Міністерство освіти і науки України
Український державний університет науки і технологій
Факультет «Комп'ютерні технології і системи»
Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка

до кваліфікаційної роботи

ОС магістра

на тему: Дослідження складності конструктивно-продукційних моделей зон рекуперації постійного струму

за освітньою програмою 12 Інженерія програмного забезпечення

зі спеціальності: 121 Інженерія програмного забезпечення

Виконав: студент групи ПЗ2421: /Євген ШАПОВАЛ/

Керівник: /проф. Віктор ШИНКАРЕНКО/

Нормоконтролер: /доц. Світлана ВОЛКОВА/

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент

Дніпро – 2026

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies
Faculty «Computer technologies and systems»
Department «Computer information technology»

Explanatory Note

to Master's Thesis

on the topic: Study of the complexity of structural and production models of DC recovery zones

according to educational curriculum 12 Software engineering

in the speciality: 121 Software engineering

Done by the student of the group PZ2421: /Yevhen SHAPOVAL/

Scientific Supervisor: /prof. Viktor SHYNKARENKO/

Normative controller: /doc. Svitlana VOLKOVA/

Dnipro – 2026

Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Рівень вищої освіти: магістр

Освітня програма: 12 Інженерія програмного забезпечення

Спеціальність: 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри КІТ

_____ /Вадим ГОРЯЧКІН/

Дата _____

ЗАВДАННЯ

на кваліфікаційну роботу ОС магістра

студенту _____Шаповалу Євгену Віталійовичу_____

1. Тема роботи: «Дослідження складності конструктивно-продукційних моделей зон рекуперації постійного струму»
   Керівник роботи: проф. Віктор ШИНКАРЕНКО,
   затверджені наказом №1401 ст від 02.10.2025
2. Строк подання студентом роботи: 09.01.2026
3. Вихідні дані до роботи: Конструктивно-продукційні моделі зон рекуперації у формалізованому вигляді (LaTeX); розроблений програмний засіб `electron-svelte-latex` для аналізу метрик Холстеда.
4. Зміст пояснювальної записки (перелік питань до розробки): Вступ; Практичне значення роботи та методологія дослідже; Аналітична частина та теоретичні основи; Розробка програмного засобу для аналізу складності формул; Тестування та налагодження програмного засобу; Експериментальне дослідження та аналіз результатів; Загальні висновки; Список використаних джерел; Додатки.
5. Перелік демонстраційного матеріалу:
   1. Презентація у форматі PDF;
   2. Демонстраційне відео роботи розробленого програмного засобу;
   3. Пояснювальна записка.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	04.10.2025 – 07.10.2025	
2	Формалізація задачі, розробка Технічного завдання	08.10.2025 – 15.10.2025	
3	Проектування архітектури програмного засобу	16.10.2025 – 27.10.2025	30 %
4	Реалізація модулей ПЗ	28.11.2025 – 08.12.2025	
5	Розробка інтерфейсу користувача	09.12.2025 – 12.12.2025	
6	Тестування та налагодження ПЗ	15.12.2025 – 20.12.2025	60 %
7	Проведення експериментальних досліджень та аналіз результатів	20.12.2025 – 25.01.2026	
8	Написання основних розділів ПЗ та висновків	25.12.2025 – 04.01.2026	
9	Підготовка демонстраційних матеріалів	05.01.2026 – 08.01.2026	100 %
10	Подання кваліфікаційної роботи до кафедри	09.01.2026	
11	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	22.01.2026	

Студент _____ Євген ШАПОВАЛ

Керівник роботи _____ проф. Віктор ШИНКАРЕНКО

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра: 80 с., 20 рис., 28 табл., 30 джерел, 4 додатка.

Об’єкт дослідження – процеси аналізу та оцінки складності формалізованих моделей на прикладі конструктивно-продукційних моделей (КПМ) зон рекуперації постійного струму.

Предмет дослідження – методи кількісної оцінки складності, зокрема метрики Холстеда, та їх застосування для аналізу конструктивно-продукційних моделей.

Мета роботи – дослідити та кількісно оцінити складність КПМ зон рекуперації постійного струму шляхом розробки програмного інструментарію, який автоматизує розрахунок метрик Холстеда.

Методи дослідження – системний аналіз, елементи теорії конструктивно-продукційного моделювання, метричний аналіз (метрики Холстеда), об’єктно-орієнтоване проєктування, обчислювальний експеримент. Програмну реалізацію виконано із використанням Electron, Svelte та TypeScript.

У роботі проаналізовано предметну область КПМ та підходи до оцінювання складності формалізованих описів. Запропоновано методику, за якою семантичні елементи моделі (операції, зв’язки, функції) розглядаються як оператори, а сутності та їх параметри (символи, константи, атрибути) – як операнди, що дає змогу коректно обчислювати метрики Холстеда для моделей у вигляді формул.

Спроєктовано та реалізовано програмний засіб, який підтримує повний цикл дослідження: автоматичне вилучення LaTeX-описів моделей з документів, перетворення виразів у семантичне дерево MathJSON, збір операторів та операндів рекурсивним обходом, розрахунок метрик (Volume, Difficulty, Effort), а також візуалізацію результатів і порівняння моделей. Для зручності роботи реалізовано редагування та перейменування моделей, а також збереження і відновлення проєктів у власному форматі файлів.

Проведено експериментальне дослідження на вибірці КПМ, що підтвердило працездатність запропонованої методики та коректність роботи інструменту. Додатково виконано тестування ключових модулів обробки формул (вилучення, очищення та обчислення метрик), що забезпечило стабільність результатів.

Практичне значення роботи полягає у можливості швидко отримувати об’єктивні числові оцінки складності моделей і порівнювати альтернативні конфігурації зон рекуперації на ранніх етапах проєктування.

КЛЮЧОВІ СЛОВА: КОНСТРУКТИВНО-ПРОДУКЦІЙНЕ МОДЕЛЮВАННЯ, ОЦІНКА СКЛАДНОСТІ, МЕТРИКИ ХОЛСТЕДА.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАК, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Скорочення предметної області та загальні:

ГІК – графічний інтерфейс користувача.

ЗР – зона рекуперації.

КПМ – конструктивно-продукційна модель.

КПС – конструктивно-продукційна структура.

ПЗ – програмний засіб (програмне забезпечення).

ТСЕ – система тягового електропостачання.

Математичні позначення та метрики Холстеда:

D – складність (за Холстедом).

E – зусилля (показник трудомісткості розуміння/супроводу) за Холстедом.

$N = N_1 + N_2$ – довжина моделі.

N_1 – загальна кількість операторів.

N_2 – загальна кількість операндів.

$n = n_1 + n_2$ – словник (словниковий запас) моделі.

n_1 – кількість унікальних операторів.

n_2 – кількість унікальних операндів.

V – обсяг (інформаційний обсяг) за метриками Холстеда.

Операнд – елемент формалізованого опису, що є сутністю, параметром або значенням (у межах метрик Холстеда).

Оператор – елемент формалізованого опису, що задає дію, зв’язок або перетворення (у межах метрик Холстеда).

Технології та інструменти:

IPC (міжпроцесна взаємодія) – механізм обміну повідомленнями між процесами застосунку.

LaTeX – мова розмітки для набору математичних формул.

Pandoc – конвертер документів, який може перетворювати `.docx` у проміжне подання для подальшого аналізу.

Worker thread (потік-обробник) – окремий потік для виконання обчислювально важких операцій без блокування інтерфейсу.

Формати даних і файли:

`.docx` – формат документів Microsoft Word.

`.fmx` – власний формат файлів проєкту для збереження переліку моделей, їх метаданих та результатів аналізу.

`.tex` – текстовий файл з LaTeX-розміткою.

JSON – текстовий формат обміну даними (JavaScript Object Notation).

MathJSON – формат семантичного подання математичних виразів у вигляді дерева.

ВСТУП

Сучасні системи тягового електропостачання постійного струму дедалі активніше використовують рекуперацію енергії, однак ефективність її повернення суттєво залежить від структури зони рекуперації та логіки керування. Для опису таких систем зручно застосовувати конструктивно-продукційні моделі (КПМ), які формалізують побудову структури у вигляді правил підстановки та послідовностей їх застосування. Проте зі зростанням кількості елементів і зв’язків моделі швидко ускладнюються: зростає навантаження на розуміння, підвищується ризик помилок у проєктуванні та стає важче порівнювати альтернативні конфігурації на основі «інтуїції». Саме тому актуальною є задача кількісної оцінки складності КПМ – з можливістю автоматизованого аналізу великої кількості моделей.

Об’єкт дослідження – процеси аналізу та оцінки складності конструктивно-продукційних моделей зон рекуперації постійного струму.

Предмет дослідження – методи кількісної оцінки складності формалізованих описів, зокрема метрики Холстеда, та їх застосування для аналізу КПМ.

Мета роботи – дослідження складності конструктивно-продукційних моделей зон рекуперації постійного струму та розробка інструментальних засобів для автоматизованого розрахунку і візуалізації метрик складності.

Для досягнення поставленої мети необхідно розв’язати такі завдання:

- проаналізувати теоретичні основи конструктивно-продукційного моделювання та особливості подання КПМ у формалізованому вигляді;
- обґрунтувати доцільність використання метрик Холстеда для кількісної оцінки складності моделей та визначити правила віднесення елементів КПМ до операторів і операндів;
- розробити методику автоматизованого вилучення описів моделей та підготовки їх до аналізу (очищення, нормалізація, усунення неоднозначностей);
- спроєктувати та реалізувати програмний засіб для семантичного розбору описів моделей, обчислення метрик Холстеда та відображення результатів у зручному вигляді;
- виконати експериментальне дослідження на вибірці КПМ і проінтерпретувати отримані результати;

- перевірити працездатність і коректність ключових модулів програмного засобу за допомогою тестування.

Методи дослідження – системний аналіз, елементи теорії конструктивно-продукційного моделювання, метричний аналіз (метрики Холстеда), об’єктно-орієнтоване проєктування, обчислювальний експеримент, тестування програмного забезпечення. Програмну реалізацію виконано із використанням Electron, Svelte та TypeScript.

У ході роботи сформовано методологію аналізу КПМ, у якій семантичні елементи моделі (операції, зв’язки, функції) трактуються як оператори, а сутності та їх параметри – як операнди. Реалізовано програмний інструментарій, що автоматизує процес вилучення описів моделей, їх семантичне подання у вигляді дерева MathJSON, обчислення метрик Холстеда (обсяг V, складність D, зусилля E) та візуалізацію результатів для порівняльного аналізу.

Практичне значення отриманих результатів полягає в можливості швидко отримувати об’єктивні числові оцінки складності КПМ і використовувати їх для порівняння варіантів конфігурацій зон рекуперації на ранніх етапах проєктування та дослідження.

Структура роботи включає вступ, п’ять розділів, висновки та список використаних джерел.

1 ПРАКТИЧНЕ ЗНАЧЕННЯ РОБОТИ ТА МЕТОДОЛОГІЯ ДОСЛІДЖЕННЯ

1.1 Актуальність і практичний контекст

Зони рекуперації у системах тягового електропостачання постійного струму – це не просто “ділянки мережі”, а місця, де в один момент часу сходяться енергетика, логіка керування та обмеження безпеки. Під час гальмування рухомий склад може повертати енергію в мережу, але ця енергія не завжди може бути корисно використана: за відсутності споживача або за невдалого режиму керування вона переходить у втрати (наприклад, через гальмівні резистори або обмеження по напрузі). Тому інженерні рішення в цій області часто зводяться до порівняння багатьох варіантів схем, правил і режимів, а також до пошуку “раціонального” керування, яке працює стабільно в різних поїзних ситуаціях [1].

У межах конструктивно-продукційного моделювання КПМ/КПС зручно задають такі варіанти у формалізованому вигляді: через структуру схеми та набір продукцій/правил, які описують переходи між станами, взаємодію підстанцій, накопичувачів, навантажень тощо. Це добре видно на прикладах робіт, де КПМ застосовують саме до задач тягового електропостачання і рекуперації [2–4].

Проблема в тому, що “більше правил” і “більше елементів” ще не означає “краще”: складні моделі важче перевіряти, підтримувати, порівнювати між собою й пояснювати іншим. У реальній роботі це швидко перетворюється на ситуацію, коли зміни в моделі стають ризиком: важко передбачити побічний ефект, важко знайти помилку, важко обґрунтувати, чому саме цей варіант кращий за інший.

Звідси і практичне значення цієї роботи: потрібен простий, відтворюваний і максимально “об’єктивний” спосіб порівнювати складність різних КПМ зон рекуперації. Щоб не сперечатися “на відчуттях”, а мати числові оцінки, які можна звести в рейтинг, побачити на графіках і пояснити в звіті.

Окремий практичний нюанс – у яких форматах ці моделі живуть. У публікаціях та внутрішніх матеріалах формалізовані описи дуже часто записані як формули: у LaTeX або у Word (`.docx`). Саме тому методика в цій роботі спеціально підлаштована під аналіз формульного подання (а не “ручне переписування” в якийсь окремий формат).

1.2 Об’єкт та предмет дослідження

Об’єкт дослідження – процеси аналізу та кількісної оцінки складності конструктивно-продукційних моделей зон рекуперації постійного струму в межах задач системи тягового електропостачання [2].

Предмет дослідження – методика адаптації метрик Холстеда до елементів формалізованого (формульного) опису КПМ, а також програмні методи вилучення, нормалізації та семантичного опрацювання формул з файлів `.tex` і `.docx` для автоматизованого розрахунку цих метрик [5].

1.3 Мета й завдання дослідження

Мета роботи – розробити та перевірити методологію кількісної оцінки складності КПМ зон рекуперації постійного струму і створити програмний засіб, який автоматизує цей аналіз на матеріалі реальних формалізованих описів.

