного двигуна. Загальна логіка побудована таким чином, щоб від заданих геометричних і режимних параметрів перейти до похідних характеристик циклу (стискання → згоряння → розширення), а далі – до моделювання впорскування, випаровування, формування закону тепловиділення та інтегрування циклу в кутовій області. У розширеній частині моделі враховано газообмін (впуск/випуск через клапанні перерізи) та теплообмін між газами і стінками (через узагальнений коефіцієнт теплопередачі). Результати подаються у вигляді табличних даних і графіків, що дозволяє контролювати фізичну правдоподібність розрахунку за формою індикаторної діаграми, поведінкою об'єму $V(\varphi)$, профілями частки випареного палива та швидкості тепловиділення.

Візуалізація результатів виконується засобами matplotlib з автоматичним збереженням зображень у каталозі plots/. Такий підхід забезпечує прозорість розрахунків: після кожного запуску формується набір графічних матеріалів (індикаторна діаграма, залежність об'єму від кута, профілі випаровування/впорскування тощо), які можуть бути безпосередньо використані у пояснювальній записці або під час аналізу серій варіантів.

4.1 Використані засоби та структура даних

1) Застосовані бібліотеки та модулі

Програмна реалізація виконана з використанням базових бібліотек Python та типового інженерно-наукового стеку:

- **json** – для зчитування вхідних параметрів двигуна з файлу parameters.json, а також (за потреби) проміжних або узагальнених розрахункових величин із файлу calculation_results.json. Формат JSON обрано як компактний і прозорий для відтворення серій розрахунків за фіксованими наборами параметрів.
- **numpy** – для чисельної роботи з масивами, обчислення похідних і інтегралів (градієнти, трапецієподібне інтегрування), а також формування рівномірних сіток за кутом повороту колінчастого вала \varphi. Зокрема, індикаторна робота визначається чисельно через інтегрування P\,dV за дискретизованими залежностями.
- **math** – для елементарних математичних операцій і тригонометричних перетворень, необхідних у кінематиці кривошипно-шатунного механізму та

політоропних співвідношеннях. Для уникнення помилок одиниць кутова величина явно переводиться з градусів у радіани.

- **pandas** – для формування таблиць результатів у вигляді DataFrame (параметр → одиниці → значення → опис) і подальшого форматованого виводу. Така форма зручна для контролю коректності та для перенесення в звітні документи.
- **matplotlib** – для побудови графіків і збереження їх у файли (PNG) з роздільною здатністю, достатньою для вставлення в текст роботи. Графіки використовуються не лише як ілюстрації, а й як засіб верифікації результатів за характером кривих.

Окрім стандартних бібліотек, використано прикладні модулі проєкту:

- **fuel_data.py** – довідник палив fuel_types (елементний склад, теплота згоряння, цетанове число, густина, в'язкість, енергія активації) та функція add_fuel_type() для додавання нового палива без модифікації базової логіки розрахунку.
- **database.py** – інструменти створення бази даних (create_database) і збереження детальних результатів у структурованому вигляді (save_detailed_calculation_results). Модуль використовується для накопичення серій розрахунків і спрощення подальшого порівняльного аналізу.

2) Вхідні дані та їх організація

Вхідні дані задаються у файлі parameters.json, який містить основні групи параметрів:

- геометричні параметри: діаметр циліндра D, хід поршня S, ступінь стиснення ε, співвідношення радіуса кривошипа до довжини шатуна тощо;
- режимні параметри: частота обертання n, циклова кратність, число циліндрів;
- параметри наддуву та охолодження: ступінь підвищення тиску компресора π_k, показник політропи, температури середовища, охолоджувача, залишкових газів;
- коефіцієнти процесів: коефіцієнт надлишку повітря α, повнота індикаторної діаграми, механічний ККД, коефіцієнти ефективності згоряння/розширення;
- параметри впорскування та горіння: кут початку впорскування φ_{inj}, тривалість впорскування φ_{vp}, допоміжні константи моделі випаровування і тепловиділення.

Параметри палива задаються через довідник fuel_types (наприклад, дизельне паливо, альтернативні суміші тощо). Вибір палива визначає елементний склад C, H, O, S, нижчу теплоту згоряння Q_n, цетанове число та реологічні властивості, що безпосередньо впливають на термохімічний розрахунок, затримку самозаймання та характеристики розпилу.

3) Структура вихідних даних та принцип збереження результатів

Ключові розрахункові величини акумулюються у структурі типу словника results, де для кожного параметра зберігаються:

- числове значення (value);
- одиниці вимірювання (unit);
- короткий опис призначення (description).

Такий підхід забезпечує єдине джерело даних для:

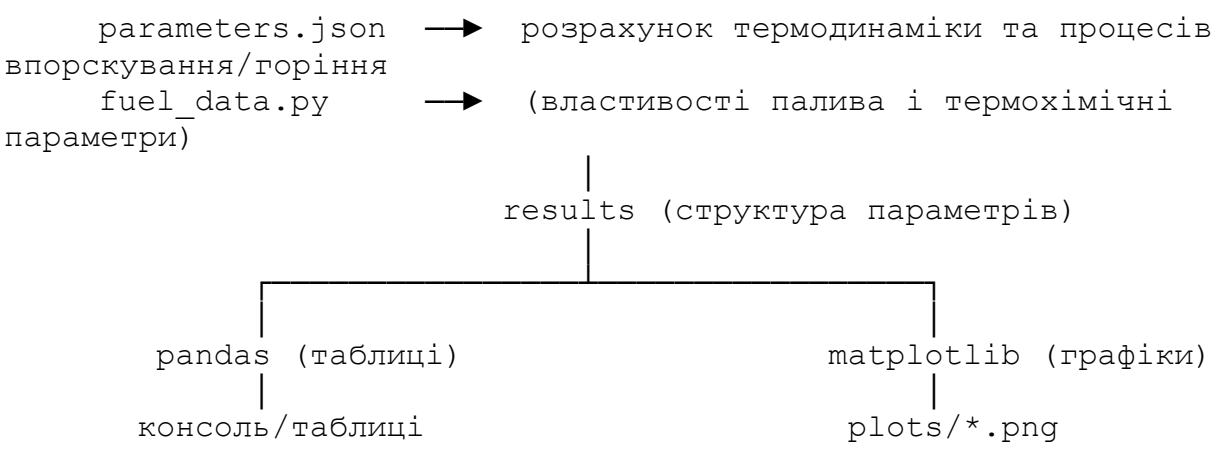
1. формування узагальнювальних таблиць у pandas,
2. запису проміжних величин у JSON-файли,
3. побудови графіків і контрольних залежностей.

Приклад читання параметрів та організації збереження (фрагмент коду):

```
def load_parameters(filepath="parameters.json"):
    with open(filepath, "r", encoding="utf-8") as f:
        return json.load(f)
results = {
    "V_s": {"value": V_s, "unit": "м³", "description": "Робочий об'єм циліндра"},
    "P_s": {"value": P_s, "unit": "кПа", "description": "Тиск повітря перед циліндром"},
    "T_c": {"value": T_c, "unit": "K", "description": "Температура заряду в кінці стиснення"},
}
```

4) Потік даних і формування графічних матеріалів

Потік обробки даних можна подати у вигляді узагальненої схеми:



Каталог для графіків створюється автоматично на початку роботи програми, що гарантує коректне збереження результатів при повторних запусках і при серійному моделюванні:

```
PLOTS_DIR = "plots"
if not os.path.exists(PLOTS_DIR):
    os.makedirs(PLOTS_DIR)
```

Збереження графіків у plots/ дозволяє швидко порівнювати варіанти за формою індикаторної діаграми, величиною пікового тиску, зміною тривалості згоряння та іншими характерними ознаками режиму, не виконуючи ручного перенесення даних між середовищами (рис. 4.1).

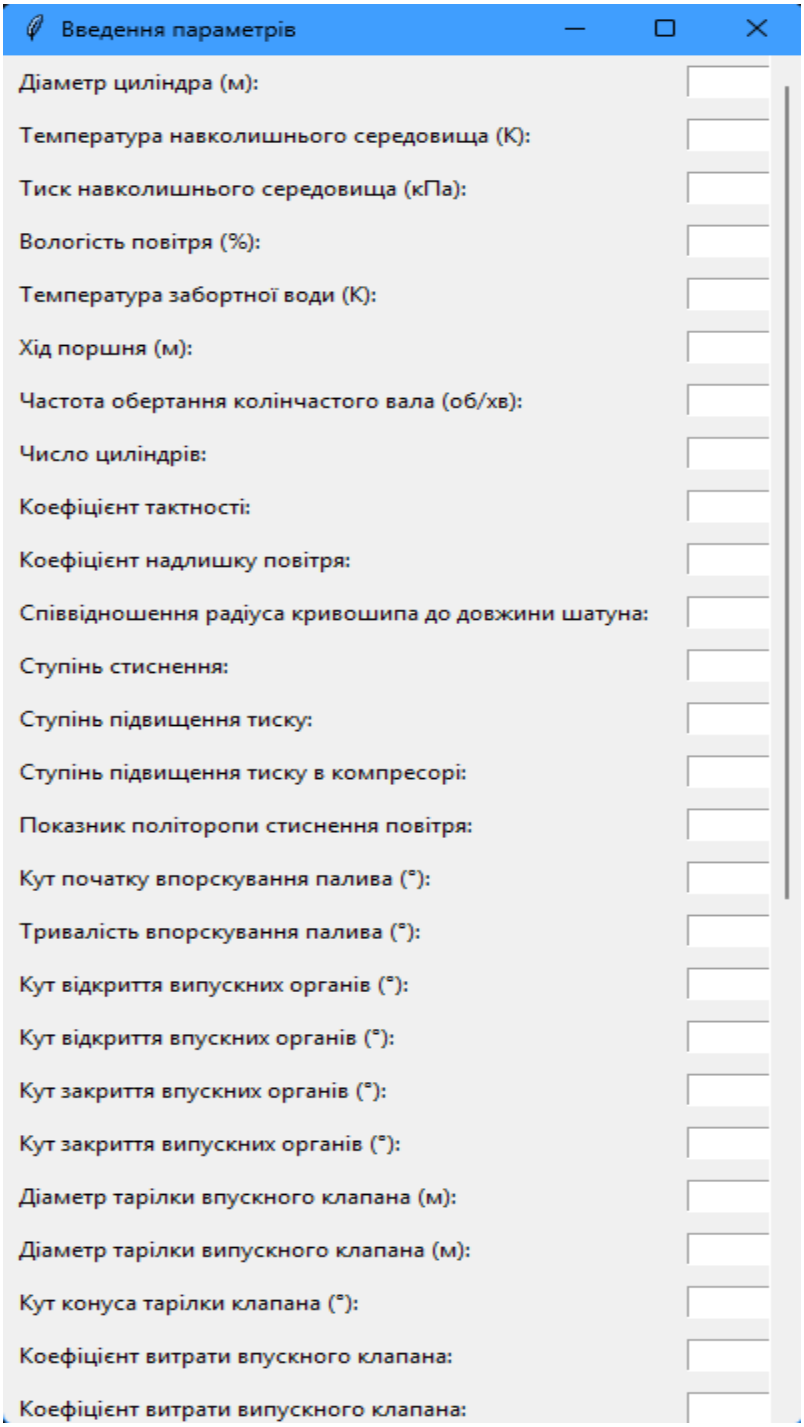


Рисунок 4.1 Введення початкових параметрів

4.2 Підготовка середовища та каталогу результатів

На старті виконання скрипта реалізовано просту, але принципово важливу процедуру підготовки робочого середовища: створення окремого каталогу для графічних ілюстрацій та “табличних рисунків”. Це рішення знімає типову проблему, коли результати візуалізації розкидані по різних папках або перезаписуються без контролю.

Каталог для результатів задається окремою змінною:

```
PLOTS_DIR = "plots"
if not os.path.exists(PLOTS_DIR):
    os.makedirs(PLOTS_DIR)
    print(f"Створено каталог: {PLOTS_DIR}")
```

Логіка роботи така:

- якщо каталог plots/ відсутній, він створюється через os.makedirs(PLOTS_DIR);
- після створення у консоль виводиться службове повідомлення, що полегшує контроль виконання (особливо при запуску на іншому ПК або після очищення робочої директорії);
- усі збережені графіки (індикаторна діаграма, $V(\varphi)$, профілі випаровування тощо), а також зображення таблиць (наприклад, зведена таблиця параметрів, збережена як PNG) записуються саме в цей каталог.

Такий підхід забезпечує **відтворюваність** (кожен запуск формує повний пакет ілюстративних матеріалів) і **уніфікує структуру** підготовки рисунків для розділів пояснювальної записки/дисертації. Крім того, на практиці це суттєво економить час під час оформлення: файли не потрібно “виловлювати” вручну – вони завжди в одному місці та мають стабільні назви.

4.3 Вибір палива та його параметри

Паливо в моделі задається не “жорстко” в тілі скрипта, а через окремий модуль fuel_data.py, де сформовано довідник fuel_types і реалізовано функцію розширення переліку палив. Така організація є методично правильною: термохімічні характеристики та реологічні властивості палива належать до вхідних даних, а не до логіки розрахунку.

1) Отримання даних про паливо з довідника

У модулі зберігається структура типу словника, де ключ – назва (або короткий ідентифікатор) палива, а значення – набір його властивостей. У скрипті реалізовано вивід доступних позицій і вибір палива:

```
from fuel_data import fuel_types, add_fuel_type

print("\nВиберіть вид палива зі списку:")
for fuel in fuel_types.keys():
    print(f"- {fuel}")

selected_fuel = input("\nВведіть вид палива: ")
```

```
if selected_fuel not in fuel_types:
    print("Некоректний вибір палива! Завершення програми.")
    exit()

fuel_data = fuel_types[selected_fuel]
```

Цей фрагмент забезпечує:

- вивід доступних видів палива зі словника fuel_types;
- введення користувачем потрібного типу через input();
- перевірку коректності вибору (якщо ключ відсутній – робота завершується, щоб не допустити “тихих” помилок);
- завантаження властивостей вибраного палива у змінну fuel_data.

2) Контроль і документування властивостей вибраного палива

Після вибору палива скрипт друкує набір його параметрів у консоль. Це використовується як “паспорт режиму”: у разі серії запусків завжди видно, яке паливо реально використовувалось у конкретному розрахунку.

```
print("\n=== Характеристики обраного палива ===")
for param, value in fuel_data.items():
    print(f"{param}: {value}")
```

До типових параметрів палива, які далі входять у термохімічний та кінетичний блоки, належать:

- елементний склад: *C, H, O, S* (масові частки або %, залежно від прийнятого формату);
- нижча теплота згоряння *Qn*;
- цетанове число (як індикатор схильності до самозаймання, важливий у прив’язці до моделі затримки займання);
- густина Density та в’язкість Viscosity (визначають гідродинаміку витікання через сопло, число Вебера, дисперсність крапель і константи випаровування);
- енергія активації Activation_Energy (використовується в експоненційному множнику моделі затримки самозаймання).

3) Розширення довідника палив без зміни основної програми

Окремо передбачено механізм додавання нового палива через add_fuel_type(...). Це принципова перевага при роботі з альтернативними видами палива (синтетичні дизельні фракції, HVO, паливо з переробленої сировини

тощо): змінюються лише довідникові дані, а обчислювальна частина залишається сталою.

Типовий виклик виглядає так:

```
add_fuel_type("НовийТип", {
    "C": 85.0, "H": 13.0, "O": 0.5, "S": 0.0,
    "Q_n": 42000, "Cetane": 50.0,
    "Density": 800, "Viscosity": 4.5,
    "Activation_Energy": 8200
})
```

У результаті модель може бути адаптована до нового палива без модифікації основної логіки розрахунків – змінюється лише інформаційний шар (довідник). Це спрощує серійні дослідження та робить структуру коду ближчою до інженерної практики, де паливо розглядається як набір виміряних/паспортних властивостей, а не як “вшитий” у програму випадок.

4.4 Термодинамічний розрахунок робочого циклу (наддув – наповнення – стискання – згоряння – розширення)

Після зчитування вихідних даних із parameters.json виконується послідовний розрахунок базових параметрів робочого циклу дизельного двигуна. Реалізація побудована так, щоб у коді зберігався прозорий ланцюг: від геометрії та режиму – до параметрів наповнення/стиснення – до розрахунку згоряння й розширення – до інтегральних показників ефективності.

Геометричні та кінематичні параметри

На цьому етапі визначаються геометричні розміри циліндра і кінематика поршневого механізму, які надалі використовуються практично в кожній формулі циклу. До вихідних належать діаметр циліндра D, хід поршня S, частота обертання n, геометричні коефіцієнти механізму.

У скрипті одразу формується робочий об’єм циліндра:

$$V_s = \frac{\pi D^2}{4} S,$$

а також середня швидкість поршня:

$$C_m = \frac{S \cdot n}{30},$$

і максимальна швидкість (за прийнятим емпіричним співвідношенням):

$$C_{m\,max} = 1.57\,C_m.$$

фрагмент реалізації в коді (типово у вигляді прямого обчислення з друком у консоль):

```
V_s = (math.pi * D ** 2 / 4) * S
Cm = (S * n) / 30
Cm_max = 1.57 * Cm
```

Окремо враховується **втратний хід** на газообмін (параметр \psi_a), який коригує дійсний ступінь стискання:

$$\varepsilon = (1 - \psi_a)\varepsilon_0 + \psi_a.$$

У програмі це реалізовано через задану висоту верхньої кромки вікон/органів газообміну h_a та відношення $\psi_a = h_a/S$.

Розрахунок наддуву і параметрів повітря перед циліндром

Блок наддуву задає тиск і температуру заряду після компресора та після охолоджувача. На цьому кроці за ступенем підвищення тиску в компресорі Π_k визначається:

$$P_k = P_0\,\Pi_k,$$

а температура після компресора розраховується за політропним законом:

$$T_k = T_0\,\Pi_k^{\frac{n_k-1}{n_k}}.$$

Далі враховується охолодження наддувочного повітря через ефективність охолоджувача \eta_o та температуру забортної води $T_{зв}$:

$$T_s = T_k - \eta_o\,(T_k - T_{зв}),$$

а тиск перед циліндром отримують через гідравлічний коефіцієнт $\eta_{гх}$:

$$P_s = P_k\,\eta_{гх}.$$

У коді ці переходи виконуються послідовно й однозначно:

```
P_k = P0 * Pi_k
T_k = T0 * (Pi_k ** ((n_k - 1) / n_k))
T_s = T_k - eta_o * (T_k - T_zv)
P_s = P_k * eta_gx
```

Наповнення та стискання (включно із залишковими газами)

Після визначення параметрів перед циліндром задається тиск у кінці наповнення:

$$p_a = \xi_a P_s,$$

де ξ_a – відношення тиску заряду в початку стискання до тиску наддуву.

Залишкові гази враховуються через коефіцієнт γ_r, який визначається з умови змішування й балансів тиску/температури. У коді він розраховується як:

$$gamma_r = (T_s/T_r) * \left(P_r/(epsilon * p_a - P_r)\right)$$

Далі визначається температура заряду на початку стискання з урахуванням підігріву ΔT_s та домішки залишкових газів:

$$T_a = \frac{T_s + \Delta T_s + \gamma_r T_r}{1 + \gamma_r}.$$

Показник політропи стискання n_1 у програмі уточнюється за емпіричною формулою (із залежністю від T_a, та проміжного значення n_1). Після цього визначаються параметри кінця стискання:

$$T_c = T_a \varepsilon^{n_1 - 1}, \qquad P_c = p_a \varepsilon^{n_1}.$$

Розрахунок згоряння (стехіометрія та максимальні параметри)

Згоряння в термодинамічному блоці починається зі стехіометричних співвідношень для 1 кг палива. На основі елементного складу C, H, O, S визначається теоретично необхідна кількість повітря L_0, маса повітря G_0, а також поправка на вологість із переходом до L_a. Дійсна кількість повітря:

$$L = \alpha L_a,$$

де α – коефіцієнт надлишку повітря.

Далі визначаються коефіцієнти молекулярної зміни β_0, β, β_z – вони потрібні для коректного переходу між станами газової суміші при згорянні та подальшому розширенні.

Максимальний тиск циклу визначається через ступінь підвищення тиску λ:

$$P_z = P_c\,\lambda.$$

Температура згоряння T_z обчислюється з використанням теплоємностей (параметри a та b у лінійній апроксимації $C_v = a + bT$) та балансу енергії. У скрипті це реалізовано через допоміжні коефіцієнти A,B,C та явну формулу для T_z (із контрольним значенням T_{z1}, що використовується як початкова оцінка в співвідношенні).

Процес розширення

На етапі розширення визначається ступінь попереднього розширення:

$$\rho = \frac{\beta_z T_z}{\lambda\, T_c},$$

а ступінь подальшого розширення:

$$\delta = \frac{\varepsilon}{\rho}.$$

Показник політропи розширення n_2 задається розрахунковою залежністю (з урахуванням часток згоряння ξ_z, ξ_b, теплоємностей та параметрів робочого тіла). Після цього визначаються параметри кінця розширення:

$$T_b = \frac{T_z}{\delta^{\,n_2-1}}, \qquad P_b = \frac{P_z}{\delta\, n_2}.$$

Інтегральні показники ефективності

Після завершення розрахунку станів циклу формуються узагальнені характеристики: середні тиски, потужності, ККД та питомі витрати. Зокрема:

- середній індикаторний тиск передбачуваного циклу P_i' (аналітичний вираз через $\varepsilon, \lambda, \rho, \delta, n_1, n_2$);
- середній індикаторний тиск фактичного циклу:

$$P_i = \xi P_i',$$

де ξ – коефіцієнт повноти індикаторної діаграми;

- індикаторна потужність N_i та ефективна потужність $N_e = N_i \eta_m$;
- індикаторний і ефективний ККД:

$$\eta_e = \eta_i \, \eta_m;$$

- питомі витрати g_i, g_e і циклова подача g_c.

Для узагальнення даних результати заносяться у словник results зі структурою:

```
    results = {
        "P_z": {"value": P_z, "unit": "кПа", "description":
"Максимальний тиск згоряння"},
        ...
    }
```

Після цього будується зведена таблиця df_all_params, яка:

- виводиться у консоль;
- візуалізується як рисунок таблиці і зберігається у файл plots/table_all_parameters.png.

4.5 Підмодель теплообміну

Для урахування впливу тепловіддачі від робочого тіла до стінок циліндра та охолоджувальної рідини реалізовано окремий набір функцій, що використовуються як у допоміжних оцінках, так і безпосередньо в інтеграторі циклу.

Поточна площа теплообміну

Площа теплообміну змінюється протягом циклу, оскільки змінюється положення поршня та геометрія газового об’єму. Функція F_phi(...) обчислює миттєву поверхню як суму площі днища/кришки та бічної поверхні циліндра з поправкою на поточну висоту газового простору:

```
def F_phi(D, S, epsilon, l_sh, phi_rad):
    return (math.pi * D**2 / 2 +
            math.pi * D * ((S / (epsilon - 1)) +
```

```
                (S / 2) * ((1 - math.cos(phi_rad)) + (l_sh / 2) *
math.sin(phi_rad)**2)))
```

Коефіцієнт тепловіддачі газів до стінки

Коефіцієнт α_g формується як функція діаметра циліндра, тиску та температури в циліндрі; у програмі використано емпіричну залежність виду $\alpha_g \sim p^{0.3}T^{0.75}$, що відображає зростання теплопередачі при підвищенні інтенсивності теплообміну в робочому тілі:

```
    def alpha_g(D, p, T):
        return 1.12e-3 * D**0.25 * (3 / math.sqrt(math.pi * 1e-3))
* p**0.3 * T**0.75
```

Загальний коефіцієнт теплопередачі через стінку

Теплопередача реалізується як послідовність теплових опорів: від газу до стінки, через товщу стінки та від стінки до охолоджувача. Тому загальний коефіцієнт $k_{об}$ задається класичним співвідношенням:

```
    def k_ob(alpha_g, delta_st, lambda_tm, alpha_bv):
        return 1 / (1 / alpha_g + delta_st / lambda_tm + 1 /
alpha_bv)
```

Допоміжні функції dq(…) та Sw(…)

Функція dq(...) використовується для оцінки теплового потоку через поверхню теплообміну при відомій різниці температур:

```
    def dq(F_phi_val, k_ob_val, T_g, T_b_val, bn=1):
        return k_ob_val * F_phi_val * (T_g - T_b_val) / bn
```

Функція Sw(...) вводить тепловий вплив у вигляді члена, який застосовується в інтеграторі циклу як складова рівнянь зміни тиску/температури (узгоджено з прийнятою формою енергетичного балансу):

```
    def Sw(k_ob, T, T_b, Hmp_val, D, k, n):
        numerator = D / 2 + Hmp_val
        denominator = D * Hmp_val
        return (2 / 3) * (k - 1) * (numerator / denominator) *
(k_ob * (T - T_b) / n)
```

Температура охолоджувальної рідини

Температура охолоджувача в моделі визначається як середня між температурою на вході та на виході, що задаються у parameters.json:

$$T_{\text{в}} = \frac{T_{\text{вх}} + T_{\text{вих}}}{2}.$$

У коді це реалізовано окремою функцією $T_v(...)$, а самі значення T_{vkh}, T_{vy} беруться з параметрів. Така постановка відповідає інженерній практиці: в більшості розрахункових схем реальний тепловий режим сорочки охолодження описують середньою температурою теплоносія при відомих граничних значеннях на вході/виході.

4.6 Модель упорскування, випаровування та тепловиділення

Підмодель «упорскування–випаровування–тепловиділення» призначена для відтворення фізично обґрунтованого перебігу підготовки палива до згоряння та формування закону тепловиділення у циліндрі. Вона побудована як послідовність взаємопов'язаних блоків: (1) кінематика об'єму циліндра; (2) визначення стану робочого тіла в момент початку впорскування; (3) оцінка характеристик струменя (факела) та краплинного спектра; (4) побудова профілю впорскування й випаровування; (5) формування дискретних залежностей $dx(\varphi)$ і $x(\varphi)$, що надалі використовуються у інтеграторі циклу.

4.6.1 Кінематика об'єму циліндра $V(\varphi)$ та похідної $dV/d\varphi$

Миттєвий об'єм циліндра обчислюється з урахуванням геометрії КШМ і співвідношення $\lambda = r/l$ (радіус кривошипа до довжини шатуна). У програмі це реалізовано функціями calc_volume(φ) та calc_dvolume(φ):

```
    def calc_volume(phi_deg: float, V_s: float, epsilon: float,
lambda_ratio: float) -> float:
        phi = math.radians(phi_deg)
        clearance = V_s / (epsilon - 1)
        return clearance + 0.5 * V_s * (1 - math.cos(phi) + 0.5 *
lambda_ratio * math.sin(phi)**2)

    def calc_dvolume(phi_deg: float, V_s: float, lambda_ratio:
float) -> float:
        phi = math.radians(phi_deg)
        return 0.5 * V_s * (math.sin(phi) + 0.5 * lambda_ratio *
math.sin(2 * phi))
```

Функція calc_volume задає $V(\varphi)$ як суму «кліренсного» об'єму $V_c = V_s/(\varepsilon - 1)$ та змінної складової, що описує переміщення поршня. Функція calc_dvolume використовується для отримання похідної об'єму за кутом і надалі

потрібна при інтегруванні роботи циклу та при формуванні членів рівнянь зміни стану через $d(\ln V)/d$.