Для досягнення мети було виконано такі завдання:

- проаналізувати практичні приклади КПМ/КПС для задач тягового електропостачання і рекуперації та визначити, які саме фрагменти формул доцільно брати як “модель” для порівняння [2–4];
- обґрунтувати вибір метрик Холстеда як інструменту порівняльної оцінки складності та визначити обмеження їх застосування (метрики корисні саме для порівняння варіантів, а не як “абсолютна істина”) [5–7];
- реалізувати вилучення формул з `.tex` (пошук математичних середовищ і inline-формул) з прив’язкою до джерела;
- реалізувати вилучення формул з `.docx` через конвертацію у LaTeX-подання (конвеєр на базі Pandoc) [8];
- виконати валідацію та очищення LaTeX-фрагментів (перевірка балансу дужок/розділювачів, нормалізація запису, обробка типових “сміттєвих” символів);
- побудувати семантичне подання формул у вигляді дерева (AST/MathJSON) для подальшого обходу та підрахунку елементів [9];
- визначити правила класифікації операторів і операндів для КПМ та реалізувати збір n_1, n_2, N_1, N_2;
- реалізувати розрахунок похідних метрик Холстеда (обсяг V, складність D, зусилля E) та підготовку результатів до порівняння;

- реалізувати подання результатів у зручному вигляді (сортування, графіки, порівняння моделей між собою);
- додати тестування ключових частин (коректність вилучення, стабільність парсингу, правильність підрахунку та формул метрик).

1.4 Методика проведення дослідження

Методика побудована як послідовний конвеєр: від підготовки формульних описів до отримання порівняльних числових оцінок і їх візуального представлення. Ключова ідея – звести різні за походженням джерела (LaTeX/Word) до одного семантичного подання, а вже після цього рахувати метрики однаковим способом.

1.4.1 Підготовка дослідного матеріалу

Як дослідний матеріал використовувалися формалізовані описи моделей, що відображають структуру та правила роботи зон рекуперації постійного струму. Основу вибірки склали приклади КПМ/КПС з публікацій, де конструктивно-продукційний підхід застосовується до задач тягового електропостачання і розподілу енергії рекуперації [2–4].

Щоб порівняння було чесним, для кожної моделі фіксувалися:

- джерело (файл/документ);
- межі формульного фрагмента, який вважається “описом моделі”;
- мінімальні метадані (назва моделі/варіанта, коротка примітка, за потреби – група).

Окремо до вибірки додавалися “контрольні” штучні приклади невеликого розміру – для перевірки крайніх випадків (вкладені дужки, індекси, функції, складні оператори тощо). Це важливо для тестування, бо в реальних документах такі конструкції трапляються нерівномірно.

1.4.2 Вилучення формул з `.tex`

Для файлів `.tex` вилучення виконувалося через пошук математичних конструкцій у тексті:

- inline-формули (`$...$`, `\(...\)`),
- блочні формули (`\[...\]`),
- типові математичні середовища (`equation`, `align`, `gather` тощо).

На практиці важливо не просто “вирізати по символах”, а робити це акуратно: з урахуванням вкладених фігурних дужок, команд LaTeX та типових випадків, коли символи `$` або дужки зустрічаються у нетипових контекстах. Тому в методиці передбачено крок попередньої нормалізації та перевірки балансу розділювачів ще до семантичного парсингу.

1.4.3 Вилучення формул з `.docx` через конвертацію

У `.docx` математичні формули зберігаються у власному внутрішньому поданні, тому прямий парсинг “як тексту” майже завжди дає поганий результат. Практичне рішення – конвертувати документ у LaTeX-подання, яке вже можна обробляти так само, як `.tex`.

У цій роботі для цього використано Pandoc як універсальний конвертер: `.docx` → проміжне LaTeX-подання → вилучення формул тим самим механізмом, що і для `.tex` [8]. Це дає дві переваги:

1. однаковий формат на вході семантичного аналізу;
2. відтворюваність конвеєра (важливо для повторних експериментів).

1.4.4 Валідація та очищення LaTeX

Перед семантичним аналізом кожен фрагмент проходить валідацію та “легке” очищення. Мета тут не “переписати формулу”, а прибрати те, що заважає стабільному парсингу і спотворює підрахунки.

Типові кроки:

- перевірка балансу фігурних дужок `{}` та базових розділювачів;
- нормалізація пробілів і службових переносів;
- уніфікація деяких варіантів запису (наприклад, дрібні стилістичні відмінності, які не несуть семантики, але впливають на токенізацію);
- контроль того, що фрагмент дійсно є “математикою”, а не випадковим шматком тексту.

Цей крок помітно підвищує відсоток формул, які можна аналізувати автоматично без ручного втручання.

1.4.5 Класифікація операторів/операндів та збір метрик

Ключовий момент адаптації – узгодити, що саме у формулі вважається оператором, а що – операндом.

У класичному розумінні метрик Холстеда оператори – це “дії/операції”, а операнди – “сущності/значення” [5]. Для формалізованих описів КПМ це інтерпретується так:

- до операторів відносяться, зокрема: математичні операції, логічні зв’язки, відношення, функціональні застосування, а також конструкції, які задають перетворення або залежності між елементами моделі;
- до операндів відносяться ідентифікатори сутностей (елементи схеми, змінні, параметри), константи, індекси та значення, які “підставляються” в оператори.

Щоб ці правила були реалізовані не “на око”, формула перетворюється у деревоподібне подання, після чого виконується рекурсивний обхід і підрахунок:

- n_1 – кількість унікальних операторів,
- n_2 – кількість унікальних операндів,
- N_1 – загальна кількість операторів,
- N_2 – загальна кількість операндів.

Далі обчислюються стандартні похідні величини (у прийнятій інтерпретації Холстеда) [5]:

$$n = n_1 + n_2, \tag{1.1}$$

$$N = N_1 + N_2, \tag{1.2}$$

$$V = N * \log_2(n), \tag{1.3}$$

$$D = \left(\frac{n_1}{2}\right) * \left(\frac{N_2}{n_2}\right), \tag{1.4}$$

$$E = D * V \tag{1.5}$$

Водночас у методиці прямо враховано, що метрики Холстеда – це інструмент порівняння й оцінки “інформаційної насиченості/складності”, а не універсальний вимір “якості” моделі. Тому результати інтерпретуються обережно: нас цікавить не абсолютне число, а різниця між варіантами та її пояснюваність [6, 7].

Для семантичного подання в реалізації використовується MathJSON (дерево виразу), що дозволяє уніфікувати аналіз формул незалежно від їх початкового формату та стабільно обходити структуру [9].

1.4.6 Узагальнення та візуалізація результатів

Після розрахунку метрик для кожного зразка формується набір результатів, придатний для порівняння:

- ранжування моделей за V, D, E;
- порівняння груп (наприклад, різні варіанти однієї схеми або різні підходи до опису правил);
- візуальні представлення (графіки/діаграми), які дозволяють швидко побачити "переломні точки": де складність починає рости непропорційно.

Оскільки застосунок орієнтований на практичну роботу з вибіркою формул, результати подаються так, щоб їх було легко перевірити: з можливістю швидко перейти від числа назад до конкретної формули/моделі.

1.4.7 Тестування та перевірка коректності

Щоб результати були надійними, методика передбачає тестування на трьох рівнях:

- тести вилучення: перевіряють, що з файлу беруться саме ті фрагменти, які очікуються (окремо для `.tex` і `.docx`);
- тести парсингу: перевіряють стабільність перетворення формул у деревоподібне подання (включно з вкладеними конструкціями);
- тести метрик: для контрольних прикладів з відомими підрахунками перевіряється правильність n_1, n_2, N_1, N_2 та похідних величин.

Додатково застосовується точкова ручна верифікація для кількох реальних моделей з вибірки: це потрібно, щоб переконатися, що правила класифікації операторів/операндів не дають систематичного перекосу на "живих" формулах.

Висновки до розділу 1

У цьому розділі обґрунтовано практичну потребу у кількісній оцінці складності КПМ зон рекуперації постійного струму та показано, чому саме формульне подання (LaTeX/Word) є природним джерелом даних для такого аналізу.

Визначено об'єкт, предмет, мету та конкретні завдання дослідження, які безпосередньо відповідають реалізованим можливостям програмного засобу.

Сформовано методику проведення дослідження у вигляді конвеєра: підготовка вибірки, вилучення формул з `.tex` та `.docx`, валідація й очищення, семантичне подання, класифікація операторів/операндів і розрахунок метрик Холстеда.

Окремо зазначено принцип інтерпретації результатів: метрики розглядаються як інструмент порівняння варіантів моделей, а не як абсолютний критерій "якості".

Передбачено тестування ключових етапів – вилучення, парсингу та обчислення метрик – щоб забезпечити відтворюваність і коректність отриманих оцінок.

2 МЕТРИКИ СКЛАДНОСТІ ТА ФОРМАЛІЗАЦІЯ КПМ ЗОН РЕКУПЕРАЦІЇ

2.1 Предметна область та постановка задачі оцінювання складності

У задачах тягового електропостачання постійного струму рекуперація енергії розглядається як важливий механізм підвищення енергоефективності: частина енергії гальмування може бути повернена в мережу та використана іншими споживачами або накопичувачами [1]. На практиці ефект рекуперації визначається не лише характеристиками рухомого складу, а й конфігурацією дільниці, розміщенням підстанцій, наявністю накопичувачів, режимами споживачів та правилами керування. Через це одна й та сама ідея керування може давати різні результати в різних поїзних ситуаціях.

Для інженерного аналізу важливо мати формалізований опис, який дозволяє:

- фіксувати склад елементів і зв’язків у зоні рекуперації;
- задавати правила та обмеження керування у відтворюваному вигляді;
- порівнювати альтернативні варіанти без втрати деталей.

Конструктивно-продукційні моделі та конструктивно-продукційні структури забезпечують таку “мову опису” через сукупність сутностей та продукційних правил. Для предметної області зон рекуперації цей підхід уже використовується у профільних роботах, де КПМ/КПС застосовуються як засіб формалізації конфігурацій і керувальних впливів [2–4].

Разом із розвитком моделі виникає типова проблема – зростання складності формалізованого опису. Додаються нові сутності, параметри, уточнення правил, винятки та додаткові обмеження. У результаті погіршується читабельність, складніше перевіряти узгодженість продукцій, а порівняння альтернатив часто спирається на суб’єктивні оцінки. Тому виникає практична задача: отримати кількісну оцінку складності формул, що описують КПМ/КПС, так щоб цю оцінку можна було обчислювати автоматично та застосовувати для ранжування варіантів моделей.

У програмній інженерії для подібних задач використовують метрики складності. Сучасні огляди підкреслюють, що метрики корисні як індикатори, однак потребують обережної інтерпретації й не є “універсальною мірою якості” [10, 11]. У межах цієї роботи метрики застосовуються саме як інструмент порівняння та

ранжування формалізованих описів, а не як спосіб прогнозувати дефектність або оцінювати якість програмного коду.

2.2 Конструктивно-продукційне моделювання для зон рекуперації

2.2.1 Базові поняття

Конструктивно-продукційна модель – це формалізований опис, у якому структура об’єкта моделювання задається через множини елементів та систему правил перетворення. У контексті зон рекуперації такими елементами є об’єкти інфраструктури та їхні параметри, а правила – це формальні продукції, що описують допустимі перетворення або уточнення структури та атрибутів моделі.

Конструктивно-продукційна структура – це структурна складова моделі, яка задає склад і організацію елементів, а також узгодженість їх взаємозв’язків. У публікаціях з предметної області КПМ/КПС часто подаються у вигляді системи означень і продукцій, записаних формулами [2–4].

2.2.2 Структура опису

Формалізовані описи КПМ/КПС для зон рекуперації зазвичай містять:

- множини сутностей, що представляють елементи зони рекуперації;
- множини відношень, атрибутів або параметрів, що описують властивості сутностей;
- продукції, які додають елементи, уточнюють атрибути, змінюють склад множин або вводять обмеження.

У таких описах значна частина інформації концентрується саме у формулах: вони одночасно фіксують структуру, правила та залежності. Тому складність КПМ/КПС у цій роботі розглядається через складність формульного запису – тобто через те, наскільки насиченим і структурно складним є опис продукцій та означень.

2.2.3 Типові фрагменти формульних описів

Для моделей зон рекуперації характерні кілька типових формульних фрагментів:

- означення структури у вигляді кортежів або трійок множин;
- продукції перетворення, що змінюють одну або кілька складових структури;
- операції над множинами та підмножинами, включно з об’єднанням;

- правила, які вводять нові елементи або атрибути через фіксовані символи й параметри.

Саме різноманітність таких фрагментів і різний рівень деталізації призводять до істотного розкиду складності формул у межах одного документа, що робить кількісні індикатори корисними для порівняння.

2.3 Метрики Холстеда та адаптація для формульних описів

Метрики Холстеда базуються на підрахунку операторів і операндів у формальному описі. У класичному вигляді вони були запропоновані для програмного тексту, але ідея “операції – сутності” переноситься на формули, якщо визначити правила класифікації елементів формального запису [5].

Для цієї роботи важливо, що метрики Холстеда є обчислюваними: за наявності стабільного деревоподібного подання виразу можна автоматично зібрати частоти операторів та операндів і на їхній основі отримати підсумкові показники. Це відповідає практичній потребі – аналізувати не одну формулу, а вибірку формул з документа.

2.3.1 Оператори й операнди у формулі

У межах методики цієї роботи:

- оператор – елемент формалізованого опису, що задає дію, зв’язок або перетворення;
- операнд – елемент формалізованого опису, що є сутністю, параметром або значенням.