Для контролю реалізації формується графік $V(\varphi)$ на інтервалі 0–720° з позначенням кута початку впорскування φ_{inj}. Рисунок автоматично зберігається у файл plots/cylinder_volume.png, що дає змогу візуально перевірити коректність прив'язки фаз упорскування до кінематики циліндра (зокрема, відповідність початку впорскування ділянці стискання) (рис. 4.2).

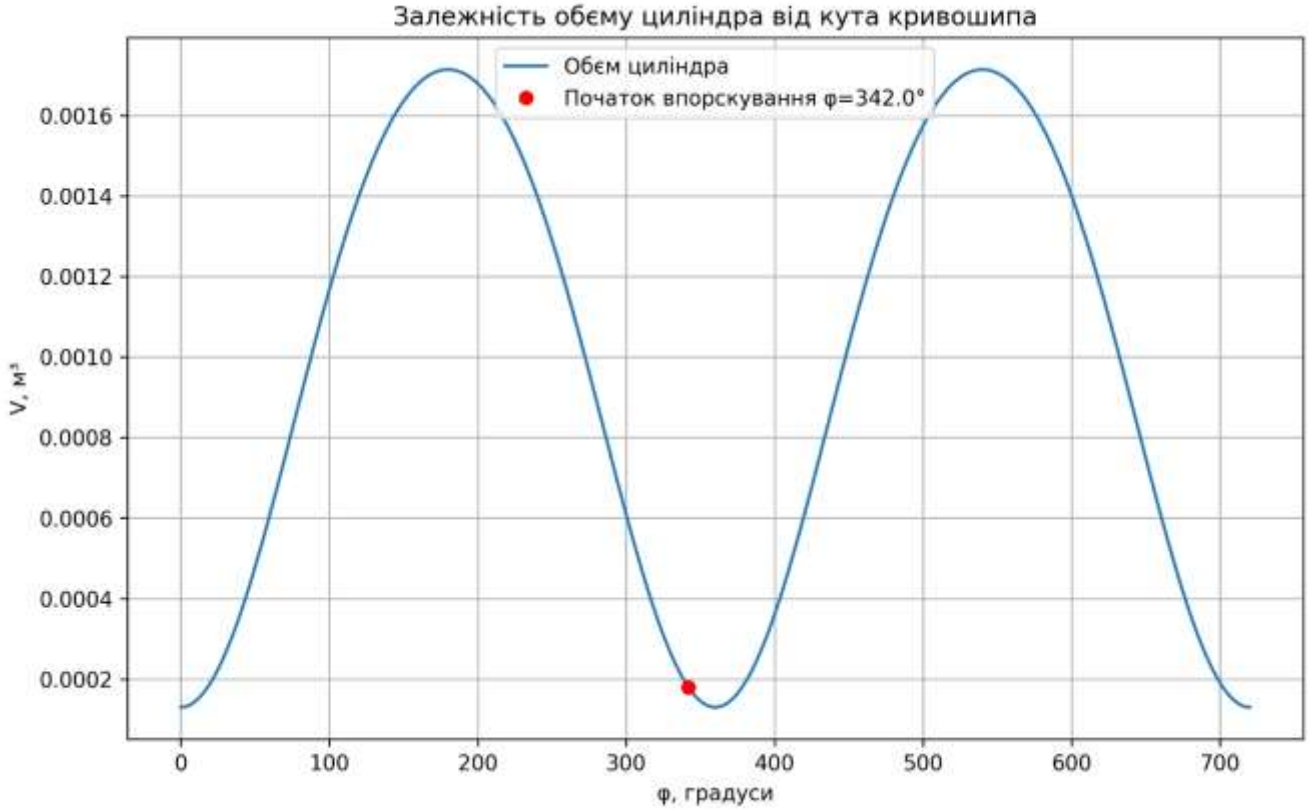


Рисунок 4.2 Залежність об'єму циліндра від кута кривошипа

4.6.2 Параметри в момент початку впорскування та затримка самозаймання

Ключовими для подальших розрахунків є тиск і температура у циліндрі в момент початку впорскування: p_j та T_j. Тиск визначається як результат політропного стискання від умов перед циліндром (тиск наддувного повітря перед циліндром P_s) до поточного об'єму $V(\varphi_{inj})$:

```
    def calc_pressure_injection(P_s, V_s, epsilon, phi_inj,
lambda_ratio, V_c, n1):
        V_inj = calc_volume(phi_inj, V_s, epsilon, lambda_ratio)
        return P_s * ((V_c + V_s) / V_inj) ** n1
```

Температура в точці впорскування визначається з рівняння стану через обчислені p_j, $V(\varphi_{inj})$ газову сталу суміші та масу свіжого заряду:

```
    def calc_temperature_injection(p_j, V_s, phi_inj, epsilon,
lambda_ratio, R_cm, mass_charge):
        V_inj = calc_volume(phi_inj, V_s, epsilon, lambda_ratio)
        return (p_j * V_inj) / (R_cm * mass_charge) / 10**6
```

Затримка самозаймання φ_i оцінюється експоненційною залежністю типу Арреніуса з урахуванням енергії активації палива E_a, температури T_j, тиску p_j та кінематичного параметра (середньої швидкості поршня C_m). У коді затримка формується як кутова величина (у градусах), що зручно для безпосереднього використання у фазових розрахунках:

```
def calc_ignition_delay(E_a, T_j, p_j, Cm):
    term1 = (1.0 / (8.314 * T_j) - (1.0 / 17190.0))
    term2 = (21.2 / (1e-2 * p_j - 12.4)) ** 0.63
    exponent = E_a * (term1 + term2)
    return (0.36 + 0.22 * Cm) * math.exp(exponent)
```

На підставі φ_i визначається момент займання:

$$\varphi_v = \varphi_{inj} + \varphi_i,$$

який надалі використовується як «точка запуску» тепловиділення в інтеграторі.

4.6.3 Характеристики факела: $We, \gamma, L_{ct}, d_{32}, b_{it}, \tau_v, \tau_i$

Блок розпилювання та факела потрібний для переходу від «геометрії впорскування» до «швидкості підготовки палива», тобто до оцінки випаровування. У моделі використано набір характерних емпіричних залежностей, які пов’язують умови витікання з параметрами струменя:

- число Вебера We як критерій співвідношення інерційних і поверхневих сил краплі/струменя;
- кут розкриття факела γ, що визначає інтенсивність змішування та ширину області взаємодії з повітрям;
- довжина вільного прольоту факела L_{ct} – ділянка розвитку струменя до суттєвого гальмування/втрати імпульсу;
- середній поверхневий діаметр крапель d_{32} як узагальнений показник «тонкості» розпилювання;

- константа випаровування b_{it} та час випаровування τ_v.

Окремо формується часова затримка займання у секундах:

$$\tau_i = \frac{\varphi_i}{6n},$$

а також оцінка тривалості горіння (через співвідношення для φ_z), що дозволяє визначити кут кінця горіння φ_{kr}. Для прив'язки фаз використовуються:

$$\varphi_{kv} = \varphi_{inj} + \varphi_{vp}, \qquad \varphi_{kr} = \varphi_v + \varphi_z,$$

де φ_{vp} – тривалість упорскування.

Практичний сенс цього блоку полягає в тому, що саме d_{32} та b_{it} задають швидкість зменшення невипареної частки палива, а γ і L_{ct} впливають на інтенсивність перемішування та можливість контакту факела зі стінкою (що, в свою чергу, коригує ефективне випаровування).

4.6.4 Профілі впорскування $\sigma(\varphi), \sigma_1(\varphi), \sigma_2(\varphi)$та похідні $d\sigma/d\tau$

Подача палива в часі/по куту задається безрозмірними функціями $\sigma(\varphi)$, які описують накопичену частку поданої порції. У програмі використано двоскладову форму $\sigma_1 + \sigma_2$ із різними показниковими параметрами, що дозволяє відтворити нерівномірний характер подачі (наростання–основна подача). Для аналізу швидкості подачі додатково реалізовані похідні dsigma1, dsigma2, а також сумарна dsigma.

У результаті формується комбінований графік характеристик упорскування з кривими:

$$\sigma(\varphi), \quad \frac{d\sigma_1}{d\tau}, \quad \frac{d\sigma_2}{d\tau}, \quad \frac{d\sigma}{d\tau},$$

який зберігається у plots/fuel_injection_characteristics.png. На графіку також відмічаються вертикальні лінії характерних кутів (початок упорскування, займання, кінець упорскування), що дозволяє перевірити узгодженість фаз (рис. 4.3).

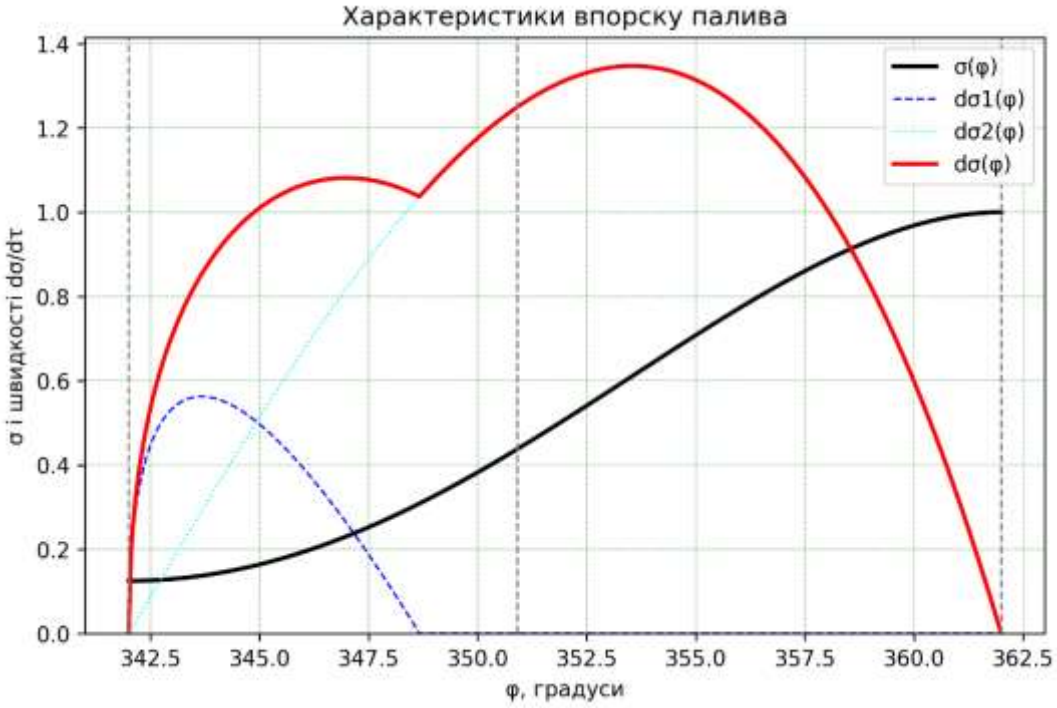


Рисунок 4.3 Характеристика впорскування палива

4.6.5 Частка випареного палива $\sigma_i(\varphi)$ та швидкість випаровування

Після задання профілю впорскування модель переходить до оцінки випаровування. Частка випареного палива $\sigma_i(\varphi)$ у програмі формується інтегруванням швидкісної функції $d_{sigma_n}(\varphi)$, яка враховує:

- поточний час від початку впорскування $\tau(\varphi) = \left(\varphi - \varphi_{inj}\right)/(6n)$;
- «ступінь випаровування» через допоміжні функції $B(\varphi)$ та $B_1(\varphi)$, що зменшують невипарену частку з плином часу;
- корекцію після контакту факела зі стінкою через множник $\xi(\varphi)$ (ефективність випаровування на стінці).

Накопичена частка $\sigma_i(\varphi)$ реалізована як дискретна сума по куту (фактично – інтеграл по часу через перетворення градуси↔секунди). Для візуального контролю автоматично будуються й зберігаються два графіки:

- plots/fuel_evaporated_fraction.png – $\sigma_i(\varphi)$, тобто накопичена частка випареного палива;
- plots/evaporation_rate.png – $d\sigma_i/d\varphi$, що показує інтенсивність випаровування на різних ділянках процесу.

Ці графіки важливі з практичної точки зору: за ними добре видно, чи «встигає» паливо випаруватися до моменту активного тепловиділення, а також чи не з’являються нефізичні стрибки швидкості випаровування (рис. 4.4-4.5).

Рисунок 4.4 Швидкість випаровування палива

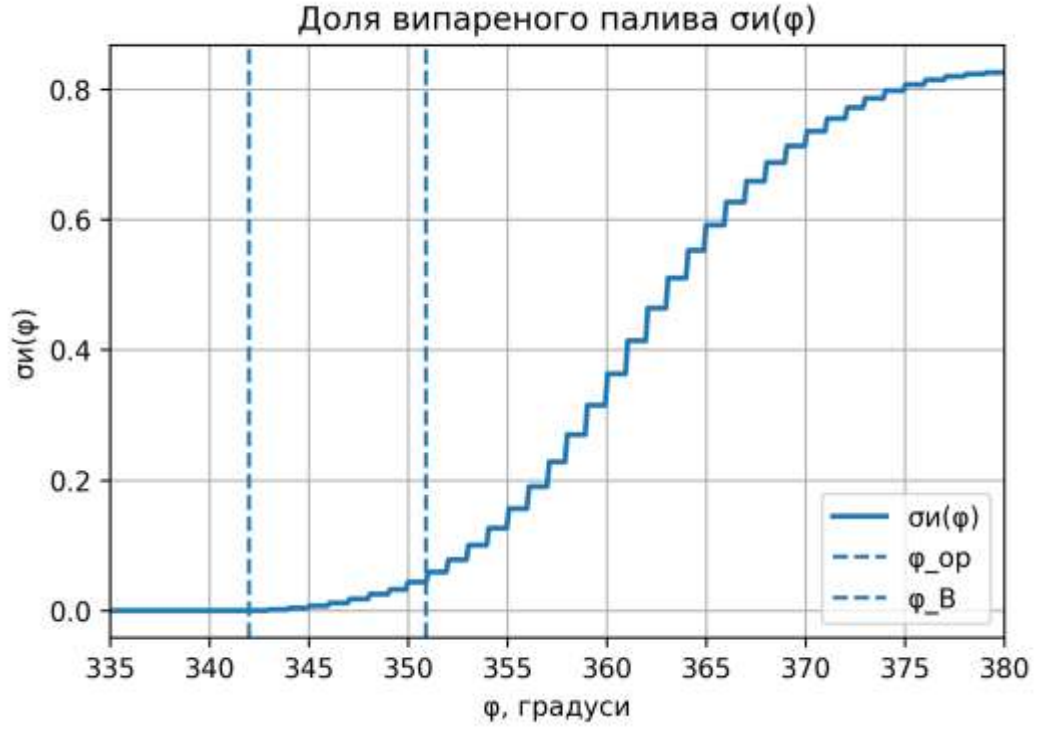


Рисунок 4.5 Доля випареного палива

4.6.6 Тепловиділення: формування дискретних масивів $dx(\varphi)$ та $x(\varphi)$

Тепловиділення у моделі описується через частку згорілого палива x(\varphi) та її приріст $dx(\varphi)$. Важливо, що ці величини формуються не як суто

аналітична функція, а як дискретні масиви з кроком по куту, що природно узгоджується з подальшим покроковим інтегруванням циклу.

Розрахунок розбитий на дві фази:

1. Ділянка впорскування (φ_{inj} ... φ_{kv}).

Тут накопичення згорілої частки пов'язане з надходженням та випаровуванням палива, а також із фактом, що до моменту займання φ_v активне згоряння ще не розвивається. У програмі застосовано допоміжну змінну x_0, яка виконує роль «підготовленої» частки, та логіку перемикання між різними джерельними членами залежно від співвідношення x_0 і σ_i.

2. Ділянка розвиненого горіння/догоряння (φ_{kv} ... φ_{kr}).

Для цієї фази розраховується ефективний коефіцієнт надлишку повітря в зоні горіння $\alpha_T(\varphi)$ через функцію $\xi_B(\varphi)$ та далі – приріст dx у вигляді добутку x(1-x)-типу з поправкою на недожог $\Delta\tau$. Така форма забезпечує природне «самозатухання» процесу при наближенні x до кінцевого рівня.

Після завершення обох фаз формується підсумковий графік plots/dx_and_x_functions.png, де на одній діаграмі показано:

- $x(\varphi)$ – інтегральну частку згорілого палива;
- $dx(\varphi)$ – інтенсивність тепловиділення (у коді для зручності масштабована множником).

Цей графік є ключовим для перевірки моделі: він показує, чи має тепловиділення реалістичний фронт наростання, максимум та спад, а також чи узгоджується його тривалість із φ_{kv} і φ_{kr} (рис. 4.6).

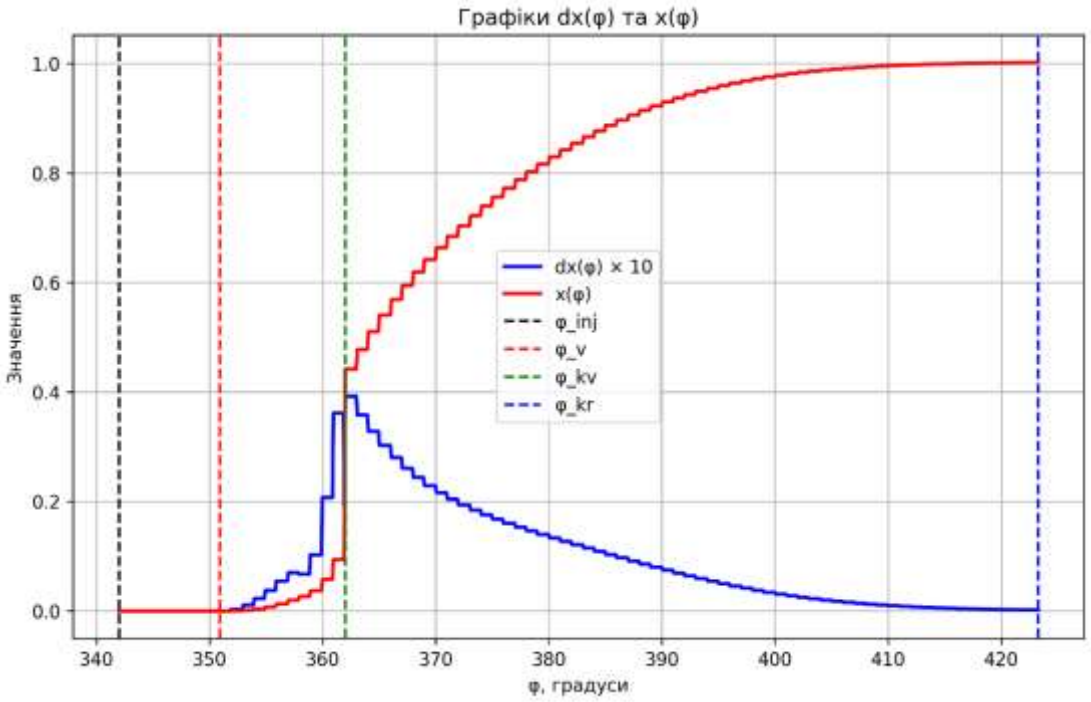


Рисунок 4.6 Характеристики тепловиділення

4.7 Газообмін і інтегрування робочого циклу

Розрахунок робочого циклу в моделі виконується для 4-тактного дизельного двигуна на інтервалі 720° з урахуванням газообміну (впуск/випуск), теплообміну та підведення теплоти через закон тепловиділення $dx(\varphi)$. Центральним елементом є інтегратор cycle_integration(...), який покроково оновлює стан робочого тіла за кутом колінчастого вала.

4.7.1 Пропускна здатність клапанів: $\mu F(\varphi)$

Газообмін описується через ефективні проходні перерізи клапанів, що залежать від фази газорозподілу. Для випуску та впуску реалізовано функції:

- muFv(φ) – ефективна пропускна здатність випускного клапана;
- muFvp(φ) – ефективна пропускна здатність впускного клапана.

Вони є кусочно-заданими залежностями: на ділянках відкривання/закривання використовується косинусний закон (м'яке наростання/спад), а на плато – сталий рівень. Такий підхід відтворює типову інженерну апроксимацію реальних діаграм підйому клапана без надмірного ускладнення моделі.

4.7.2 Швидкісні функції потоку та тискові відношення

Швидкість витікання/припливу через клапани розраховується функціями:

- w23(P, Bt, T) – для випуску (потік з циліндра до тракту/турбіни);
- w12(P, Bs, T) – для впуску (потік з ресивера/впуску до циліндра).

Тут Bt та Bs – відношення тисків, що визначають режим течії. У формулах враховано показники адіабати для повітря та відпрацьованих газів (k, k_r). Логіка реалізації дво-режимна: для певних відношень тиску застосовується вираз для субкритичної течії, для інших – для режиму з «обмеженням» швидкості (критичний характер потоку).

4.7.3 Інтегрування циклу

cycle_integration(...)

Функція cycle_integration(...) виконує інтегрування на інтервалі 720° із кроком d\varphi (у програмі – dphi=1.0). На кожному кроці розраховуються:

1. Кінематика та об'єм: $V(\varphi)$, $dV/d\varphi$, допоміжні геометричні величини (для теплообміну).
2. Теплообмін: коефіцієнти α_g, k_{ob}, члени впливу теплообміну на зміну параметрів (через Sw).

3. Газообмін: масові прирости впуску dG_n та випуску dG_x, накопичення $G_n(\varphi), G_x(\varphi)$.
4. Тепловиділення: значення $dx(\varphi)$ і $x(\varphi)$ із підмоделі згоряння.
5. Оновлення стану: тиск $P(\varphi)$ та температура $T(\varphi)$ оновлюються покроково, причому вираз для dP задається кусочно залежно від ділянки циклу (до займання, під час згоряння, під час відкриття органів газообміну тощо).

   Результатом інтегрування є масиви:

$$P(\varphi),\ T(\varphi),\ M(\varphi),\ G_n(\varphi),\ G_x(\varphi),$$

а також похідні припливу/витікання dG_n, dG_x на кожному кроці. Ці масиви є основою для подальшого розрахунку індикаторної роботи та енергетичних показників за «інтегрованим» циклом.

4.7.4 Індикаторна діаграма

За отриманим масивом $P(\varphi)$ формується індикаторна діаграма у координатах P–\varphi та зберігається у файл plots/indicator_diagram.png (рис. 4.7). Практично цей рисунок виконує дві функції:

- ілюстративну – відображення характеру зміни тиску протягом циклу;
- контрольну – швидка перевірка узгодженості газообміну, теплообміну та тепловиділення (наявність нефізичних розривів, надмірних стрибків або «полиць» тиску за фазами).

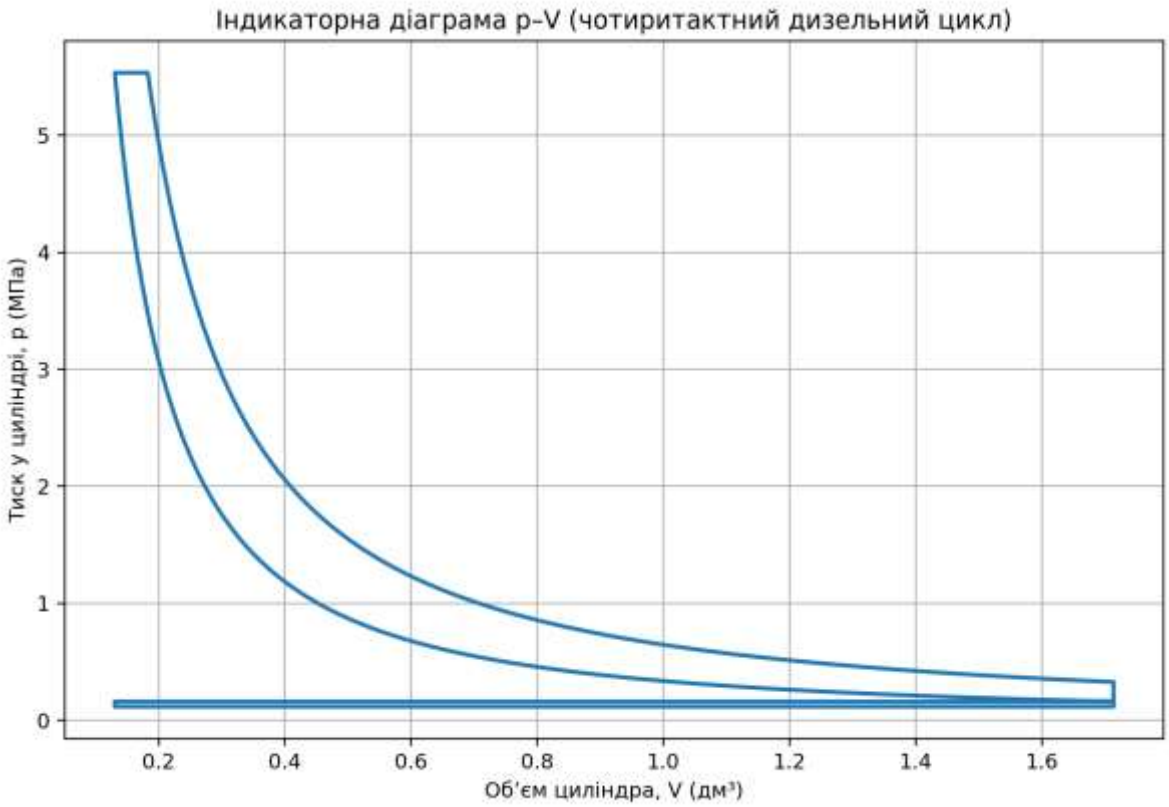


Рисунок 4.7 Індикаторна діаграма

4.8 Розрахунок енергетичних показників за інтегрованим циклом

Після завершення покрокового інтегрування робочого циклу (отримання масивів P(\varphi) та відповідних V(\varphi)) виконується окремий блок постобробки, метою якого є визначення інтегральних енергетичних показників уже не за «розрахунковим» термодинамічним циклом, а безпосередньо за інтегрованими даними моделі.

Основою розрахунку є індикаторна робота за цикл як контурний інтеграл:

$$L_i = \oint P \; dV,$$

який у програмі реалізується чисельно: спочатку формується масив об'ємів $V(\varphi)$, далі обчислюється похідна $dV/d\varphi$ (через np.gradient), після чого інтегрується добуток $P(\varphi) \cdot dV/d$ методом трапецій.

Фрагмент реалізації має вигляд:

```
phi_vals = np.array(results['phi'])          # φ [град]
```

```
    P_vals   = np.array(results['P'])           # P [Па]
    V_vals   = np.array([calc_volume(phi, V_s, epsilon,
lambda_ratio) for phi in phi_vals])  # V [м³]

    dphi   = np.mean(np.diff(phi_vals))
    dV_vals = np.gradient(V_vals, dphi)

    dLi_vals = P_vals * dV_vals
    Li = np.trapezoid(dLi_vals, dx=dphi) / 1000  # [кДж]
```

Такий підхід дає можливість отримати L_i узгоджено з реальною формою кривої $P(\varphi)$, що сформована інтегратором із урахуванням газообміну, теплообміну та дискретного тепловиділення.