До операторів зараховуються, зокрема, арифметичні дії, операції над множинами, відношення рівності, групування дужками та інші конструкції, які визначають структуру виразу. До операндів належать імена сутностей і множин, символи параметрів, константи, індекси та інші позначення, які виступають “аргументами” операторів.

Ключова вимога – однакові правила класифікації для всіх формул, що порівнюються. Без цього метрики перестають бути зіставними: одна і та сама конструкція в різних формулах має трактуватися однаково.

2.3.2 Базові величини

У методиці використовуються такі підрахунки:

- n_1 – кількість унікальних операторів;
- n_2 – кількість унікальних операндів;
- N_1 – загальна кількість операторів з повторами;
- N_2 – загальна кількість операндів з повторами.

Параметри n_1 і n_2 характеризують різноманітність елементів формули, тоді як N_1 і N_2 відображають насиченість запису повторами операцій і використань позначень.

2.3.3 Похідні метрики та їх інтерпретація

На основі базових величин обчислюються:

- $n = n_1 + n_2$ – словник опису;
- $N = N_1 + N_2$ – довжина опису;
- $V = N * \log_2(n)$ – обсяг за Холстедом;
- $D = \left(\frac{n_1}{2}\right) * \left(\frac{N_2}{n_2}\right)$ – складність за Холстедом;
- $E = D * V$ – зусилля за Холстедом [5].

Для формульних описів КПМ/КПС доцільно інтерпретувати ці показники так:

- V відображає, наскільки "інформаційно насиченим" є запис, і зазвичай зростає зі збільшенням довжини формули та словника позначень;
- D реагує на співвідношення операторів і операндів, а також на повторюваність операндів відносно їх різноманітності, що часто корелює зі сприйняттям структурної заплутаності;
- E поєднує два ефекти та є зручним для ранжування формул за очікуваною трудомісткістю аналізу.

У межах цієї роботи цього достатньо: метою є не прогнозувати дефекти, а порівняти формалізовані описи моделей між собою.

2.3.4 Межі застосування метрик та умови коректного порівняння

Метрики Холстеда не є універсальним мірилом "складності всього", а дають оцінку складності формального запису. Дослідження останніх років показують, що зв'язок метрик із практичними характеристиками залежить від контексту, набору даних і того, як саме визначено елементи підрахунку [10–12].

Щоб порівняння було коректним, необхідно:

- фіксувати правила класифікації операторів і операндів;

- застосовувати однакову нормалізацію запису для всіх формул;
- враховувати, що порівнюються саме формульні описи, а не фізична складність об’єкта моделювання.

2.4 Представлення формул у LaTeX як джерельних даних

LaTeX є стандартним форматом запису математичних виразів у технічних публікаціях і навчальних матеріалах, тому значна частина джерельних формул для аналізу зустрічається саме у цьому вигляді.

2.4.1 Типи формул

У документах зазвичай використовуються:

- inline-формули, які вбудовані у речення;
- display-формули, які подані окремими блоками, інколи з нумерацією або вирівнюванням.

Для автоматизованої обробки це означає, що вилучення має підтримувати короткі фрагменти та великі блоки, де один логічний вираз може бути розбитий переносами рядків і вирівнюванням.

2.4.2 Середовища equation, align, gather та особливості вирівнювання

Середовища вирівнювання використовують спеціальні розділювачі для колонок і переходів рядків. Це важливо з двох причин. По-перше, один “видимий” вираз може складатися з кількох рядків. По-друге, символи вирівнювання та розриву рядка є службовими для LaTeX, але після нормалізації мають бути враховані так, щоб зберегти математичний зміст виразу. Якщо ці особливості ігнорувати, зростає ймовірність некоректного структурного розбору.

2.4.3 Вкладені конструкції

Навіть у відносно коротких формулах зустрічаються вкладені структури:

- дроби, корені, індекси;
- групування дужками з вкладеністю;
- матриці, табличні блоки;
- спеціальні функції та команди.

Саме вкладеність робить небезпечним “плоский” текстовий аналіз: одна й та сама послідовність символів може мати різний зміст залежно від структури.

Тому у методиці цієї роботи основний акцент робиться на деревоподібному поданні виразу.

2.4.4 Типові проблеми

Під час обробки LaTeX найчастіше трапляються:

- користувацькі макроси та нестандартні команди, яких немає у словнику парсера;
- неоднозначності групування, особливо у великих виразах;
- комбінування математичних і текстових команд у межах одного фрагмента;
- конструкції, що коректно відображаються у редакторі, але ускладнюють автоматизований розбір.

З практичного погляду це означає, що потрібні правила перевірки та нормалізації на вході, а також поведінка застосунку у випадку частково некоректного або “неповного” запису.

2.5 Представлення формул у `.docx` та особливості підготовки даних

2.5.1 OOXML та подання математики OMML

Формули у документах `.docx` зберігаються не як звичайний текст, а як структуровані елементи Office Open XML. Для математичних об’єктів використовується Office Math Markup Language, який описує формули як дерево вузлів, зокрема через елементи `oMath` та пов’язані конструкції [13, 14].

2.5.2 Проблеми прямого вилучення

Пряме вилучення формул шляхом “читання тексту” з `.docx` є ненадійним, оскільки математичний запис кодується окремими XML-структурами, а подання може залежати від редактора та варіанта збереження. Для надійного аналізу потрібен або повний розбір OMML-дерева, або приведення документа до іншого формату, зручнішого для подальшої обробки.

2.5.3 Конвертація `.docx` у LaTeX та контроль якості

У межах цієї роботи використовується підхід з конвертацією `.docx` у LaTeX-подання і подальшим аналізом формул так само, як для `.tex`. Такий підхід спрощує уніфікацію: після конвертації застосовуються ті самі правила нормалізації, структурного розбору та підрахунку метрик.

Водночас конвертація може породжувати артефакти: нестандартні команди, зайві службові символи або неточне групування. Тому контроль якості результату конвертації є обов'язковою частиною підготовки даних. Як інструмент конвертації використовується Pandoc, який підтримує перетворення документів між форматами та широко застосовується в інженерній практиці [8].

2.6 Підходи до структурного аналізу формул

2.6.1 Структурний розбір та переваги для метрик

Структурний підхід полягає у перетворенні формули в дерево виразу, де кожна операція і кожен аргумент займають своє місце у вкладеній структурі. Для задачі підрахунку операторів і операндів це принципово: обхід дерева дозволяє однозначно визначати типи вузлів, рахувати частоти елементів і формувати стабільні результати навіть за значної вкладеності.

У цій роботі семантичне подання MathJSON використовується як базовий формат для подальшого аналізу, оскільки воно явно відображає структуру виразу і зручне для рекурсивного обходу. Як засіб перетворення LaTeX у MathJSON розглядається Compute Engine, який декларує підтримку такого перетворення [9].

2.6.2 Текстові методи та їх обмеження

Текстові правила, включно з регулярними виразами, можуть бути корисні для локальних завдань, наприклад, для первинного вилучення фрагментів або для базової нормалізації. Однак для повного аналізу складних формул вони є нестійкими через вкладеність і неоднозначність LaTeX-запису. Саме тому текстові методи у методиці виконують допоміжну роль — як інструмент підготовки даних, а не як основа підрахунку метрик.

2.6.3 Семантичне подання MathJSON як уніфікація

MathJSON дозволяє уніфікувати аналіз незалежно від джерела: після перетворення формули в дерево однаково обробляються фрагменти з `.tex` і `.docx`. Це зменшує кількість різних гілок логіки і спрощує забезпечення відтворюваності результатів. Додатковою перевагою є можливість однаково збирати статистику по всіх формулах вибірки та агрегувати результати на рівні документа.

2.7 Огляд інструментів і обґрунтування вибору підходу

2.7.1 Бібліотеки для розбору та відображення LaTeX

Інструменти для роботи з LaTeX умовно поділяються на:

- рендерери, що орієнтовані на відображення формул у вебі, зокрема MathJax і KaTeX [15, 16];
- бібліотеки, що надають структурний розбір LaTeX-документів і дозволяють отримувати дерево документа [17];
- рішення, що можуть формувати семантичні подання виразів, придатні для подальшого аналізу [9].

Для задачі цієї роботи пріоритет має саме структурний і семантичний аспект: потрібне подання, з якого можна стабільно виділяти оператори й операнди.

2.7.2 Вимоги до інструментів для автоматизованого підрахунку метрик

До інструментального забезпечення висуваються такі вимоги:

- можливість отримати деревоподібне подання виразу або еквівалентну структуру;
- стабільна обробка вкладених конструкцій і службових символів;
- передбачувана поведінка при невідомих командах або частково некоректному записі;
- можливість інтеграції у настільний застосунок із повторюваними запусками аналізу.

Під час реалізації також важливі вимоги до відтворюваності: однаковий ввід має давати однаковий результат, а відхилення мають бути пояснюваними і відслідковуваними.

2.7.3 Узгоджені правила нормалізації як умова відтворюваності

Незалежно від вибраних бібліотек, без узгоджених правил нормалізації порівняння формул є некоректним. Нормалізація включає:

- усунення службових фрагментів, які не впливають на математичний зміст, але заважають аналізу;
- приведення подібних записів до однакового вигляду;
- перевірку балансу дужок і базових структурних обмежень.

Саме наявність таких правил робить можливим обчислюване порівняння складності формул у вибірці, а також повторне застосування методики до нових документів без ручного переписування матеріалу.

2.8 Висновки до розділу 2

У розділі розглянуто предметну область зон рекуперації постійного струму та обґрунтовано потребу кількісного оцінювання складності формалізованих описів КПМ/КПС для порівняння альтернативних варіантів моделей.

Наведено метрики Холстеда та пояснено адаптацію для формульних подань через правила класифікації операторів і операндів. Описано базові величини n_1, n_2, N_1, N_2 і похідні метрики V, D, E, а також умови коректного застосування метрик для зіставлення формул.

Показано специфіку LaTeX як джерела даних: наявність вкладених конструкцій, середовищ вирівнювання та нестандартних команд, що робить нестійким суто текстовий підхід і підсилює роль структурного подання виразу.

Розглянуто особливості `.docx`: математичний запис зберігається у вигляді OMML у складі OOXML, що ускладнює пряме вилучення та підштовхує до конвертації або окремого розбору XML-структур.

Сформульовано вимоги до інструментів аналізу та обґрунтовано вибір підходу, де основою є деревоподібне семантичне подання формули, а нормалізація забезпечує відтворюваність і зіставність результатів.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ ДЛЯ АНАЛІЗУ СКЛАДНОСТІ ФОРМУЛ

3.1 Опис системи та функціональні можливості

У межах роботи розроблено настільний програмний засіб (далі – застосунок), призначений для автоматизованого аналізу складності формалізованих описів КПМ/КПС зон рекуперації постійного струму, поданих у вигляді формул. Основна ідея полягає в тому, що формули розглядаються як структуровані вирази: після вилучення та нормалізації вони перетворюються у деревоподібне подання, що дає можливість стабільно підраховувати оператори й операнди та обчислювати метрики Холстеда.

Застосунок підтримує такі функціональні можливості:

- імпорт джерел у форматах `.tex` і `.docx` та побудова переліку формул для подальшої роботи;
- автоматизоване вилучення формул з документа, збереження прив'язки до джерела та базових метаданих;
- редагування LaTeX-подання формули (виправлення дрібних помилок, уніфікація запису, перейменування);
- запуск аналізу: класифікація операторів/операндів, обчислення n_1, n_2, N_1, N_2 і похідних метрик V, D, E;
- сортування та фільтрація формул за вибраною метрикою, а також агрегування результатів на рівні документа;
- візуалізація результатів у вигляді діаграм та теплової карти (heatmap) для швидкого порівняння;
- збереження та відновлення робочої сесії у файлі `.fmx` (JSON) з версіонуванням формату;
- підтримка двомовного інтерфейсу (EN/UK) з перемиканням мови під час роботи.

Архітектурною основою застосунку обрано Electron як платформу для настільного застосунку та Svelte/TypeScript як інструменти реалізації інтерфейсу і прикладної логіки [18–20].

3.2 Архітектурні засади та структура застосунку

3.2.1 Загальна схема Electron (Main/Renderer)

Electron-застосунок складається з двох типів процесів: головного процесу (Main), який керує життєвим циклом застосунку та доступом до системних ресурсів, і процесів відображення (Renderer), які відповідають за інтерфейс користувача [18].

Головний процес виконує такі задачі:

- керування вікном застосунку та налаштування середовища виконання;
- робота з файловою системою (відкриття файлів, збереження `.fmx`);
- запуск важких обчислень у фоновому виконанні (через worker thread) та передача результатів у інтерфейс.

Процес відображення виконує:

- показ переліку формул, редактора та панелей аналітики;
- запуск операцій через IPC-інтерфейси (без прямого доступу до Node.js-API);
- керування станом UI (вибір формул, видимість серій на графіках, локалізація тощо) [20].

3.2.2 IPC-взаємодія та безпечний обмін даними

Обмін даними між Main та Renderer реалізовано через IPC (міжпроцесну взаємодію). У Electron для цього використовуються `ipcMain` та `ipcRenderer`, які передають повідомлення через канали з визначеними іменами [21].

З міркувань безпеки застосунок налаштовано з увімкненою ізоляцією контексту (context isolation) та з використанням `contextBridge` у preload-скрипті. Такий підхід обмежує доступ інтерфейсу до привілейованих API та дозволяє відкривати лише “білий список” функцій, потрібних для роботи застосунку [22–25].

У результаті процес відображення не викликає системні операції напряму. Замість цього він звертається до обгорток, оголошених у preload-скрипті, а головний процес вже виконує файлові операції, конвертацію або аналіз і повертає структурований результат.

3.2.3 Окремий worker-thread для аналізу

Обчислення метрик (парсинг, обходи дерева, підрахунки) може бути ресурсомістким на великій вибірці формул. Щоб інтерфейс залишався чуйним, аналіз винесено в окремий потік виконання на основі `worker_threads` Node.js [26].

Такий поділ забезпечує:

- відсутність "підвисання" інтерфейсу під час аналізу;
- можливість показувати прогрес/стан операції в UI;
- ізоляцію обчислювальних помилок (проблемна формула не має зупиняти весь застосунок).

3.3 Структури даних і формат збереження сесій

3.3.1 Модель "Формула"

У застосунку основною одиницею даних є сутність "Формула". Вона зберігає:

- `id` – унікальний ідентифікатор;
- `name` – назва/мітка (для зручності посилання в UI);
- `latex` – нормалізоване LaTeX-подання;
- `source` – дані про джерело (тип файлу, ім'я, позиція/номер у документі);
- `analysis` – результати аналізу (базові лічильники та метрики);
- `metadata` – додаткові поля (примітки, група, час аналізу, статуси валідації).

Результати аналізу зберігаються у вигляді:

- `operators`: словник частот операторів;
- `operands`: словник частот операндів;
- `halstead`: n_1, n_2, N_1, N_2, V, D, E;
- `warnings`: список попереджень (наприклад, нестандартні конструкції, часткова обробка).

3.3.2 Формат `.fmx` (JSON) та версіонування

Для збереження сесій використовується формат `.fmx`, який є JSON-документом. Він містить:

- версію формату (наприклад, `schemaVersion`);
- список формул з усіма полями (включно з результатами аналізу);
- налаштування інтерфейсу (мова, видимість графіків, сортування);

- інформацію про джерело/джерела, з якими працювали.

Версіонування необхідне для сумісності під час розвитку застосунку: за зміни структури `.fmx` передбачено міграцію даних або коректне повідомлення користувача про несумісність з поточною версією.

3.3.3 Сценарії збереження/відновлення та обробка помилок

Передбачено такі типові сценарії:

- “Зберегти файл” – запис `.fmx` у файл, з перевіркою доступності шляху та коректності JSON;
- “Відкрити файл” – завантаження `.fmx`, валідація `schemaVersion`, відновлення переліку формул і стану інтерфейсу;
- “Відновлення після помилки” – якщо частина формул не пройшла аналіз або містить попередження, вони відмічаються в UI, але дані сесії не втрачаються.

Усі критичні помилки (некоректний формат, пошкоджений файл, неможливість запису) супроводжуються повідомленням користувачу в інтерфейсі (див. підрозділ про сповіщення).

3.4 Реалізація конвеєра обробки

3.4.1 Завантаження файлу та первинна валідація

Після вибору файлу застосунок визначає тип джерела (`.tex` або `.docx`) та виконує первинну перевірку:

- доступність для читання;
- базові обмеження розміру/розширення;
- можливість запуску конвертації (для `.docx`).

Далі формується список “кандидатів” на формули з мінімальними метаданими (ідентифікатор, позиція, фрагмент тексту).

3.4.2 Вилучення формул з `.tex`

Для `.tex` виконується пошук математичних фрагментів (inline і display) та типових середовищ. Вилучення виконується обережно, з урахуванням вкладених дужок і команд, щоб не “обрізати” формулу по першому символу-розділювачу. Після вилучення кожна формула проходить нормалізацію та додається до переліку.

3.4.3 Вилучення формул з `.docx`

Для `.docx` застосовується конвертація через Pandoc: документ переводиться у LaTeX-подання, після чого вилучення формул виконується тим самим механізмом, що й для `.tex` [8].

Оскільки відомі випадки, коли конвертація може породжувати некоректні фрагменти, передбачено фіксацію попереджень і можливість ручного редагування формул у UI.

3.4.4 Brace-aware очищення та нормалізація

Перед семантичним аналізом формула проходить “легке” очищення:

- перевірка балансу фігурних дужок `{}` та базових розділювачів;
- уніфікація пробілів і переносів;
- видалення/заміну символів, що не несуть семантики, але заважають парсингу;
- відмітка “підозрілих” випадків (наприклад, невідомі команди).

Очищення виконується з урахуванням вкладеності дужок (brace-aware), щоб не зруйнувати структуру виразу.

3.4.5 Токенізація та класифікація

Після нормалізації формула перетворюється у деревоподібне подання MathJSON за допомогою парсера, який підтримує LaTeX-ввід [9]. Далі виконується рекурсивний обхід дерева та збір токенів.

Класифікація елементів ґрунтується на такому правилі:

- оператори – математичні операції, відношення, логічні зв’язки, функціональні застосування та конструкції, що задають перетворення/залежності;
- операнди – позначення сутностей моделі, параметри, індекси, константи, числові значення.

На виході формується частотний розподіл операторів та операндів, а також базові лічильники n_1, n_2, N_1, N_2.

3.4.6 Обчислення метрик Холстеда та агрегація по документу

Після отримання базових лічильників застосунок обчислює:

$$V = N * \log_2(n), \tag{3.1}$$

$$D = \left(\frac{n_1}{2}\right) * \left(\frac{N_2}{n_2}\right), \tag{3.2}$$

$$E = D * V \tag{3.3}$$

Окремо формується агрегована статистика документа:

- мінімальні/максимальні значення метрик;
- середні значення;
- ранжування формул за обраною метрикою;
- розподіли для побудови графіків.

Усі проміжні й фінальні результати зберігаються в структурі сесії і можуть бути збережені у `.fmx`.

3.5 Опис інтерфейсу користувача розробленої програми

У цьому підрозділі наведено короткий опис інтерфейсу застосунку, достатній для розуміння логіки роботи, сценаріїв користувача та відповідності реалізованих функцій поставленим завданням.

3.5.1 Загальна концепція інтерфейсу

Інтерфейс організовано у вигляді панелі керування (dashboard) з двома основними областями: ліва панель для роботи зі списком формул і права панель для аналітики та візуалізації. Такий поділ дозволяє одночасно бачити перелік формул і результат аналізу вибраного документа.

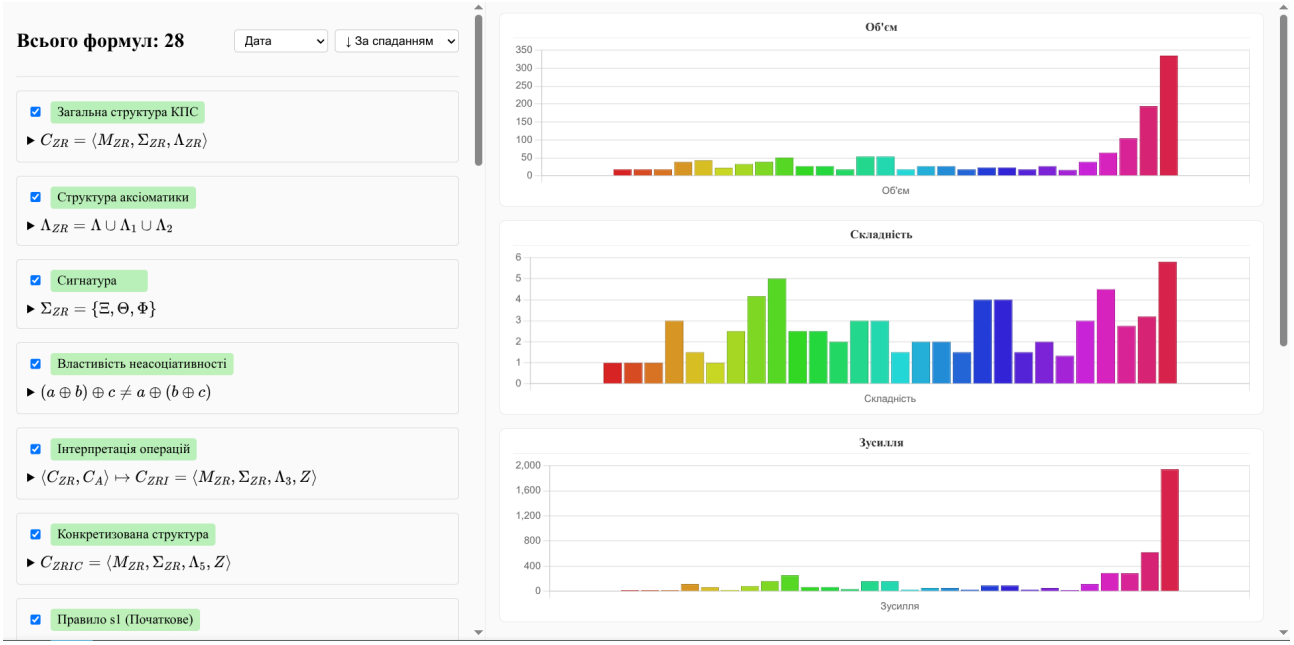

Рисунок 3.1 – Головне вікно програми

На рис. 3.1 показано загальний вигляд головного вікна: ліворуч розміщено список формул і керування, праворуч – блок узагальнених показників та графіки.

3.5.2 Ліва панель: список формул (Formula Cards) та керування

Ліва панель містить перелік формул у вигляді карток. Кожна картка відображає:

- назву формули;
- короткий фрагмент LaTeX;
- ключові метрики або статус (проаналізовано/попередження/помилка).

Передбачено такі операції:

- вибір формули для перегляду деталей;
- перемикання видимості формули на графіках (чекбокс/перемикач);
- перейменування формули для зручності посилань у дослідженні;
- додавання/видалення формул у межах поточної сесії;
- редагування LaTeX та повторний запуск аналізу для вибраної формули.

Рисунок 3.2 – Ліва панель

На рис. 3.2 наведено приклад відображення списку формул і базових дій: вибір, керування видимістю, перейменування та перехід до редагування.

3.5.3 Права панель: аналітика документа та графіки

Права панель складається з двох частин:

- коротке резюме документа (кількість формул, статистичні показники метрик);
- блок візуалізації: три графіки для V, D, E та теплову карту для E.

Візуалізація побудована на Chart.js, який дозволяє інтерактивно керувати серіями даних і оновлювати графік без перезавантаження інтерфейсу [27, 28].

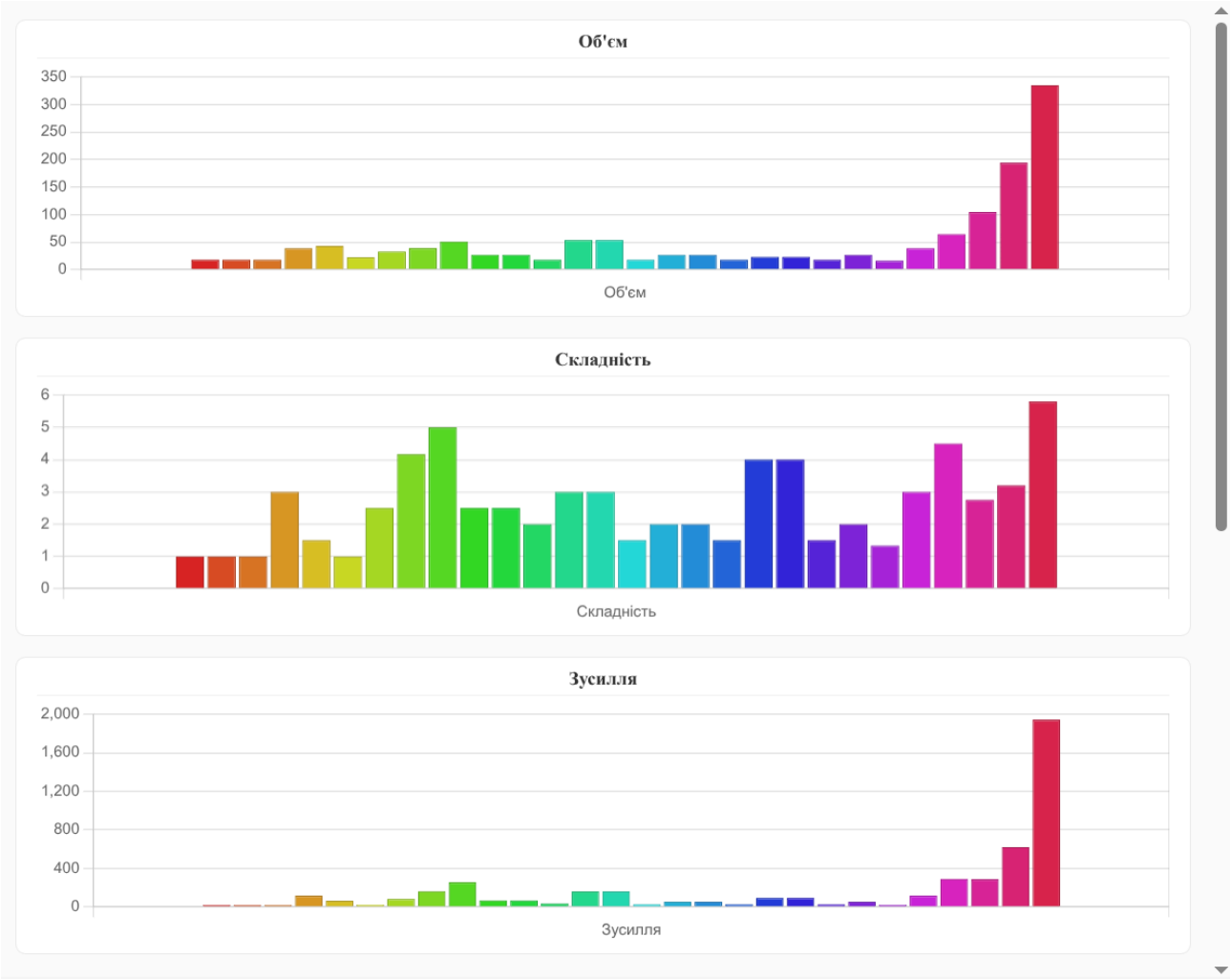

Рисунок 3.3 – Права панель

На рис. 3.3 показано приклад відображення узагальнених показників і графіків, за допомогою яких виконується порівняння формул.