На основі L_i визначається середній індикаторний тиск:

$$p_i = \frac{L_i}{V_s},$$

де V_s – робочий об’єм циліндра. Далі середній ефективний тиск обчислюється через механічний ККД:

$$p_e = p_i \cdot \eta_m.$$

Окремо визначаються індикаторний та ефективний ККД. У реалізації використовується співвідношення, яке зв’язує виконану індикаторну роботу з теплотою, що підводиться паливом у циклі:

$$\eta_i = \frac{p_i V_s}{Q_n\, g_c}, \qquad \eta_e = \eta_i \cdot \eta_m,$$

де Q_n – нижча теплота згоряння палива, g_c – циклова подача палива. Така постановка дає безпосередній «енергетичний баланс» для інтегрованого циклу.

Питомі витрати визначаються у традиційному вигляді:

$$g_i = \frac{3600}{Q_n\, \eta_i}, \qquad g_e = \frac{g_i}{\eta_m},$$

що дозволяє отримати величини в одиницях $\text{г}/(\text{кВт}\cdot\text{год})$ у прийнятій інженерній формі подання.

Потужності розраховуються на основі середнього індикаторного тиску та геометрії циліндра. У коді реалізовано класичну оцінку індикаторної потужності (з урахуванням режиму та коефіцієнта циклів) і перехід до ефективної потужності множенням на η_m:

$$N_e = N_i \cdot \eta_m.$$

Додатково формується перевірочне значення циклової подачі g_c за розрахованими g_e, N_e та частотою обертання, що є корисним контролем узгодженості одиниць і формул:

$$g_{c,\,check} = \frac{g_e\, N_e}{60\, z\, n}.$$

Результати блоку постобробки збираються у таблицю pandas.DataFrame, яка друкується в консоль у компактному вигляді, що спрощує подальший перенос чисел у текстові таблиці роботи або додатки.

4.9 Формування вихідних матеріалів для звіту

Фінальна частина скрипта виконує роль «вивідного контуру» моделі: підтверджує формування всіх графічних матеріалів, що створюються в процесі розрахунків, та забезпечує їх уніфіковане збереження у каталозі plots/. Це важливо з погляду відтворюваності: один запуск моделі автоматично генерує повний набір ілюстрацій, потрібних для пояснення структури розрахунку, аналізу фаз процесу та демонстрації результатів інтегрованого циклу.

Після виконання розрахунків скрипт формує перелік збережених файлів і виводить службове повідомлення з підрахунком імен, наприклад:

```
saved_plots = [
    "table_all_parameters.png",
    "indicator_diagram.png",
    "cylinder_volume.png",
    "fuel_evaporated_fraction.png",
    "evaporation_rate.png",
    "fuel_injection_characteristics.png",
    "dx_and_x_functions.png"
]

print(f"✅ Успішно збережено {len(saved_plots)} графіків у каталозі '{PLOTS_DIR}/':")
for plot_name in saved_plots:
```

```
print(f"    {plot_name}")
```

Поточний набір ілюстративних матеріалів включає:

- table_all_parameters.png – табличний рисунок з повним переліком розрахованих термодинамічних параметрів;
- indicator_diagram.png – індикаторна діаграма за інтегрованим циклом у координатах $P(\varphi)$;
- cylinder_volume.png – залежність $V(\varphi)$ з позначенням початку впорскування;
- fuel_evaporated_fraction.png – частка випареного паливаσ$_i(\varphi)$;
- evaporation_rate.png – швидкість випаровування $d\sigma_i/d$;
- fuel_injection_characteristics.png – комбінована діаграма профілю подачі та швидкісних характеристик $\sigma(\varphi), d\sigma/d$;
- dx_and_x_functions.png – графіки $dx(\varphi)$ та $x(\varphi)$ як основа закону тепловиділення.

Таким чином програмна реалізація охоплює повний замкнений цикл розрахунку: від завантаження вхідних параметрів і вибору властивостей палива до термодинамічної оцінки процесів, моделювання упорскування–випаровування–згоряння, інтегрування робочого циклу з урахуванням газообміну та теплообміну, а також автоматичного формування комплекту графічних матеріалів, придатних для включення у текст дисертації та додатки.

5 Техніко-економічне обґрунтування розробки програмної платформи «Цифровий двійник» дизельного двигуна

Розроблена програмна платформа «Цифровий двійник» дизельного двигуна реалізує розрахунок повного робочого циклу з підмоделями упорскування–випаровування–тепловиділення, теплообміну та газообміну, а також автоматичним формуванням графіків і таблиць результатів. Практична цінність рішення полягає у переході від фрагментарних «ручних» розрахунків (розрізнені файли, різні формати даних, повторне перенесення параметрів) до відтворюваного параметричного пайплайна: вхідні дані → розрахунок → інтегрування → метрики → візуалізація. Це безпосередньо зменшує витрати робочого часу інженера/дослідника, підвищує дисципліну даних і знижує ризик помилок на етапі серійних варіантних досліджень.

Нижче наведено розрахунок вартості створення та оцінювання економічного ефекту у числовому вигляді. Для визначеності використано реалістичні припущення для інженерно-дослідницької роботи (ставка, трудомісткість, кількість сценаріїв на рік). У разі уточнення фактичних значень (ставки, реальної трудомісткості, статистики запусків) формули залишаються тими самими, а перерахунок виконується прямою підстановкою.

5.1 Розрахунок вартості створення (витрат на розробку)

Загальна вартість розробки оцінюється як:

$$C_{dev} = Z + C_{ovh} + C_{eq} + C_{sw} + C_{other},$$

де Z – фонд оплати праці; C_{ovh} – накладні витрати; C_{eq} – витрати на використання обладнання; C_{sw} – витрати на ПЗ; C_{other} – інші витрати (документація, тестування, оформлення результатів, резерв).

В таблиці 5.1 представлені етапи робіт.

1) Фонд оплати праці Z

Приймається погодинна ставка виконавця (інженер/аспірант/науковий співробітник з Python-розрахунками):

$$r = 350 \text{ грн/год.}$$

Таблиця 5.1 Трудомісткість робіт доцільно деталізувати за етапами, щоб уникнути «абстрактної» оцінки:

Етап робіт	Зміст	Трудомісткість, год
Аналіз вимог і постановка задач	структура параметрів, набір вихідних/результатних величин, сценарії	20
Реалізація термодинамічного ядра	наддув, наповнення, стискання, згоряння, розширення, метрики	60
Підмоделі упорскування/випаровування/тепловиділення	функції $\sigma(\varphi), \sigma_i(\varphi), dx(\varphi), x(\varphi)$	40
Інтегратор циклу з газообміном і теплообміном	масовий баланс, клапани, теплообмінні члени, стабілізація	25
Тестування, контроль адекватності, підготовка графіків	валідація на контрольних режимах, «кутові» перевірки	15
Документація (опис, приклади, шаблони JSON)	опис структури даних, перелік виходів, ілюстрації	10
Разом		160

Тоді фонд оплати праці:

$$Z = T \cdot r = 160 \cdot 350 = 56\,000 \text{ грн.}$$

2) Накладні витрати C_{ovh}

Накладні витрати в навчально-наукових розробках часто задають як частку від Z. Приймається норматив:

$$k_{ovh} = 0.25.$$
$$C_{ovh} = k_{ovh} \cdot Z = 0.25 \cdot 56\,000 = 14\,000 \text{ грн.}$$

3) Витрати на використання обладнання C_{eq}

Оцінка проводиться через амортизацію, пропорційну часу використання комп’ютера в межах проєкту:

$$C_{eq} = \frac{P_{eq}}{N_{am}} \cdot \frac{T_{use}}{T_{year}},$$

де P_{eq} – вартість робочого ноутбука/ПК, N_{am} – строк амортизації, T_{use} – час використання в проєкті, T_{year} – річний фонд часу.

Приймається:

- $P_{eq} = 50\,000$ грн,
- $N_{am} = 3$ роки,
- $T_{use} = 160$ год,
- $T_{year} = 1800$ год.

Тоді:

$$C_{eq} = \frac{50\,000}{3} \cdot \frac{160}{1800} = 16\,666.67 \cdot 0.08889 \approx 1\,481.48 \text{ грн.}$$

4) Витрати на ПЗ C_{sw}

Платформа реалізована на Python та бібліотеках з відкритою ліцензією (NumPy, Pandas, Matplotlib), тому:

$$C_{sw} \approx 0 \text{ грн.}$$

5) Інші витрати C_{other}

Доцільно передбачити резерв на дрібні витрати (узгодження формату звітних матеріалів, додаткові тестові запуски, оформлення рисунків). Приймається 10% від Z:

$$C_{other} = 0.10 \cdot Z = 0.10 \cdot 56\,000 = 5\,600 \text{ грн.}$$

Підсумок вартості створення

$$C_{dev} = 56\,000 + 14\,000 + 1\,481.48 + 0 + 5\,600 = 77\,081.48 \text{ грн.}$$

Округлено:

$$\boxed{C_{dev} \approx 77.1 \text{ тис. грн}}.$$

5.2 Оцінювання економічного ефекту від впровадження

Економічний ефект проявляється переважно в економії робочого часу при серійних варіантних розрахунках (режими, наддув, параметри палива, закони тепловиділення, кути впорскування тощо). Оцінка виконується через різницю трудомісткості одного сценарію «до» і «після» впровадження.

Річний ефект:

$$E_{year} = (t_{old} - t_{new}) \cdot N_{cases} \cdot c_{hour},$$

де t_{old} – час на один сценарій без платформи, t_{new} – час із платформою, N_{cases} – кількість сценаріїв на рік, c_{hour} – вартість години роботи.

Прийняті реалістичні значення:

- ручний/напівручний підхід (підготовка, перерахунки, перенесення даних, побудова графіків):
$$t_{old} = 6.0 \text{ год/сценарій}$$
- автоматизований підхід (редагування parameters.json, запуск, аналіз готових графіків і таблиць):
$$t_{new} = 1.5 \text{ год/сценарій}$$
- кількість сценаріїв на рік (дослідження + навчальні/методичні кейси):
$$N_{cases} = 120 \text{ сценаріїв/рік } (\approx 10/\text{міс.})$$
- вартість години:
$$c_{hour} = 350 \text{ грн/год.}$$

Тоді економія часу на один сценарій:

$$\Delta t = t_{old} - t_{new} = 6.0 - 1.5 = 4.5 \text{ год.}$$

Річний ефект:

$$E_{year} = 4.5 \cdot 120 \cdot 350 = 540 \cdot 350 = 189\,000 \text{ грн/рік.}$$

Округлено:

$$\boxed{E_{year} \approx 189 \text{ тис. грн/рік}}.$$

Окремо варто зазначити, що частина ефекту не «потрапляє» в цю формулу напряму, але є практично важливою: (а) менше помилок через єдину структуру JSON і автоматичне формування графіків; (б) повторюваність сценаріїв; (в) швидший попередній відбір режимів для експерименту (економія часу стенду і витратних матеріалів на невдалі/неперспективні точки).

5.3 Показники економічної ефективності

1) Строк окупності
Найпростіша оцінка – строк окупності:

$$T_{pb} = \frac{C_{dev}}{E_{year}} = \frac{77\,081.48}{189\,000} \approx 0.408 \text{ року}.$$

У місяцях:

$$0.408 \cdot 12 \approx 4.9 \text{ міс}.$$

Отже:

$$\boxed{T_{pb} \approx 5 \text{ місяців}}.$$

2) Дисконтована оцінка (NPV) на горизонті 3 років
Для більш реалістичного представлення можна врахувати:

- ставку дисконту r=0.15 (типове значення для оцінок із ризиками та альтернативною вартістю ресурсів),
- щорічні витрати на підтримку (дрібні правки, оновлення залежностей, контрольні тестові запуски): наприклад, 40 год/рік.

$$C_{maint} = 40 \cdot 350 = 14\,000 \text{ грн/рік}.$$

Тоді чистий ефект на рік:

$$E_{net} = E_{year} - C_{maint} = 189\,000 - 14\,000 = 175\,000 \text{ грн/рік}.$$

NPV за 3 роки:

$$NPV = \sum_{t=1}^{3} \frac{E_{net}}{(1+r)^t} - C_{dev}.$$

Підстановка:

$$NPV = \frac{175\,000}{1.15} + \frac{175\,000}{1.15^2} + \frac{175\,000}{1.15^3} - 77\,081.48 \approx 322\,483 \text{ грн.}$$

Отже:

$$\boxed{NPV_{3y} \approx 322 \text{ тис. грн} > 0},$$

що підтверджує економічну доцільність розробки навіть за консервативних припущень і з урахуванням щорічного супроводу.

Висновок до розділу 5

Розрахунки показують, що створення програмної платформи «Цифровий двійник» із вартістю розробки близько 77 тис. грн забезпечує річний економічний ефект від економії часу на варіантних розрахунках на рівні ≈189 тис. грн/рік, а орієнтовний строк окупності становить близько 5 місяців. На горизонті трьох років дисконтована оцінка дає позитивний NPV, що додатково підтверджує ефективність рішення. Поза прямою економією часу, платформа підвищує відтворюваність розрахунків, стандартизує структуру даних і прискорює відбір перспективних режимів для експериментальної перевірки, формуючи практичну основу для подальшого розширення функціоналу «цифрового двійника».