3.5.4 Кольорові позначення та інтерпретація кольорів

Кольором позначаються за значенням E (зусилля за Холстедом). Її призначення – швидко виділяти формули з найбільшими значеннями E у межах документа. Кольорова шкала інтерпретується як відносна складність: менші значення відображаються зеленішими відтінками, більші – червонішими.

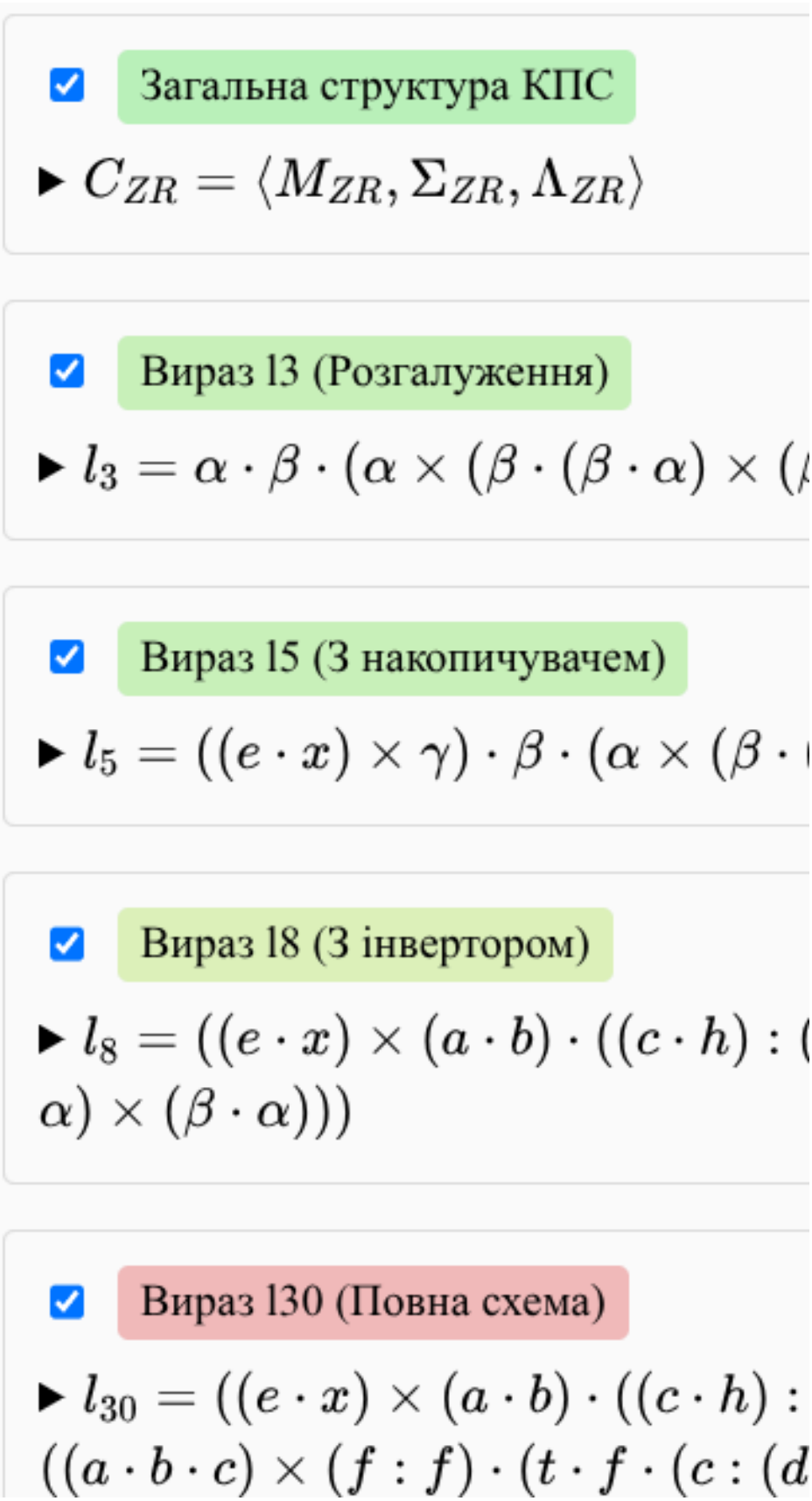

Рисунок 3.4 – Кольорові позначення основані на метриці зусилля для формул документа

На рис. 3.4 наведено приклад, де найбільш червоні формули – ті що з найбільшими значеннями E.

3.5.5 Легенда графіків і керування видимістю формул

Для зручності аналізу реалізовано синхронізацію стану видимості:

- перемикання формули у списку;
- взаємодія з легендою на графіках.

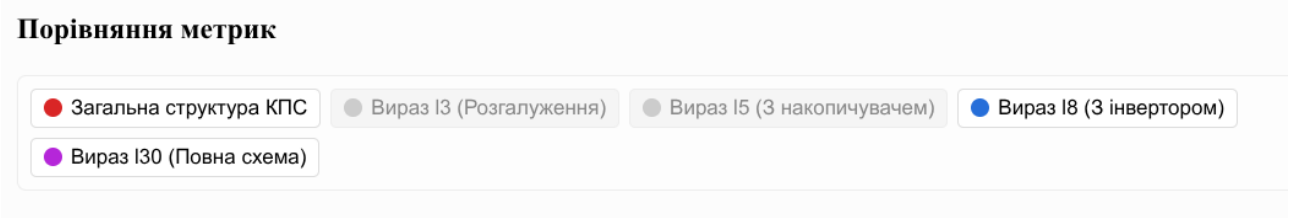

Рисунок 3.5 – Легенда графіків з можливістю вмикати та вимикати відображення окремих формул

На рис. 3.5 показано легенду, за допомогою якої виконується вибір формул для відображення на графіках порівняння метрик.

Після зміни видимості формули (через чекбокс або кліком по легенді) застосунок узгоджує цей стан на всіх графіках, щоб користувач бачив один і той самий набір формул у всіх діаграмах. Chart.js підтримує керування видимістю даних і події легенди, що дозволяє реалізувати описаний механізм [27, 28].

3.5.6 Блок «Загальні метрики»

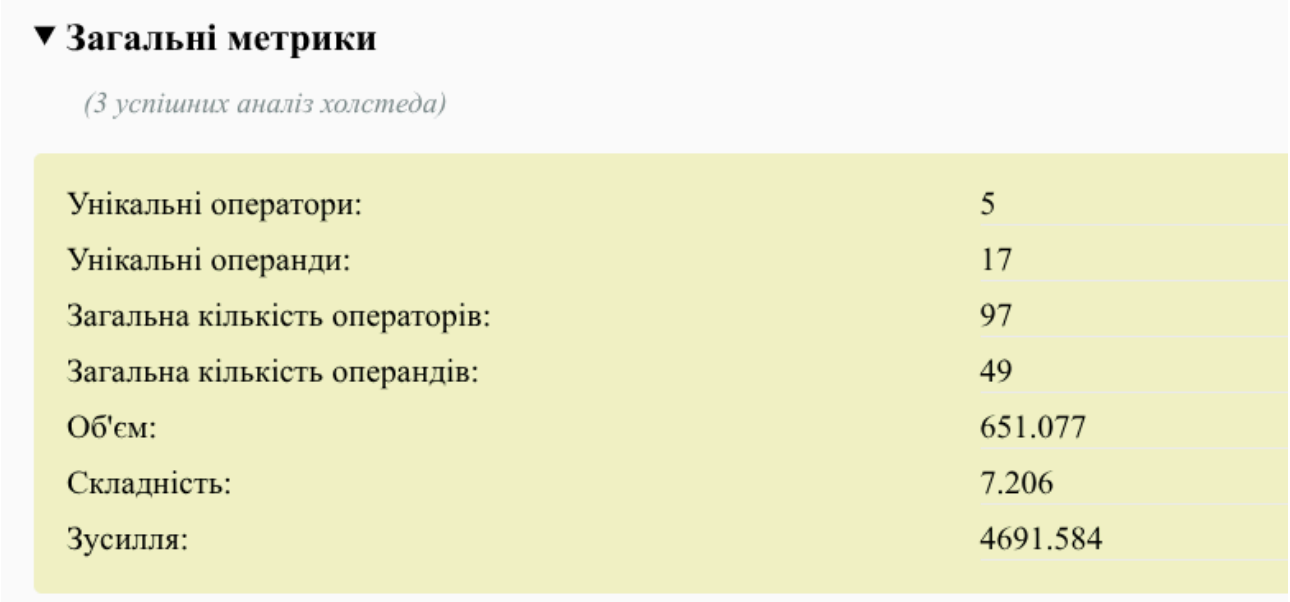

Рисунок 3.6 – Відображення загальних значень метрик Холстеда для вибраного документа

На рис. 3.6 наведено приклад блоку, у якому показано n_1, n_2, N_1, N_2, а також похідні метрики V, D, E.

3.5.7 Кнопка «Додати формулу»

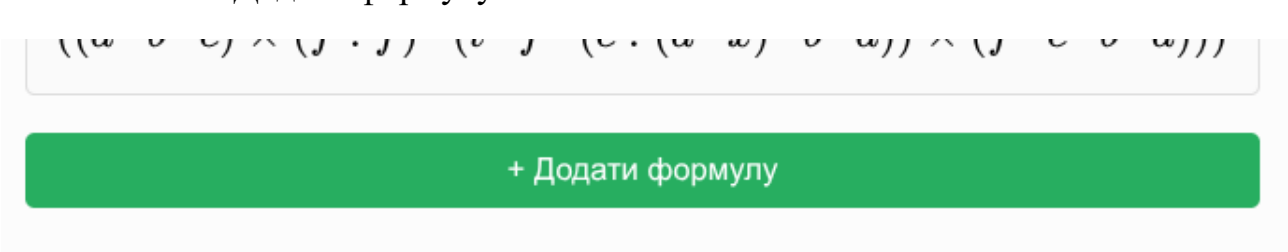

Рисунок 3.7 – Кнопка додавання нової формули до поточної сесії

На рис. 3.7 показано елемент керування, який додає нову формулу до переліку для подальшого редагування та аналізу.

3.5.8 Кнопка видалення формули

Рисунок 3.8 – Елемент керування для видалення формули з переліку

На рис. 3.8 показано кнопку видалення, яка прибирає вибрану формулу з поточної сесії.

3.5.9 Розгортання формули для редагування LaTeX та перегляду метрик

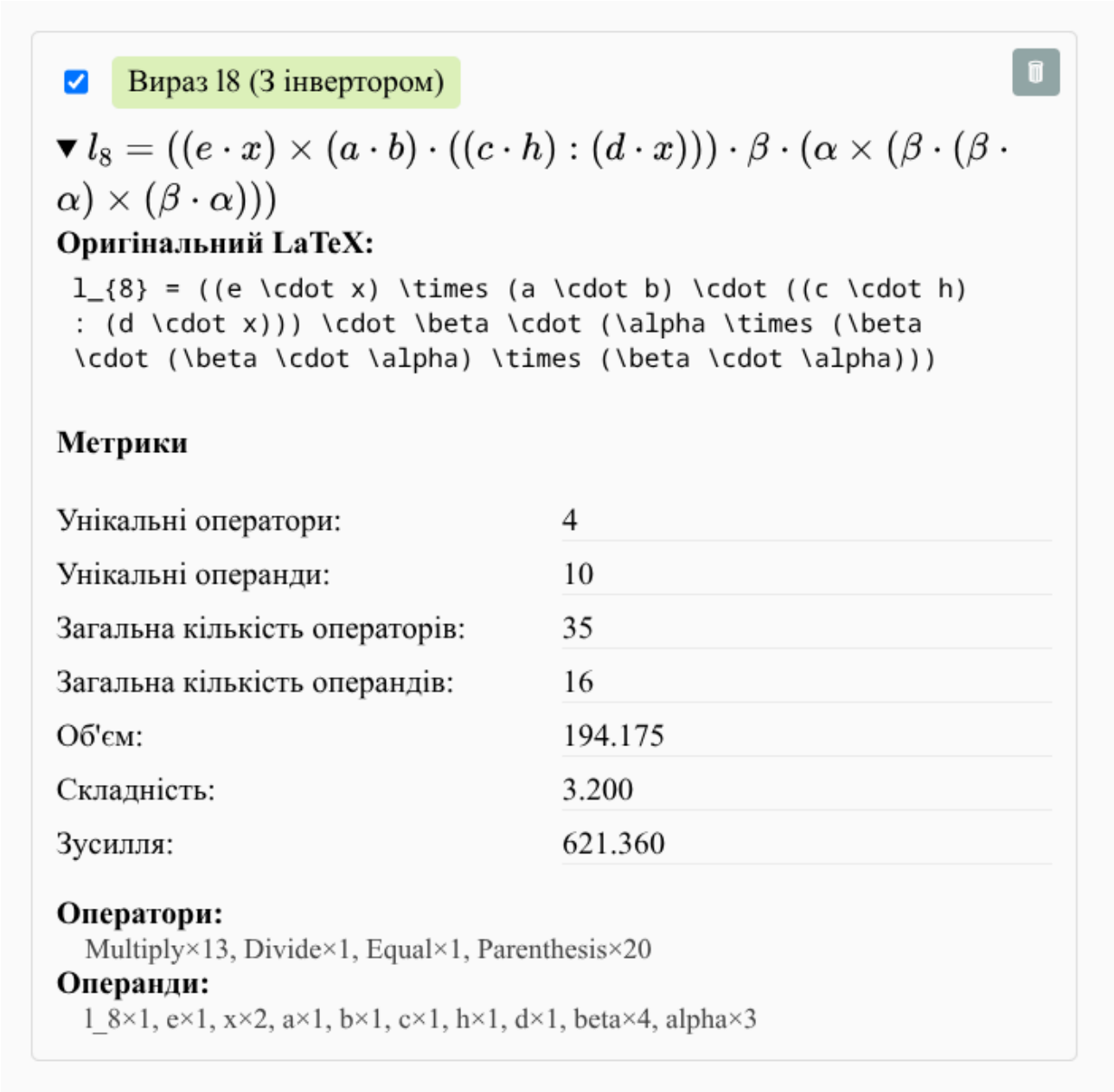

Рисунок 3.9 – Розгорнутий вигляд формули

На рис. 3.9 наведено розгорнутий вигляд картки формули з можливістю перегляду та редагування оригінального LaTeX та результати аналізу.