6 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

6.1 Аналіз потенційно небезпечних і шкідливих факторів у приміщенні

Під час розробки та експлуатації програмної платформи «Цифровий двійник» дизельного двигуна основні ризики пов’язані не з технологічним обладнанням дизеля, а з умовами праці розробника/користувача ПЗ: роботою за персональним комп’ютером, периферією, мережевим обладнанням, джерелами безперебійного живлення, а також з організацією робочого місця та режимом праці й відпочинку. Вимоги до безпечної роботи при використанні ЕОМ визначаються нормативними актами з охорони праці та відповідними інструкціями підприємства/закладу освіти.

До типових небезпечних і шкідливих факторів для такого робочого місця належать:

- ураження електричним струмом при пошкодженні ізоляції, неправильному підключенні, відсутності заземлення, перевантаженні мережі;
- пожежна небезпека через кабельні траси, блоки живлення, UPS, папір, пластики;
- недостатнє/надмірне освітлення, відблиски на екрані, мерехтіння;
- шум від системних блоків, серверів, вентиляторів, кондиціонерів;
- несприятливий мікроклімат (перегрів приміщення, сухість повітря);
- ергономічні ризики (неправильна поза, статичні навантаження, перенапруження зору);
- психофізіологічні фактори (високе розумове навантаження, дефіцит часу, монотонність).

6.1.1 Електробезпека

Робоче місце розробника ПЗ використовує електрообладнання класу офісної техніки: ПК/ноутбук, монітор(и), роутер/комутатор, зарядні пристрої, інколи – UPS та окремий сервер/робоча станція для розрахунків. Основні небезпеки: дотик до струмоведучих частин у разі пошкодження корпусу/кабелю, нагрівання контактів у перевантажених подовжувачах, «плаваюче» заземлення.

Заходи забезпечення електробезпеки:

1. живлення лише від справної мережі з автоматичними вимикачами; бажано – із застосуванням УЗО/дифавтомата для групи розеток ПК-місць;
2. наявність захисного заземлення для стаціонарних ПК/серверів та металевих корпусів;

3. заборона використання пошкоджених кабелів, розхитаних розеток, «скруток», перевантажених трійників;
4. розміщення кабелів так, щоб виключити перетирання, натяг, защемлення меблями;
5. регламентні огляди та інструктаж користувачів; виконання вимог інструкції з охорони праці при експлуатації ЕОМ.

6.1.2 Пожежна безпека

Приміщення з комп'ютерною технікою належить до пожежонебезпечних через наявність горючих матеріалів (кабельна ізоляція, пластик, папір, меблі), а джерелом займання може стати коротке замикання, перегрів блока живлення, несправний подовжувач, порушення правил експлуатації електроприладів.

Основні протипожежні заходи:

- дотримання Правил пожежної безпеки для будівель/приміщень;
- забезпечення приміщення первинними засобами пожежогасіння (вогнегасник порошковий або CO_2 відповідної місткості), вільного доступу до них;
- заборона блокування евакуаційних виходів, проходів; наявність плану евакуації;
- вимкнення обладнання після завершення роботи (або переведення в безпечний режим), контроль стану UPS/подовжувачів;
- заборона використання саморобних обігрівачів, несправних зарядних пристроїв.

6.1.3 Освітлення

Якість освітлення критично впливає на безпеку праці за ПК: при недостатній освітленості зростає втома та ризик помилок, при надмірній – з'являються відблиски, сльозотеча, головний біль. Нормування освітлення для робочих місць у приміщеннях виконується за будівельними нормами та стандартами з освітлення.

Організаційні вимоги:

- робочий стіл розміщувати так, щоб уникати прямих відблисків від вікна/світильників на екрані;
- застосовувати світильники з розсіяним світлом (панелі/лінійні LED) та, за потреби, локальне освітлення без «жорсткої» тіні;
- підтримувати чистоту плафонів і поверхонь (це прямо впливає на освітленість через коефіцієнт запасу).

6.1.4 Захист від шуму та вібрації

Для офісу/лабораторії характерний низькочастотний шум від вентиляторів ПК, серверів, кондиціонерів. Вібрації зазвичай не є визначальним фактором, але можуть виникати від серверних стійок або систем охолодження.

Заходи:

- використання тихих вентиляторів, справних підшипників, очищення систем охолодження;
- винесення серверів/обчислювальних вузлів у окрему зону або застосування шумоізоляційних шаф;
- контроль технічного стану кондиціонерів.

6.1.5 Виробничий мікроклімат

Розрахунки робочого циклу дизеля (особливо параметричні серії) можуть тривати довго, спричиняючи підвищене тепловиділення від ПК/серверів. Несприятливий мікроклімат (висока температура, сухе повітря) підсилює втому й знижує працездатність.

Рекомендовано:

- підтримувати температуру та вологість на рівнях, прийнятних для офісної роботи;
- застосовувати вентиляцію/кондиціонування, але уникати прямого потоку холодного повітря на людину;
- робити провітрювання та технологічні перерви.

6.1.6 Організація робочого місця

Ергономіка для розробника ПЗ є ключовою: неправильна посадка та висота монітора призводять до перенапруження м’язів спини/шиї, кистей, а також до втоми очей.

Вимоги та рекомендації:

- регульований стілець із підтримкою попереку;
- верхня межа екрана – приблизно на рівні очей, відстань до монітора – комфортна для читання без нахилу голови;
- клавіатура і миша – на висоті, що не викликає «зламу» кисті;
- перерви та зміна виду діяльності згідно з інструкціями/регламентами для роботи з ЕОМ.

6.2 Розрахунок освітлення (метод світлового потоку) для приміщення 9 × 4,5 м

Вихідні дані

Розміри приміщення: $A = 9$ м, $B = 4.5$ м

Площа:

$$S = A \cdot B = 9 \cdot 4.5 = 40.5 \text{ м}^2.$$

Нормована освітленість робочої зони: $E = 300$ лк

Коефіцієнт нерівномірності: z=1.1

Коефіцієнт запасу: k=1.5

Світильник: LED-панель, $\Phi = 3600$ лм

Коефіцієнт використання світлового потоку: $\eta = 0.55$

Визначення кількості світильників

$$N = \frac{E \cdot S \cdot z \cdot k}{\Phi \cdot \eta}.$$

Чисельник:

$$E \cdot S \cdot z \cdot k = 300 \cdot 40.5 \cdot 1.1 \cdot 1.5.$$
$$300 \cdot 40.5 = 12150,$$
$$12150 \cdot 1.1 = 13365,$$
$$13365 \cdot 1.5 = 20047.5.$$

Знаменник:

$$\Phi \cdot \eta = 3600 \cdot 0.55 = 1980.$$

Отже:

$$N = \frac{20047.5}{1980} = 10.125.$$

Приймається (округлення у більшу сторону):

$$\boxed{N = 11 \text{ світильників}}.$$

Перевірка фактичної освітленості

$$E_{\text{ф}} = \frac{N \cdot \Ф \cdot \eta}{S \cdot z \cdot k}.$$

Чисельник:

$$N \cdot \Ф \cdot \eta = \ 11 \cdot 3600 \cdot 0.55 = 11 \cdot 1980 = 21780.$$

Знаменник:

$$S \cdot z \cdot k \ = \ 40.5 \cdot 1.1 \cdot 1.5.$$
$$40.5 \cdot 1.1 = 44.55,$$
$$44.55 \cdot 1.5 = 66.825.$$

Тоді:

$$E_{\text{ф}} = \frac{21780}{66.825} \approx 325.9 \text{ лк } (\approx 326 \text{ лк}).$$

(Для контролю: при N=10 було б $E_{\text{ф}} \approx 296.3 \text{ лк} < 300 \text{ лк}$, тому 11 є мінімально достатнім).

Висновок

Для приміщення 9×4.5 м$(S = 40.5 \text{ м}^2)$ за прийнятими коефіцієнтами та світильниками $\Ф = 3600$ лм необхідно встановити $\boxed{11}$ LED-панелей. Забезпечується фактична освітленість $\boxed{E_{\text{ф}} \approx 326 \text{ лк}}$, що відповідає та перевищує норму 300 лк.

6.3 Безпека в надзвичайних ситуаціях

Для робочого місця розробника/оператора ПЗ «Цифровий двійник» дизельного двигуна найбільш імовірні надзвичайні ситуації: пожежа в приміщенні, задимлення від короткого замикання, відключення електроенергії, аварії інженерних мереж, а також (в умовах України) сигнал «Повітряна тривога». Загальні вимоги організації цивільного захисту визначаються Кодексом цивільного захисту України.

Основні заходи підготовки:

- наявність інструкцій дій персоналу при пожежі та евакуації, призначення відповідальних осіб;
- оповіщення (сирена/локальна система), доступність шляхів евакуації;

- справні вогнегасники та знання персоналом правил їх застосування;
- резервне копіювання даних, щоб мінімізувати втрати результатів моделювання (це також елемент «стійкості» ІС).

Дії при пожежі/задимленні:

1. припинити роботу, за можливості – знеструмити обладнання без ризику для життя;
2. повідомити відповідальних та викликати 101;
3. локалізувати загоряння первинним засобом пожежогасіння, якщо це безпечно;
4. організовано евакуюватися за планом, не користуватися ліфтами.

Дії при раптовому зникненні електроживлення:

- перевести систему в безпечний стан (UPS дає короткий час на коректне завершення), зберегти результати;
- від’єднати чутливе обладнання при нестабільній напрузі;
- після відновлення живлення виконати перевірку працездатності.

Дії при «Повітряній тривозі» (для закладів освіти/офісів в Україні):

- припинити роботу, зберегти дані, за можливості вимкнути живлення робочого місця;
- перейти до найближчого укриття згідно з планом цивільного захисту організації;
- діяти за вказівками відповідальних осіб до сигналу відбою.

6.4 Охорона навколишнього середовища

Розробка програмної платформи не створює прямих викидів чи відходів виробничого типу, проте впливає на довкілля опосередковано – через споживання електроенергії, використання електронного обладнання та утворення електронних відходів.

Заходи екологічної відповідальності в межах проєкту:

- оптимізація обчислень (ефективні алгоритми, профілювання коду, кешування результатів), що зменшує енергоспоживання серверів/ПК;
- застосування енергоощадних режимів (sleep/hibernate, вимкнення моніторів);
- централізоване зберігання та контроль версій для зменшення дублювання даних;
- передача відпрацьованої електроніки (блоки живлення, АКБ UPS, монітори) на утилізацію/переробку через ліцензовані канали.

Висновки

У магістерській роботі виконано дослідження технологій та розроблено концепцію програмної платформи «Цифровий двійник» дизельного двигуна на базі моделі повного робочого циклу. У межах поставленої мети проаналізовано підходи до математичного моделювання дизельного циклу та сучасні інструменти реалізації цифрових двійників, визначено вимоги до інформаційної системи та запропоновано архітектуру програмного комплексу з можливістю розширення функціоналу.

У роботі сформовано вимоги до вхідних даних (геометрія циліндро-поршневої групи, параметри газообміну та наддуву, характеристики палива і впорскування, початкові термодинамічні умови), а також до вихідних результатів моделювання (індикаторні діаграми $p(\varphi)$, залежності $T(\varphi), V(\varphi)$, інтегральні енергетичні та паливно-економічні показники $p_i, p_e, N_i, N_e, \eta_i, \eta_e, g_i, g_e$ тощо). Реалізовано розрахунковий конвеєр моделювання, що поєднує термодинамічний розрахунок, оцінювання затримки займання, підмоделі розпилення/випаровування/тепловиділення та чисельну інтеграцію параметрів циклу по куту повороту колінчастого вала φ з формуванням графічних і табличних результатів.

Розглянуто питання охорони праці, пожежної та електробезпеки для робочого місця розробника/оператора ПЗ, а також наведено приклад розрахунку освітлення методом світлового потоку з отриманням кількості світильників, достатньої для забезпечення нормативної освітленості. Запропоновано організаційні та технічні заходи щодо зниження ризиків при роботі з комп’ютерною технікою і дій у надзвичайних ситуаціях.

Практичне значення роботи полягає в можливості використання створеної платформи як навчально-дослідницького інструмента для аналізу впливу параметрів двигуна, наддуву та паливоподачі на показники робочого циклу, а також як основи для подальшого розвитку цифрового двійника з інтерактивною візуалізацією, інтеграцією з базами даних експериментів і підключенням зовнішніх клієнтів через API.

Список використаної літератури

1. Grieves M. Digital Twin: Manufacturing Excellence through Virtual Factory Replication : white paper. – 2014.
2. Tao F., Qi Q., Liu A., Kusiak A. Data-driven smart manufacturing // *Journal of Manufacturing Systems*. – 2018. – Vol. 48. – P. 157–169.
3. Lu Y., Liu C., Wang K. I.-K., Huang H., Xu X. Digital Twin-driven smart manufacturing: Connotation, reference model, applications and research issues // *Robotics and Computer-Integrated Manufacturing*. – 2020. – Vol. 61.
4. Kritzinger W., Karner M., Traar G., Henjes J., Sihn W. Digital Twin in manufacturing: A categorical literature review // *IFAC-PapersOnLine*. – 2018. – Vol. 51, No. 11. – P. 1016–1022.
5. ISO 23247-1:2021. Digital twin framework for manufacturing – Part 1: Overview and general principles. – Geneva : International Organization for Standardization, 2021.
6. ISO 23247-2:2021. Digital twin framework for manufacturing – Part 2: Reference architecture. – Geneva : International Organization for Standardization, 2021.
7. Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities / INCOSE. – 4th ed. – Hoboken : Wiley, 2015.
8. NASA Systems Engineering Handbook. – NASA SP-2016-6105 Rev2. – Washington, DC : NASA, 2016.
9. Heywood J. B. Internal Combustion Engine Fundamentals. – 2nd ed. – New York : McGraw-Hill Education, 2018.
10. Stone R. Introduction to Internal Combustion Engines. – 4th ed. – London : Palgrave Macmillan, 2012.
11. Guzzella L., Onder C. H. Introduction to Modeling and Control of Internal Combustion Engine Systems. – Berlin : Springer, 2010.
12. Ferguson C. R., Kirkpatrick A. T. Internal Combustion Engines: Applied Thermosciences. – 3rd ed. – Hoboken : Wiley, 2015.
13. Taylor C. F. The Internal-Combustion Engine in Theory and Practice. Vol. 1–2. – Cambridge : MIT Press, 1985.
14. Watson N., Janota M. S. Turbocharging the Internal Combustion Engine. – London : Macmillan, 1982.
15. Blair G. P. Design and Simulation of Four-Stroke Engines. – Warrendale : SAE International, 1999.