3.5.10 Сповіщення та повідомлення користувачу

Застосунок використовує короткі сповіщення для інформування про завершення операцій та помилки. Зокрема, виводяться повідомлення:

- про помилки читання файлу/конвертації/парсингу;
- про наявність попереджень (часткова обробка або підозрілі фрагменти).

Це дає можливість не перевантажувати інтерфейс технічними деталями, але водночас не приховувати факт проблеми, якщо вона впливає на коректність результату.

3.5.11 Підтримка мов (EN/UK) та перемикання

Інтерфейс застосунку локалізовано двома мовами: англійською (EN) та українською (UK). Локалізуються:

- назви кнопок, меню та підказок;
- підписи осей і легенди на графіках;
- тексти повідомлень про помилки та сповіщення.

Перемикання мови змінює підписи без перезавантаження сесії, тобто всі поточні дані аналізу зберігаються.

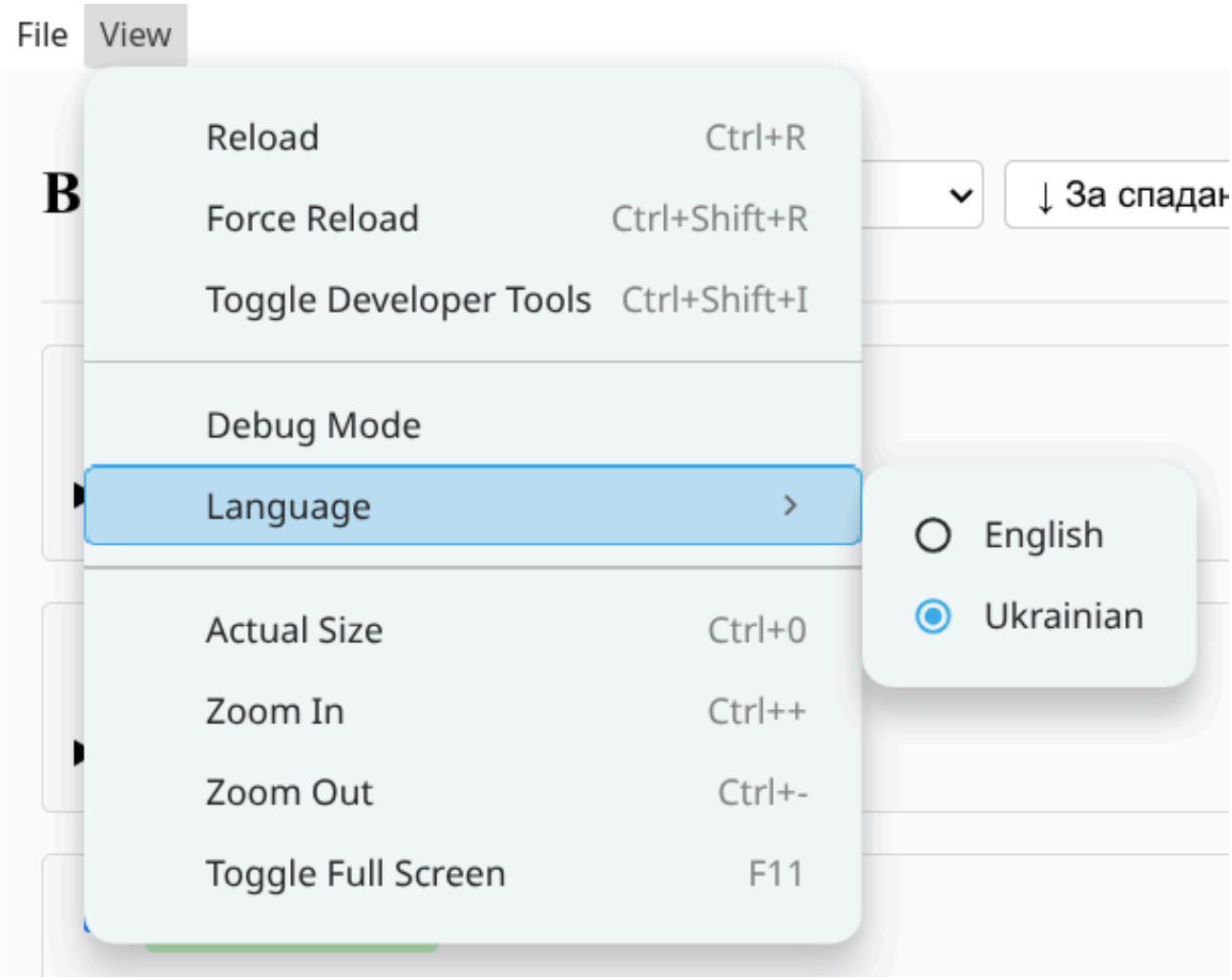

Рисунок 3.10 – Перемикання мови інтерфейсу

На рис. 3.10 показано приклад елемента керування мовою інтерфейсу.

3.6 Висновки до розділу 3

У розділі описано розроблений програмний засіб для автоматизованого аналізу складності формульних описів КПМ/КПС зон рекуперації постійного струму.

Наведено перелік реалізованих функціональних можливостей: імпорт `.tex`/`.docx`, редагування формул, розрахунок метрик Холстеда, візуалізація та збереження сесій у `.fmx`.

Обґрунтовано архітектуру на базі Electron з поділом на Main/Renderer процеси, а також застосування IPC-обміну через preload-скрипт з урахуванням рекомендацій безпеки (context isolation, contextBridge).

Показано, що обчислювальна частина винесена у worker thread, що зберігає чуйність інтерфейсу під час обробки великих наборів формул.

Описано структури даних та формат `.fmx`, включно з версіонуванням і базовими сценаріями відновлення після помилок.

Подано опис інтерфейсу користувача та ключових елементів взаємодії: список формул, аналітика, синхронізація видимості серій на графіках, сповіщення, а також підтримка двомовності.

4 ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ ПРОГРАМНОГО ЗАСОБУ

4.1 Мета та задачі тестування

Тестування у роботі виконувало дві основні ролі. По-перше, воно підтверджувало коректність ключових обчислень (підрахунок операторів/операндів і формули метрик Холстеда). По-друге, воно зменшувало ризик регресій під час розвитку конвеєра обробки: вилучення формул, нормалізація, семантичний розбір, агрегація результатів, а також обробка помилок на “складних” вхідних даних.

Задачі тестування:

- перевірити коректність вилучення формул з LaTeX для різних типів математичних конструкцій (inline, display, середовища з вирівнюванням, вкладені структури);
- перевірити валідацію та очищення LaTeX (баланс дужок, розділювачів, допустимість команд, сумісність із рендерингом);
- перевірити класифікацію операторів/операндів для характерних символів і команд (грецькі літери, “двокрапка” як операція ділення, множення `\cdot`/`\times`, спеціальні оператори);
- перевірити стабільність конвеєра “вилучення → аналіз” на документі з кількома формулами, включно з випадками частково некоректного вводу;
- перевірити поведінку застосунку у випадках помилок та обмежень (непідтримуваний тип файлу, проблемні формули після конвертації, великі набори формул);
- підготувати основу для UI-тестування ключових сценаріїв (відкриття файлу, повідомлення про помилки, відмова користувача від вибору файлу), з фіксацією поточного обмеження інструментів тестування для Svelte 5.

4.2 Підхід і види тестування

Тести додавалися поступово, паралельно з реалізацією функцій. Для кожного дефекту або нестабільного випадку спочатку фіксувався відтворюваний приклад, після чого додавався тест, який підтверджував виправлення. Такий підхід зручний для інструменту, що працює з “живими” формулами: більшість складних ситуацій проявляється лише на конкретних прикладах.

Запуск тестів виконується командою `npm run test` на базі Jest. Для TypeScript використовується ts-jest, а виконання налаштовано з урахуванням ESM-модулів (режим Node.js з експериментальними VM Modules) [29, 30].

4.2.1 Модульне тестування

Модульні тести охоплюють окремі частини конвеєра.

1) Вилучення формул з LaTeX.

Перевіряється робота `extractMath` на стандартних і "складних" прикладах:

- inline-математика в `\(...\)` та варіанти з пробілами;
- кілька формул у одному рядку;
- display-математика з розривами рядків;
- середовища на кшталт `array` і багаторядкові вирази;
- поведінка на порожньому тексті, тексті без математики, некоректно закритих фрагментах;
- усунення дублювань формул після нормалізації пробілів.

Ці перевірки зосереджені на тому, щоб вилучення було стабільним і не завершувалося аварійно на помилковому або неповному вводі (фіксується принцип: "не зламатися, а повернути контрольований результат").

2) Перевірка кутових дужок і спеціальних конструкцій. Окремий набір тестів присвячений виразам з `\langle`/`\rangle`, а також варіантам `\left\langle...\right\rangle`, включно з вкладеними конструкціями та формулами з індексами. Такі фрагменти були критичними для реальних документів і потребували окремої фіксації.

3) Валідація та очищення LaTeX. Перевіряються допоміжні функції:

- баланс звичайних і фігурних дужок;
- баланс `\left...\right` та відповідність пар дужок;
- перевірка допустимості команд (відсікання нецільових середовищ, посилань, команд, що не повинні потрапляти у формульний аналіз);
- очищення формули перед аналізом та сумісність із рендерингом у KaTeX.

Окремо перевіряється, що "коректні" приклади успішно рендеряться (рендеринг використано як додатковий індикатор того, що очищення не спотворює синтаксис).

4) Обчислення метрик Холстеда.

Тести на `analyzeFormula` перевіряють:

- коректну роботу на простих і складних формулах;
- трактування “двокрапки” як оператора ділення;
- розпізнавання множення (`\cdot`, `\times`) як єдиного класу операцій;
- облік грецьких літер як операндів;
- підтримку доданих символів і логіки, що виникали під час розширення парсингу.

Для складних формул додатково перевіряється, що отримані лічильники мають очікуваний порядок величин і що набір операторів містить потрібні класи (це дозволяє швидко виявляти регресії в класифікації).

4.2.2 Інтеграційне тестування (pipeline/IPC/worker)

Інтеграційні тести спрямовані на перевірку зв’язки “вилучення → аналіз” як єдиного процесу:

- на LaTeX-документі з текстом та різними типами формул виконується вилучення, а далі пакетний аналіз (перевіряється, що для кожної вилученої формули є рядок результату та обчислені загальні метрики документа);
- перевіряється стійкість до частково некоректних формул: аналіз не зупиняється повністю, а коректні фрагменти продовжують оброблятися;
- виконується перевірка на продуктивність для документа з великою кількістю однотипних формул, щоб контролювати час проходження конвеєра в межах прийнятного інтервалу.

Окремо зафіксовано тест для `extractFormulasFromFile`, який перевіряє реакцію на непідтримуваний тип файлу та повідомлення про помилку. Тести, що напряму залежать від запуску Pandoc і від системного вводу-виводу, не дублюються на рівні модулів: такі сценарії охоплені інтеграційними перевірками, оскільки детальне мокування запуску Pandoc та файлових операцій є нестійким і надмірно ускладнює тестовий код.

4.2.3 UI-тестування компонентів

Підготовлено набір тестів, орієнтований на перевірку сценаріїв інтерфейсу: відкриття файлу, обробка підтримуваних/непідтримуваних типів, показ повідомлень про помилки, обробка скасування діалогу вибору файлу.

Однак у поточній версії частина цих UI-тестів позначена як пропущена. Причина є технічною: компоненти Svelte 5 з runes наразі мають обмеження сумісності з `@testing-library/svelte`, тому автоматизована перевірка UI була обмежена і зведена до підготовки сценаріїв та моків, які можуть бути активовані після оновлення інструментів тестування.

4.3 Тестові сценарії та контрольні приклади

4.3.1 Контрольні формули для перевірки токенізації

Для токенізації та класифікації використовуються невеликі формули, де легко відстежити очікувані оператори/операнди:

- базові арифметичні вирази;
- вирази з функціями (`\sin`, `\sqrt`);
- вирази з індексами та степенями;
- вирази з грецькими літерами;
- оператори, важливі для КПМ-описів (множення, ділення, рівність, групування).

Для таких прикладів тести фіксують, що вираз успішно проходить семантичний розбір і що підрахунок операндів не “втрачає” позначення на кшталт `\alpha`, `\beta`.

4.3.2 Складні LaTeX-випадки (матриці, вкладені групи, align)

Окремо перевіряються багаторядкові display-формули та середовища, де структура визначається розділювачами (`\\`, `&`) і вкладеністю:

- `array` у межах `\[...\]`;
- поліноми та довгі вирази з перенесеннями;
- вкладені групи з дужками.

Метою цих тестів є не ідеалізація формул, а контроль того, що вилучення бере формулу цілісним блоком, не відкидаючи частини середовища.