16. Ghojel J. Review of the development and applications of the Wiebe function: A tribute to the contribution of Ivan Wiebe to engine combustion analysis // *Applied Thermal Engineering*. – 2010. – Vol. 30, No. 13. – P. 1709–1722.
17. Turns S. R. An Introduction to Combustion: Concepts and Applications. – 3rd ed. – New York : McGraw-Hill, 2012.
18. Glassman I., Yetter R. A., Glumac N. Combustion. – 5th ed. – Amsterdam : Academic Press, 2014.
19. Hardenberg H. O., Hase F. W. An empirical formula for computing the pressure rise delay of a fuel from its cetane number and from relevant parameters of direct-injection diesel engines // *SAE Technical Paper*. – 1979.
20. Woschni G. A universally applicable equation for the instantaneous heat transfer coefficient in the internal combustion engine // *SAE Technical Paper* 670931. – Warrendale : SAE International, 1967.
21. Hohenberg G. Advanced approaches for heat transfer calculations // *SAE Technical Paper* 790825. – Warrendale : SAE International, 1979.
22. Annand W. J. D. Heat transfer in the cylinders of reciprocating internal combustion engines // *Proceedings of the Institution of Mechanical Engineers*. – 1963. – Vol. 177, No. 1. – P. 973–996.
23. Diesel Engine Management: Systems and Components / Bosch. – 4th ed. – Stuttgart : Robert Bosch GmbH, 2014.
24. Hiroyasu H., Arai M. Structures of fuel sprays in diesel engines // *SAE Technical Paper* 902077. – Warrendale : SAE International, 1990.
25. Reitz R. D., Diwakar R. Structure of high-pressure fuel sprays // *SAE Technical Paper*. – 1987.
26. Siebers D. L. Scaling liquid-phase fuel penetration in diesel sprays based on mixing-limited vaporization // *SAE Technical Paper* 1999-01-0528. – Warrendale : SAE International, 1999.
27. Bishop I. N. The effects of design variables on friction and economy. – Warrendale : SAE International, (рік видання уточнити за вашим джерелом).
28. Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P. Numerical Recipes: The Art of Scientific Computing. – 3rd ed. – Cambridge : Cambridge University Press, 2007.
29. Virtanen P., Gommers R., Oliphant T. E., et al. SciPy 1.0: Fundamental algorithms for scientific computing in Python // *Nature Methods*. – 2020. – Vol. 17. – P. 261–272.
30. Harris C. R., Millman K. J., van der Walt S. J., et al. Array programming with NumPy // *Nature*. – 2020. – Vol. 585. – P. 357–362.

31. Hunter J. D. Matplotlib: A 2D graphics environment // *Computing in Science & Engineering*. – 2007. – Vol. 9, No. 3. – P. 90–95.
32. McKinney W. Data structures for statistical computing in Python // *Proceedings of the 9th Python in Science Conference (SciPy 2010)*. – 2010.
33. Sommerville I. Software Engineering. – 10th ed. – Boston : Pearson, 2015.
34. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner’s Approach. – 9th ed. – New York : McGraw-Hill Education, 2019.
35. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Boston : Addison-Wesley, 1994.
36. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. – Geneva : ISO, 2011.
37. ISO/IEC/IEEE 29148:2018. Systems and software engineering – Life cycle processes – Requirements engineering. – Geneva : ISO, 2018.
38. Жидецький В.Ц. Основи охорони праці. Підручник. [Текст] / Жидецький В.Ц. Львів: Афіша, 2002. – 320 с.
39. Жидецький В.Ц. Охорона праці користувачів комп’ютерів . Навчальний посібник. – Вид.2-ге доп. [Текст] / Жидецький В.Ц Львів: Афіша, 2000. – 176 с.
40. ДНАОП 0.00 – 1.31 – 99. Правила охорони праці під час експлуатації електронно-обчислювальних машин. [Текст] / К., 1999. – 111 с.
41. М.П. Купчик Охорона праці. Лабораторний практикум [Текст] / М.П. Купчик, М.П. Гандзюк, І.Ф. Гандзюк, І.Ф.Степанеь, В.Н Вендиханський, А.М. Литвиненко, О.В Іваненко. – К.: Основа, 1998. - 224 с.
42. Тубальцев А.М. Виробниче освітлення та його розрахунок: навчальний посібник. . [Текст] / Тубальцев А.М. –Миколаїв: УДМТУ, 2001. – 84 с.
43. Щедролосєв О.В. Основи охорони праці. Лабораторний практикум. [Текст] / Щедролосєв О.В., Моісеєнко Л.Л., Яглицький Ю.К. – Херсон:Айлант, 2005. – 84.
44. Кодекс цивільного захисту України від 02.10.2012 № 5403-VI (чинна редакція від 12.09.2025). URL: https://zakon.rada.gov.ua/laws/show/5403-17.
45. Постанова Кабінету Міністрів України від 24.03.2004 № 368 «Про затвердження Порядку класифікації надзвичайних ситуацій техногенного та природного характеру за їх рівнями» (з змінами). URL: https://zakon.rada.gov.ua/laws/show/368-2004-%D0%BF.
46. Постанова Кабінету Міністрів України від 09.01.2014 № 11 «Про затвердження Положення про єдину державну систему цивільного захисту»

(чинна редакція від 16.04.2025). URL: https://zakon.rada.gov.ua/laws/show/11-2014-%D0%BF.
47. Закон України «Про правовий режим воєнного стану» від 12.05.2015 № 389-VIII (з змінами). URL: https://zakon.rada.gov.ua/laws/show/389-19.
48. Офіційний сайт Державної служби України з надзвичайних ситуацій (ДСНС України). URL: https://www.dsns.gov.ua/.
49. Розпорядження Кабінету Міністрів України від 24.12.2024 про план заходів цивільного захисту на 2025 рік.
50. Матеріали щодо розробки Стратегії розвитку ДСНС до 2030 року (2025 р., за підтримки ПРООН). URL: https://hromady.org/.
51. Звіти МАГАТЕ щодо ЗАЕС (2025 р.).

ДОДАТОК А – технічне завдання

Технічне завдання на розробку програмної платформи «Цифровий двійник» дизельного двигуна на базі моделі повного робочого циклу (Python)

А.1 Загальні відомості

Найменування інформаційної системи: програмна платформа «Цифровий двійник» дизельного двигуна на базі моделі повного робочого циклу.

Умовна назва програмного продукту: *DieselCycleUa* (робоча назва).

Призначення: виконання розрахунку повного робочого циклу дизельного двигуна з отриманням інтегральних і фазових показників; побудова індикаторних діаграм та допоміжних характеристик; збереження результатів у машинозчитуваних форматах для повторного аналізу.

Підстава для розробки: виконання магістерської роботи за темою:

«Дослідження технології та розробка програмної платформи “Цифровий двійник” дизельного двигуна на базі моделі повного робочого циклу».

А.2 Мета розробки

Метою розробки є створення програмної платформи, що забезпечує:

- введення вихідних параметрів двигуна, умов роботи та параметрів паливоподачі;
- розрахунок термодинамічних параметрів циклу (наповнення – стискання – згоряння – розширення) і допоміжних підмоделей (упорскування, розпил, випаровування, тепловиділення, газообмін);
- формування вихідних показників (тиски, температури, ККД, витрати, потужність, індикаторна робота);
- візуалізацію результатів у вигляді графіків та табличних форм;
- збереження результатів у файли для відтворення розрахунків та подальшого аналізу.

А.3 Об’єкт автоматизації та межі системи

Об’єкт автоматизації: робочий цикл дизельного двигуна як обчислювальна модель (цифровий аналог фізичних процесів у циліндрі).

Система охоплює: параметризацію двигуна та середовища, розрахунок циклу, формування метрик, візуалізацію, збереження результатів.

Система не охоплює (обмеження):

- керування реальним двигуном у реальному часі;

- пряме підключення до реальних датчиків і збір телеметрії;
- CFD-моделювання (3D газодинаміка/горіння);
- автоматизовану оптимізацію параметрів (допускається як перспективний напрям розширення).

А.4 Функціональні вимоги

А.4.1 Введення та підготовка даних

Платформа повинна забезпечувати:

- завантаження параметрів з конфігураційного файлу parameters.json;
- роботу з довідником палива (вибір наявного типу, додавання нового типу);
- контроль коректності критичних параметрів (наявність ключів, типи даних, допустимі діапазони значень).

А.4.2 Розрахунок робочого циклу

Платформа повинна забезпечувати:

- обчислення геометричних та кінематичних величин, параметрів наддуву, коефіцієнтів газообміну;
- розрахунок стискання та визначення параметрів у характерних точках циклу (включно з тиском/температурою);
- підмодель упорскування/розпилу/випаровування;
- підмодель тепловиділення з поділом на фази горіння;
- інтегрування за кутом колінчастого вала для формування масивів $P(\varphi), T(\varphi), V(\varphi)$ та супутніх змінних;
- формування індикаторної роботи та енергетичних показників за інтегрованим циклом.

А.4.3 Формування результатів

Платформа повинна забезпечувати:

- формування структурованого набору результатів із полями: значення, одиниця виміру, опис;
- формування табличного подання результатів;
- побудову графіків: індикаторна діаграма $P(\varphi)$, об’єм $V(\varphi)$, частка/швидкість випаровування, тепловиділення та інші допоміжні залежності.

А.4.4 Збереження та експорт

Платформа повинна забезпечувати:

- збереження інтегральних результатів у файл calculation_results.json;

- можливість експорту табличних даних у CSV (за потреби);
- підтримку збереження розширених профілів (масивів по φ) у структурі вихідних даних.

А.5 Вимоги до якості, надійності та коректності

- Програма не повинна аварійно завершуватися при некритичних помилках введення; критичні помилки мають супроводжуватись поясненням причини (відсутній ключ JSON, некоректний тип, недопустимий діапазон).
- Передбачаються мінімальні контролі фізичної узгодженості (заборона від'ємних мас/об'ємів/температур; контроль адекватності тисків і температур для заданого режиму).
- Результати повинні бути відтворюваними при однакових вхідних даних.

А.6 Вимоги до програмних засобів та середовища

- Мова програмування: Python 3.x
- Базові модулі: json, math, os
- Масиви та чисельні операції: numpy
- Табличні структури: pandas
- Візуалізація: matplotlib
- (опційно) GUI: tkinter
- (опційно) база даних: SQLite через модуль database.py.

А.7 Вхідні та вихідні дані

Вхідні дані:

- parameters.json: геометрія двигуна, частота обертання, коефіцієнти процесів, умови середовища, наддув, фази, параметри упорскування тощо;
- довідник палива (fuel_types із параметрами складу та властивостей).

Вихідні дані:

- calculation_results.json: інтегральні параметри та службові значення (із одиницями та описами);
- графіки та табличні матеріали у каталозі plots/;
- (опційно) збережені профілі по φ для індикаторної діаграми та фазового аналізу.

А.8 Стадії та етапи робіт

1. аналіз предметної області та аналогів;

2. формування вимог і концепції;
3. проєктування структури даних та модульної архітектури;
4. реалізація математичної моделі робочого циклу;
5. реалізація збереження результатів і візуалізації;
6. тестування та перевірка узгодженості;
7. підготовка документації та матеріалів для впровадження.

А.9 Порядок контролю та приймання

Робота вважається виконаною за умови, що програмний продукт:

- коректно виконує розрахунок для типового набору параметрів;
- формує таблиці та графіки;
- зберігає результати у форматі JSON;
- забезпечує повторюваність результатів та не містить критичних помилок.

Додаток Б – загальний опис інформаційної системи «DieselCycleUa»

Б.1 Призначення системи

«DieselCycleUa» – програмна платформа, що реалізує цифровий двійник дизельного двигуна у вигляді моделі повного робочого циклу. Система призначена для дослідницьких і навчальних задач: оцінювання впливу наддуву, параметрів паливоподачі, властивостей палива, коефіцієнта надлишку повітря та фазових налаштувань на інтегральні й фазові показники циклу.

Б.2 Загальна архітектура

Система реалізована як Python-застосунок і містить такі логічні підсистеми:

1. Підсистема конфігурації та даних: завантаження параметрів із JSON; робота з довідником палива.
2. Розрахункове ядро: послідовний конвеєр обчислень (наддув → наповнення/стискання → згоряння/розширення → метрики).
3. Підсистема збереження: формування структури результатів і запис у JSON; опційно – запис у БД (SQLite).
4. Підсистема візуалізації: побудова індикаторної діаграми та допоміжних графіків; формування таблиць на базі pandas.
5. Інтерфейс користувача: консольний режим; за потреби – локальний GUI (Tkinter).