4.3.3 Випадки з пунктуацією та brace-aware очищенням

Для очищення й нормалізації перевіряються приклади, де помилка в дужках або зайвий символ може зруйнувати подальший парсинг:

- незбалансовані дужки;
- некоректні `\left...\right` пари;

- команди, що не повинні потрапляти в формульний аналіз (неформульні середовища, посилання, цитування).

Тести підтверджують, що такі випадки або коректно відсікаються, або позначаються як некоректні без аварійного завершення конвеєра.

4.3.4 Випадки відмови AST і робота резервного оброблення

У тестовому наборі присутні сценарії з “поганими” командами та некоректними фрагментами, які не мають коректного семантичного подання. Для таких випадків перевіряється, що:

- на рівні документа продовжується обробка інших формул;
- для проблемних фрагментів фіксуються помилки або попередження;
- конвеєр аналізу залишається працездатним у цілому.

На практиці саме такі сценарії найчастіше з’являються після конвертації `.docx` або при роботі з документами, де формули записані неоднорідно.

4.4 Засоби тестування та результати

Для автоматизації тестування використано Jest як основний фреймворк. TypeScript-тести виконуються через ts-jest, що дозволяє запускати перевірки без попередньої ручної збірки тестів у JavaScript [29, 30].

Поточний набір тестів охоплює:

- модульні перевірки вилучення формул та обробки спеціальних випадків;
- модульні перевірки обчислень метрик Холстеда;
- інтеграційні перевірки повного конвеєра;
- підготовлені сценарії UI-тестів (частково пропущені через обмеження інструментів).

Під час запуску тестового набору всі активні (не пропущені) перевірки завершуються успішно. Пропущені сценарії використовуються як заготовки для майбутнього розширення тестування інтерфейсу та не впливають на перевірку коректності конвеєра обробки.

4.5 Налагодження та обробка помилок

Налагодження під час розробки зосереджувалося на типових проблемах, характерних для формульного вводу.

1. Конвертація `.docx` → LaTeX. Найбільш проблемними є випадки, коли після конвертації з'являються артефакти або некоректні команди. Для таких ситуацій застосунок не зупиняє аналіз повністю: формули, які не проходять валідацію або семантичний розбір, позначаються як проблемні, а решта вибірки аналізується далі. Можливість редагування формули у UI використовується як інструмент точкового виправлення без повторного імпорту всього документа.
2. Некоректний або неповний LaTeX. Для фрагментів з незбалансованими дужками або помилковими розділювачами застосовуються перевірки балансу та правила відсікання. Мета цих перевірок – не "виправляти зміст", а гарантувати передбачувану поведінку: або коректне опрацювання, або контрольована помилка з поясненням.
3. Великі файли та масиви формул. Для таких випадків обчислення виконуються у worker thread, а інтеграційні тести додатково контролюють, що конвеєр завершується за прийнятний час. Це знижує ризик того, що застосунок стане непридатним на реальних вибірках.
4. Непідтримувані формати. На рівні обробки файлів реалізовано ранню перевірку розширення. У випадку помилки користувач отримує зрозуміле повідомлення, а сесія не пошкоджується.

4.6 Висновки до розділу 4

Тестування було інтегроване в процес розробки та використовувалося як основний механізм контролю коректності аналізу формул.

Модульні тести забезпечили стабільність вилучення формул і обробки спеціальних випадків (кутові дужки, багаторядкові конструкції, баланс `\left... \right`), а також перевірили ключові компоненти валідації LaTeX.

Окремий набір тестів підтвердив коректність класифікації операторів/операндів і розрахунку метрик Холстеда на контрольних прикладах та на формулах підвищеної складності.

Інтеграційні тести перевірили повний маршрут від вилучення до аналізу і показали стійкість до частково некоректного вводу: наявність проблемних фрагментів не блокує обробку документа в цілому.

UI-сценарії тестування підготовлено, однак частина з них у поточній конфігурації позначена як пропущена через обмеження сумісності інструментів тестування зі Svelte 5 runes.

Залишкові ризики пов’язані переважно з неоднорідністю реальних документів (особливо після конвертації з `.docx`) та появою нестандартних команд; ці випадки компенсуються валідацією, попередженнями та можливістю точкового редагування формул у застосунку.

5 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

5.1 План експерименту та критерії оцінювання

Метою експериментального дослідження було отримати кількісні оцінки складності формул, що описують елементи КПС/КПМ зон рекуперації постійного струму, та виконати порівняльний аналіз отриманих значень.

Як критерії оцінювання використано метрики Холстеда: об'єм V, складність D та зусилля E [5]. Інтерпретація показників у межах цієї роботи така:

- V відображає інформаційну насиченість формульного опису (скільки "матеріалу" містить запис);
- D характеризує структурну складність запису, що зростає за збільшенням набору операцій та відносної різноманітності операндів;
- E є інтегральним показником, який поєднує V і D та зручний для ранжування формул за очікуваною трудомісткістю аналізу.

Експеримент виконувався у такій послідовності:

- формування вибірки формул з джерельного документа та збереження їх у сесії `.fmx`;
- обчислення n_1, n_2, N_1, N_2, а також V, D, E для кожної формули;
- побудова розподілів метрик та виділення формул з найбільшими значеннями;
- детальний розгляд результатів по кожній формулі та узагальнення висновків.

5.2 Опис набору даних

Експеримент виконано на вибірці з 28 формул, отриманих зі статті, підготовленої у форматі `.docx`. Вибірка містить як короткі означення, так і розгорнуті формули правил та перетворень, що відображає реальні сценарії роботи з формалізованими описами.

За призначенням формули у вибірці можна умовно поділити на групи:

- означення та загальні співвідношення;
- правила та продукції;
- виведення/перетворення;
- окремі випадки підвищеної складності.

5.3 Умови проведення експерименту

Обчислення виконувалися в середовищі настільного застосунку на базі Electron. Аналіз формул проводився за єдиними правилами класифікації операторів і операндів, описаними у попередніх розділах. Для семантичного подання виразів використовувалось деревоподібне представлення MathJSON, отримане з LaTeX-вводу.

Коректність і стабільність реалізації контролювалася автоматизованими тестами (модульними та інтеграційними) на базі Jest/ts-jest, які додавалися під час розробки та фіксували критичні приклади з практичних документів.

5.4 Результати обчислення метрик Холстеда

Для зручності аналізу розглянуто розподіли V, D та E окремо. Узагальнені статистичні характеристики по вибірці:

- $V_{\{\min\}} = 16.253$, $V_{\{\text{median}\}} = 27.000$, $V_{\{\text{mean}\}} = 49.880$, $V_{\{\max\}} = 335.047$;
- $D_{\{\min\}} = 1.000$, $D_{\{\text{median}\}} = 2.500$, $D_{\{\text{mean}\}} = 2.580$, $D_{\{\max\}} = 5.800$;
- $E_{\{\min\}} = 18.095$, $E_{\{\text{median}\}} = 67.500$, $E_{\{\text{mean}\}} = 177.226$, $E_{\{\max\}} = 1943.274$.

5.4.1 Розподіл об’єму V

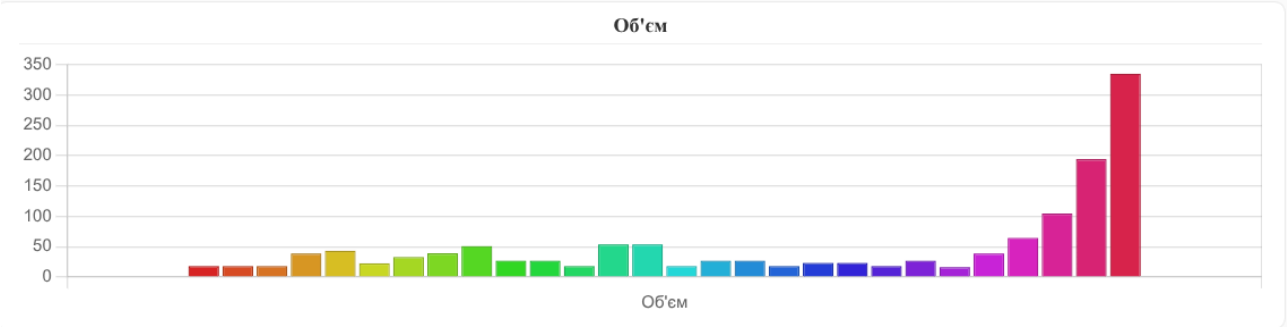

Рисунок 5.1 – Розподіл значень об’єму для формул вибірки.

На рис. 5.1 видно, що більшість формул має порівняно невеликі значення V, що відповідає коротким означенням та локальним правилам. Водночас присутні виражені пікові значення, які утворюють розгорнуті формули зі значною кількістю операцій і повторюваних елементів.

Найбільші значення V у вибірці:

- `Вираз l30 (Повна схема)` (ідентифікатор `l30`): $V = 335.047$;
- `Вираз l8 (З інвертором)` (ідентифікатор `l8`): $V = 194.175$;
- `Вираз l5 (З накопичувачем)` (ідентифікатор `l5`): $V = 104.608$.

Для таких формул характерні довгі ланцюжки множення/ділення та велика кількість групувань дужками, що збільшує N і, відповідно, V.

5.4.2 Розподіл складності D

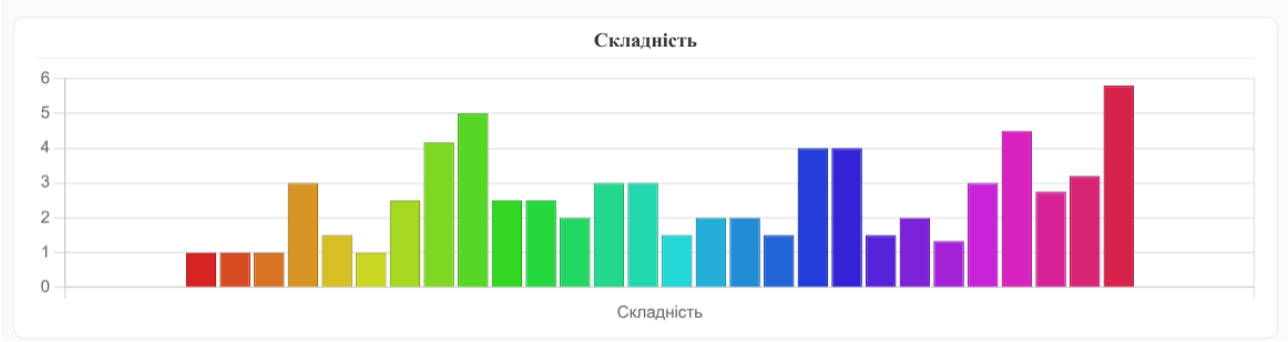

Рисунок 5.2 – Розподіл значень складності для формул вибірки.

На рис. 5.2 показано, що значення D у більшості випадків знаходяться у помірному діапазоні, проте виділяється група формул з підвищеною складністю. Такі значення виникають, коли у записі одночасно присутні різні типи операторів, а операнди використовуються у різних контекстах (індекси, параметри, повтори).

Найбільші значення D у вибірці:

- `Вираз l30 (Повна схема)` (ідентифікатор `l30`): $D = 5.800$;
- `Правило s3 (Розгалуження)` (ідентифікатор `s3`): $D = 5.000$;
- `Вираз l3 (Розгалуження)` (ідентифікатор `l3`): $D = 4.500$.

Показник D у цьому експерименті відділяє структурно насичені правила (де V може бути не максимальним) від простих означень.

5.4.3 Розподіл зусилля E та найбільш складні формули

Рисунок 5.3 – Розподіл значень зусилля для формул вибірки.

На рис. 5.3 значення E мають найбільш контрастний розподіл: основна маса формул зосереджена у нижній частині шкали, а кілька формул утворюють різко виражені максимуми. Це пояснюється тим, що E акумулює вплив і довжини запису, і структурної складності.

Найбільші значення E у вибірці:

- `Вираз l30 (Повна схема)` (ідентифікатор `l30`): $E = 1943.274$;
- `Вираз l8 (З інвертором)` (ідентифікатор `l8`): $E = 621.360$;
- `Вираз l3 (Розгалуження)` (ідентифікатор `l3`): $E = 290.808$.

Саме ці формули доцільно розглядати першими під час перевірки читабельності й підтримуваності: вони вимагають найбільшого зусилля для аналізу та порівняння.

5.5 Особливості роботи з `.docx` у межах експерименту

Оскільки джерельний матеріал отримано з `.docx`, суттєвим фактором є перетворення формул у LaTeX-подання для подальшого аналізу. У більшості випадків перетворення дає коректний результат, однак для окремих формул можливі артефакти (зайві символи, некоректні команди або розриви), які впливають на стабільність розбору.

Для зменшення впливу таких випадків застосовувалася валідація та нормалізація, а проблемні фрагменти позначалися попередженнями. Це забезпечує відтворюваність експерименту: одна й та сама вибірка формул аналізується за однаковими правилами, а відхилення фіксуються явно.

5.6 Детальний аналіз результатів для кожної формули

Нижче наведено таблиці з детальними результатами аналізу для всіх формул вибірки. Дані отримані зі збереженої сесії `.fmx` і включають базові лічильники, похідні метрики та частотні словники операторів і операндів.