Б.3 Алгоритмічний конвеєр обчислень

Обчислення організовані як послідовність етапів:

- зчитування parameters.json (за потреби – проміжних/попередніх результатів);
- визначення геометричних параметрів і базових коефіцієнтів;
- розрахунок наддуву (тиск і температура після компресора та перед циліндром);
- розрахунок параметрів наповнення, залишкових газів, початку/кінця стискання;
- розрахунок характерних точок згоряння та контрольних параметрів P_c, T_c, P_z, T_z;

- формування інтегральних показників (ККД, витрати, потужність, питомі витрати);
- розрахунок підмоделі упорскування/випаровування/тепловиділення;
- інтегрування по куту φ для отримання масивів $P(\varphi), T(\varphi)$ та побудови індикаторної діаграми;
- інтегрування $\oint P\ dV$ для визначення індикаторної роботи та енергетичних метрик за інтегрованим циклом.

Б.4 Інформаційне забезпечення (формати та структури)

- parameters.json – основний вхідний файл (налаштування двигуна, режиму, середовища, паливоподачі).
- calculation_results.json – вихідний файл зі структурою: параметр → (значення, одиниця, опис).
- Профілі по φ зберігаються як масиви у вкладених полях результатів (наприклад: phi, P, T, V, dx, x тощо).

Б.5 Основні результати, що формуються системою

- таблиця інтегральних параметрів циклу (тиски/температури у точках, ККД, витрати, потужність);
- індикаторна діаграма $P(\varphi)$;
- графіки допоміжних характеристик (об’єм, частка випаровування, швидкість випаровування, тепловиділення, профіль згоряння);
- файл результатів JSON для повторного аналізу та документування.

Додаток В – інструкція користувача (DieselCycleUa)

В.1 Призначення

Інструкція встановлює порядок підготовки вхідних даних, запуску програми та отримання результатів моделювання робочого циклу дизельного двигуна в програмній платформі «DieselCycleUa».

В.2 Вимоги до системи

- Python 3.x
- Встановлені бібліотеки: numpy, pandas, matplotlib
- (опційно) tkinter – для локального GUI
- Файли проєкту розміщені в одній робочій директорії: основний скрипт, parameters.json, модуль довідника палива, модуль БД (за наявності).

В.3 Підготовка вхідних даних

1. У робочій папці повинен бути файл parameters.json.
2. У файлі задаються параметри двигуна та режиму: діаметр циліндра, хід, ступінь стиску, частота обертання, параметри наддуву, фази, параметри упорскування, коефіцієнт надлишку повітря тощо.
3. Тип палива вибирається зі списку довідника (fuel_types) або додається як новий запис через функцію розширення довідника.

В.4 Запуск програми

1. Запуск виконується стандартним способом (через IDE або командний рядок).
2. У консолі обирається тип палива шляхом введення ключа зі списку доступних.
3. Після запуску формується вивід:

- ключові параметри у консолі;
- повна таблиця результатів у форматі pandas.DataFrame;
- графіки (індикаторна діаграма та допоміжні залежності).

В.5 Отримання та збереження результатів

- Основні результати зберігаються у структурі results і записуються у calculation_results.json.

- Для подальшого аналізу доцільно зберігати:
  - JSON із параметрами циклу;
  - профілі по φ (масиви phi, P, T, V та інші);
  - за необхідності – експорт таблиць у CSV.

B.6 Типові помилки та способи усунення

- parameters.json не знайдено: перевірити назву файлу та розташування у робочій папці.
- Некоректний JSON: перевірити синтаксис (коми, лапки, дужки).
- Відсутній ключ параметра: додати параметр у JSON або визначити значення за замовчуванням у коді.
- Нереалістичні результати (надто великі/від'ємні): перевірити одиниці вимірювання (Па/кПа/МПа), коректність коефіцієнтів і фаз, відповідність режиму заданим межам.

Додаток Г – текст програми

У друкованій версії наведено структуру, склад модулів та ключові фрагменти, необхідні для розуміння логіки роботи ІС «Цифровий двійник» дизельного двигуна. Повний вихідний код (усі функції, розширені підмоделі, повні формули та сервіси БД) подається в електронному вигляді як «Текст програми (повна версія)».

Г.1 Склад програмних компонентів

Основний файл (точка входу):

dieselcycleua.py – оркестрація: завантаження параметрів → валідація → розрахунок → постобробка → збереження → візуалізація.

Довідник палива:

fuel_data.py – структура fuel_types, функції доступу та додавання нового палива.

Модуль БД (опційно):

database.py – ініціалізація SQLite-схеми та збереження деталізованих результатів (профілі, метрики, метадані запуску).

Конфігурація запуску:

parameters.json – вхідні параметри двигуна/режиму/моделі (у т.ч. крок ($\Delta\varphi$)).

Вихідний файл результатів:

calculation_results.json – інтегральні метрики + (за потреби) агреговані профілі та службові дані.

Каталог ілюстрацій:

plots/ – графіки (індикаторна діаграма ($p! - !V$), ($P(\varphi)$), ($T(\varphi)$), таблиці у вигляді рисунків).

Логіка репозиторію: друкований додаток показує *що і як працює*, а електронний – містить *повний код*.

Г.2 Основні логічні блоки програми

Імпорт бібліотек, визначення констант та шляхів збереження (plots/, run/).

Завантаження конфігурації з parameters.json (структурний словник параметрів).

Вибір палива з довідника fuel_types та отримання його властивостей.

Термодинамічний розрахунок характерних точок (впуск/стиск/згоряння/розширення) та первинних інтегральних показників.

Підмоделі процесів (упорскування/розпил/випаровування/тепловиділення) – формування (σ(φ)), ($x(\varphi)$), (dx/d) тощо.

Інтегрування циклу по куту (φ) з формуванням профілів ($P(\varphi)$), ($T(\varphi)$), ($V(\varphi)$) (та службових масивів).

Візуалізація результатів: індикаторна діаграма ($p! - !V$), допоміжні графіки та табличні підсумки.

Формування структури результатів і збереження у calculation_results.json (та опційно в SQLite).

Г.3 Лістинг ключових фрагментів (структурний приклад)

Нижче наведено скорочені фрагменти (показують структуру та принципи), без дублювань і надмірного обсягу.

1) Створення каталогу результатів plots/

```
import os

PLOTS_DIR = "plots"

def ensure_dirs() -> None:
    """Створює каталоги для графіків та запусків (якщо
відсутні)."""
    os.makedirs(PLOTS_DIR, exist_ok=True)
```

2) Завантаження параметрів із JSON

```
import json
from typing import Any, Dict

def load_parameters(filepath: str = "parameters.json") ->
Dict[str, Any]:
    """Завантажує параметри запуску з JSON. Повертає словник
параметрів."""
    try:
        with open(filepath, "r", encoding="utf-8") as f:
            return json.load(f)
    except FileNotFoundError as e:
        raise RuntimeError(f"Файл конфігурації не знайдено:
{filepath}") from e
    except json.JSONDecodeError as e:
        raise RuntimeError(f"Некоректний JSON у файлі:
{filepath}") from e
```

3) Довідник палива

```
# fuel_data.py
from typing import Dict, Any

fuel_types: Dict[str, Dict[str, Any]] = {
    "dt": {
        "name": "Diesel fuel",
        "C": 0.86, "H": 0.13, "O": 0.01, "S": 0.00,
        "Qn": 42500e3,      # Дж/кг
        "Density": 830.0,   # кг/м^3
        "Cetane": 50.0
    }
}

def add_fuel_type(key: str, props: Dict[str, Any]) -> None:
    """Додає нове паливо у довідник (ключ + набір
властивостей)."""
    if key in fuel_types:
        raise ValueError(f"Паливо '{key}' вже існує у
довіднику.")
    fuel_types[key] = props
```

4) Уніфікована структура результатів

```
from typing import Dict

ResultsDict = Dict[str, Dict[str, object]]
results: ResultsDict = {}

def add_result(key: str, value: float, unit: str, description:
str) -> None:
    """Додає один параметр у результати із одиницями та
поясненням."""
    results[key] = {
        "value": float(value),
        "unit": unit,
        "description": description
    }
```

5) Збереження результатів у calculation_results.json

```
import json
from typing import Dict, Any

def save_results(data: Dict[str, Any], filename: str =
"calculation_results.json") -> None:
    """Зберігає результати у JSON (читабельно для перевірки та
відтворення)."""
    with open(filename, "w", encoding="utf-8") as f:
        json.dump(data, f, ensure_ascii=False, indent=2)
```

6) Кінематика: об’єм циліндра ($V(\varphi)$) та похідна (dV/d)

```
import math

def calc_volume(phi_deg: float, V_s: float, epsilon: float,
lambda_ratio: float) -> float:
    """
    Поточний об'єм циліндра V(φ), м³.
    phi_deg – кут КВ у градусах; V_s – робочий об'єм; epsilon –
ступінь стиснення;
    lambda_ratio – відношення r/l (або еквівалентний параметр
кінематики).
    """
    phi = math.radians(phi_deg)
    V_c = V_s / (epsilon - 1.0)  # об'єм камери згоряння
    return V_c + 0.5 * V_s * (1 - math.cos(phi) + 0.5 *
lambda_ratio * math.sin(phi) ** 2)

def calc_dV_dphi(phi_deg: float, V_s: float, lambda_ratio:
float) -> float:
    """Похідна dV/dφ (φ у градусах)."""
    phi = math.radians(phi_deg)
    # похідна за φ (у радіанах) * d(rad)/d(deg)
    dV_drad = 0.5 * V_s * (math.sin(phi) + 0.5 * lambda_ratio *
math.sin(2 * phi))
    return dV_drad * (math.pi / 180.0)
```

7) Інтегрування циклу по (φ)

```
import numpy as np
from typing import Dict, List

def integrate_cycle(params: dict, fuel: dict) -> Dict[str,
List[float]]:
    """
    Інтегрування повного 4-тактного циклу по φ (0–720°) з
кроком Δφ.
    Повертає профілі: phi, P, T, V, x, dx (за потреби).
    """
    dphi = float(params.get("dphi_deg", 1.0))
    phi_grid = np.arange(0.0, 720.0 + dphi, dphi)

    P = np.zeros_like(phi_grid)  # Па або кПа (узгодити у
проекті)
    T = np.zeros_like(phi_grid)  # K
    V = np.zeros_like(phi_grid)  # м³

    # (скорочено) – ініціалізація початкових умов
```

```
        # P[0] = ...
        # T[0] = ...

        for i, phi in enumerate(phi_grid):
            V[i] = calc_volume(phi, params["V_s"],
params["epsilon"], params["lambda_ratio"])
            # (скорочено) – тут: газообмін / стиск / згоряння /
розширення
            # P[i], T[i] = step_update(...)

        return {"phi": phi_grid.tolist(), "P": P.tolist(), "T":
T.tolist(), "V": V.tolist()}
```

8) Побудова індикаторної діаграми ($p! - !V$)

```
    import matplotlib.pyplot as plt

    def plot_indicator_pV(profiles: dict, out_png: str =
"plots/indicator_pV.png") -> None:
        """Будує індикаторну діаграму p–V для 4-тактного дизельного
циклу."""
        V = np.array(profiles["V"])
        P = np.array(profiles["P"])

        plt.figure(figsize=(6.2, 4.2))
        plt.plot(V, P, linewidth=2)
        plt.xlabel("V, м³")
        plt.ylabel("p, Па")  # або "p, кПа" – за прийнятою системою
одиниць
        plt.grid(True)
        plt.tight_layout()
        plt.savefig(out_png, dpi=200)
        plt.close()
```

9) Точка входу: сценарій «load → validate → compute → save → plot»

```
    from fuel_data import fuel_types

    def main() -> None:
        ensure_dirs()

        params = load_parameters("parameters.json")

        fuel_key = params.get("fuel_key", "dt")
        if fuel_key not in fuel_types:
            raise ValueError(f"Паливо '{fuel_key}' відсутнє у
довіднику fuel_types.")
        fuel = fuel_types[fuel_key]
```

```
        # validate(params)  # у повній версії: межі, одиниці,
фізична припустимість

        profiles = integrate_cycle(params, fuel)

        # приклад метрик (у повній версії: Li = ∮P dV, pi, pe, η,
g_e тощо)
        add_result("fuel_name", 0.0, "-", f"Обране паливо:
{fuel.get('name','')}")
        add_result("dphi", float(params.get("dphi_deg", 1.0)),
"deg", "Крок інтегрування по φ")

        output = {
            "metrics": results,
            "profiles": {k: profiles[k] for k in ("phi", "P", "T",
"V")}
        }
        save_results(output, "calculation_results.json")

        plot_indicator_pV(profiles, "plots/indicator_pV.png")

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

Г.4 Опис вхідних/вихідних структур

Г.4.1 Вхідна структура parameters.json (логіка)

Файл конфігурації містить:

геометрію та кінематику (наприклад: V_s, epsilon, lambda_ratio, diameter, stroke);

режим роботи (наприклад: crankshaft_speed, P0, T0, наддув);

паливо (fuel_key для вибору з довідника);

параметри моделі (крок ($\Delta\varphi$), коефіцієнти підмоделей).

Критично: конфігурація повинна бути самодостатньою, щоб запуск був відтворюваним.

Г.4.2 Вихідна структура calculation_results.json

Вихідний JSON містить два основні блоки:

metrics – словник результатів, де кожен параметр має поля:

value – числове значення;

unit – одиниця виміру;

description – текстовий опис.

profiles – профілі по фазі (φ) у вигляді масивів однакової довжини:

phi – кут КВ (°);

P – тиск;

T – температура;

V – об’єм;

x, dx, sigma тощо – для фазового аналізу горіння.

У повній версії програми реалізовано розширений набір підмоделей (упорскування/випаровування/тепловиділення/газообмін) та процедури збереження в SQLite. У друкованому додатку наведено лише ті фрагменти, що демонструють архітектуру, потоки даних та ключові інтерфейси модулів.