Таблиця 5.1 – Результати аналізу виразу загальної структури КПС

Параметр	Значення
Вираз	$C_{\mathrm{ZR}} = \langle M_{\mathrm{ZR}}, \Sigma_{\mathrm{ZR}}, \Lambda_{\mathrm{ZR}} \rangle$
LaTeX-представлення	`C_{ZR} = \langle M_{ZR}, \Sigma_{ZR}, \Lambda_{ZR} \rangle`
Унікальні оператори, n_1	2
Унікальні операнди, n_2	4

Продовження таблиці 5.1

Параметр	Значення
Загальна кількість операторів, N_1	3
Загальна кількість операндів, N_2	4
Об’єм V	18.095
Складність D	1
Зусилля E	18.095
Оператори, частоти	Equal×1, AngleBracket×2
Операнди, частоти	C_{ZR}×1, Λ_{ZR}×1, M_{ZR}×1, Σ_{ZR}×1

Таблиця 5.2 – Результати аналізу виразу структури аксіоматики

Параметр	Значення
Вираз	$\Lambda_{\mathrm{ZR}} = \Lambda \cup \Lambda_1 \cup \Lambda_2$
LaTeX-представлення	`\Lambda_{ZR} = \Lambda \cup \Lambda_{1} \cup \Lambda_{2}`
Унікальні оператори, n_1	2
Унікальні операнди, n_2	4
Загальна кількість операторів, N_1	3
Загальна кількість операндів, N_2	4
Об’єм V	18.095
Складність D	1
Зусилля E	18.095
Оператори, частоти	Equal×1, Union×2
Операнди, частоти	Λ_{ZR}×1, Λ×1, Λ_1×1, Λ_2×1

Таблиця 5.3 – Результати аналізу виразу сигнатури

Параметр	Значення
Вираз	$\Sigma_{\mathrm{ZR}} = \{\Xi, \Theta, \Phi\}$
LaTeX-представлення	`\Sigma_{ZR} = \{ \Xi, \Theta, \Phi \}`
Унікальні оператори, n_1	2
Унікальні операнди, n_2	4
Загальна кількість операторів, N_1	3
Загальна кількість операндів, N_2	4
Об’єм V	18.095
Складність D	1
Зусилля E	18.095
Оператори, частоти	Equal×1, Brace×2
Операнди, частоти	Σ_{ZR}×1, Ξ×1, Θ×1, Φ×1

Таблиця 5.4 – Результати аналізу виразу властивості неасоціативності

Параметр	Значення
Вираз	$(a \oplus b) \oplus c \neq a \oplus (b \oplus c)$
LaTeX-представлення	`(a \oplus b) \oplus c \neq a \oplus (b \oplus c)`
Унікальні оператори, n_1	3
Унікальні операнди, n_2	3
Загальна кількість операторів, N_1	9
Загальна кількість операндів, N_2	6

Продовження таблиці 5.4

Параметр	Значення
Об’єм V	38.774
Складність D	3
Зусилля E	116.323
Оператори, частоти	NotEqual×1, CircleAdd×4, Parenthesis×4
Операнди, частоти	a×2, b×2, c×2

Таблиця 5.5 – Результати аналізу виразу інтерпретації операцій

Параметр	Значення
Вираз	$\langle C_{\mathrm{ZR}}, C_A \rangle \mapsto C_{\mathrm{ZRI}} = \langle M_{\mathrm{ZR}}, \Sigma_{\mathrm{ZR}}, \Lambda_3, Z \rangle$
LaTeX-представлення	`\langle C_{ZR}, C_A \rangle \mapsto C_{ZRI} = \langle M_{ZR}, \Sigma_{ZR}, \Lambda_{3}, Z \rangle`
Унікальні оператори, n_1	3
Унікальні операнди, n_2	7
Загальна кількість операторів, N_1	6
Загальна кількість операндів, N_2	7
Об’єм V	43.185
Складність D	1.5
Зусилля E	64.778
Оператори, частоти	Equal×1, MapsTo×1, AngleBracket×4
Операнди, частоти	C_{ZR}×1, C_A×1, C_{ZRI}×1, M_{ZR}×1, σ_{ZR}×1, λ_3 ×1, Z×1

Таблиця 5.6 – Результати аналізу виразу конкретизованої структури

Параметр	Значення
Вираз	$C_{\mathrm{ZRIC}} = \langle M_{\mathrm{ZR}}, \Sigma_{\mathrm{ZR}}, \Lambda_5, Z \rangle$
LaTeX-представлення	`C_{ZRIC} = \langle M_{ZR}, \Sigma_{ZR}, \Lambda_{5}, Z \rangle`
Унікальні оператори, n_1	2
Унікальні операнди, n_2	5
Загальна кількість операторів, N_1	3
Загальна кількість операндів, N_2	5
Об’єм V	22.459
Складність D	1
Зусилля E	22.459
Оператори, частоти	Equal×1, AngleBracket×2
Операнди, частоти	C_{ZRIC}×1, M_{ZR}×1, σ_{ZR}×1, λ_5×1, Z×1

Таблиця 5.7 – Результати аналізу виразу правила s1 (початкове)

Параметр	Значення
Вираз	$s_1 = \langle \sigma \rightarrow \alpha \cdot \beta \cdot \alpha \rangle$
LaTeX-представлення	`s_{1} = \langle \sigma \rightarrow \alpha \cdot \beta \cdot \alpha \rangle`
Унікальні оператори, n_1	4
Унікальні операнди, n_2	4
Загальна кількість операторів, N_1	6

Продовження таблиці 5.7

Параметр	Значення
Загальна кількість операндів, N_2	5
Об’єм V	33
Складність D	2.5
Зусилля E	82.5
Оператори, частоти	Multiply×2, Equal×1, To×1, AngleBracket×2
Операнди, частоти	s_1×1, σ×1, α×2, β×1

Таблиця 5.8 – Результати аналізу виразу правила s2 (розширення дільниці)

Параметр	Значення
Вираз	$s_2 = \langle \alpha \to \alpha \times (\beta \cdot \alpha) \rangle$
LaTeX-представлення	`s_{2} = \langle \alpha \rightarrow \alpha \times (\beta \cdot \alpha) \rangle`
Унікальні оператори, n_1	5
Унікальні операнди, n_2	3
Загальна кількість операторів, N_1	8
Загальна кількість операндів, N_2	5
Об’єм V	39
Складність D	4.167
Зусилля E	162.5
Оператори, частоти	Multiply×2, Equal×1, To×1, Parenthesis×2, AngleBracket×2

Продовження таблиці 5.8

Параметр	Значення
Операнди, частоти	s_2×1, α×3, β×1

Таблиця 5.9 – Результати аналізу виразу правила s3 (розгалуження)

Параметр	Значення
Вираз	$s_3 = \langle \alpha \to (\beta \cdot \alpha) \times (\beta \cdot \alpha) \rangle$
LaTeX-представлення	`s_{3} = \langle \alpha \rightarrow (\beta \cdot \alpha) \times (\beta \cdot \alpha) \rangle`
Унікальні оператори, n_1	5
Унікальні операнди, n_2	3
Загальна кількість операторів, N_1	11
Загальна кількість операндів, N_2	6
Об’єм V	51
Складність D	5
Зусилля E	255
Оператори, частоти	Multiply×3, Equal×1, To×1, Parenthesis×4, AngleBracket×2
Операнди, частоти	s_3×1, α×3, β×2

Таблиця 5.10 – Результати аналізу виразу правила s4 (підстанція)

Параметр	Значення
Вираз	$s_4 = \langle \alpha \to \delta \times \gamma \rangle$

Продовження таблиці 5.10

Параметр	Значення
LaTeX-представлення	`s_{4} = \langle \alpha \rightarrow \delta \times \gamma \rangle`
Унікальні оператори, n_1	5
Унікальні операнди, n_2	3
Загальна кількість операторів, N_1	6
Загальна кількість операндів, N_2	3
Об’єм V	27
Складність D	2.5
Зусилля E	67.5
Оператори, частоти	Multiply×1, Equal×1, To×1, AngleBracket×2, delta×1
Операнди, частоти	s_4×1, α×1, γ×1

Таблиця 5.11 – Результати аналізу виразу правила s5 (накопичувач)

Параметр	Значення
Вираз	$s_5 = \langle \delta \rightarrow e \cdot x \rangle$
LaTeX-представлення	`s_{5} = \langle \delta \rightarrow e \cdot x \rangle`
Унікальні оператори, n_1	5
Унікальні операнди, n_2	3
Загальна кількість операторів, N_1	6
Загальна кількість операндів, N_2	3

Продовження таблиці 5.11

Параметр	Значення
Об’єм V	27
Складність D	2.5
Зусилля E	67.5
Оператори, частоти	Multiply×1, Equal×1, To×1, AngleBracket×2, delta×1
Операнди, частоти	s_5×1, e×1, x×1

Таблиця 5.12 – Результати аналізу виразу правила s6 (без накопичувача)

Параметр	Значення
Вираз	$s_6 = \langle \delta \rightarrow \varepsilon \rangle$
LaTeX-представлення	`s_{6} = \langle \delta \rightarrow \varepsilon \rangle`
Унікальні оператори, n_1	4
Унікальні операнди, n_2	2
Загальна кількість операторів, N_1	5
Загальна кількість операндів, N_2	2
Об’єм V	18.095
Складність D	2
Зусилля E	36.189
Оператори, частоти	Equal×1, To×1, AngleBracket×2, delta×1
Операнди, частоти	s_6×1, ε×1

Таблиця 5.13 – Результати аналізу виразу правила s7 (живлення)

Параметр	Значення
Вираз	$s_7 = \langle \gamma \to a \cdot b \cdot (\theta : \lambda) \rangle$
LaTeX-представлення	`s_{7} = \langle \gamma \rightarrow a \cdot b \cdot (\theta : \lambda) \rangle`
Унікальні оператори, n_1	6
Унікальні операнди, n_2	6
Загальна кількість операторів, N_1	9
Загальна кількість операндів, N_2	6
Об’єм V	53.774
Складність D	3
Зусилля E	161.323
Оператори, частоти	Multiply×2, Divide×1, Equal×1, To×1, Parenthesis×2, AngleBracket×2
Операнди, частоти	s_7×1, γ×1, a×1, b×1, θ×1, λ×1

Таблиця 5.14 – Результати аналізу виразу правила s8 (живлення альт.)

Параметр	Значення
Вираз	$s_8 = \langle \gamma \to (\theta : \lambda) \cdot a \cdot b \rangle$
LaTeX-представлення	`s_{8} = \langle \gamma \rightarrow (\theta : \lambda) \cdot a \cdot b \rangle`
Унікальні оператори, n_1	6
Унікальні операнди, n_2	6

Продовження таблиці 5.14

Параметр	Значення
Загальна кількість операторів, N_1	9
Загальна кількість операндів, N_2	6
Об’єм V	53.774
Складність D	3
Зусилля E	161.323
Оператори, частоти	Multiply×2, Divide×1, Equal×1, To×1, Parenthesis×2, AngleBracket×2
Операнди, частоти	s_8×1, γ×1, θ×1, λ×1, a×1, b×1

Таблиця 5.15 – Результати аналізу виразу правила s9 (випрямляч)

Параметр	Значення
Вираз	$s_9 = \langle \theta \rightarrow c \rangle$
LaTeX-представлення	`s_{9} = \langle \theta \rightarrow c \rangle`
Унікальні оператори, n_1	3
Унікальні операнди, n_2	3
Загальна кількість операторів, N_1	4
Загальна кількість операндів, N_2	3
Об’єм V	18.095
Складність D	1.500
Зусилля E	27.142
Оператори, частоти	Equal×1, To×1, AngleBracket×2

Продовження таблиці 5.15

Параметр	Значення
Операнди, частоти	s_9×1, θ×1, c×1

Таблиця 5.16 – Результати аналізу виразу правила s10 (випрямляч рег.)

Параметр	Значення
Вираз	$s_{10} = \langle \theta \rightarrow c \cdot h \rangle$
LaTeX-представлення	`s_{10} = \langle \theta \rightarrow c \cdot h \rangle`
Унікальні оператори, n_1	4
Унікальні операнди, n_2	4
Загальна кількість операторів, N_1	5
Загальна кількість операндів, N_2	4
Об’єм V	27
Складність D	2
Зусилля E	54
Оператори, частоти	Multiply×1, Equal×1, To×1, AngleBracket×2
Операнди, частоти	s_{10}×1, θ×1, c×1, h×1

Таблиця 5.17 – Результати аналізу виразу правила s11 (інвертор)

Параметр	Значення
Вираз	$s_{11} = \langle \lambda \rightarrow d \cdot x \rangle$
LaTeX-представлення	`s_{11} = \langle \lambda \rightarrow d \cdot x \rangle`

Продовження таблиці 5.17

Параметр	Значення
Унікальні оператори, n_1	4
Унікальні операнди, n_2	4
Загальна кількість операторів, N_1	5
Загальна кількість операндів, N_2	4
Об’єм V	27
Складність D	2
Зусилля E	54
Оператори, частоти	Multiply×1, Equal×1, To×1, AngleBracket×2
Операнди, частоти	s_{11}×1, λ×1, d×1, x×1

Таблиця 5.18 – Результати аналізу виразу правила s12 (без інвертора)

Параметр	Значення
Вираз	$s_{12} = \langle \lambda \rightarrow \varepsilon \rangle$
LaTeX-представлення	`s_{12} = \langle \lambda \rightarrow \varepsilon \rangle`
Унікальні оператори, n_1	3
Унікальні операнди, n_2	3
Загальна кількість операторів, N_1	4
Загальна кількість операндів, N_2	3
Об’єм V	18.095
Складність D	1.5

Продовження таблиці 5.18

Параметр	Значення
Зусилля E	27.142
Оператори, частоти	Equal×1, To×1, AngleBracket×2
Операнди, частоти	s_{12}×1, λ×1, ε×1

Таблиця 5.19 – Результати аналізу виразу правила s13 (багатопутність)

Параметр	Значення
Вираз	$s_{13}\langle \beta \to \beta : \beta \rangle$
LaTeX-представлення	`s_{13} = \langle \beta \rightarrow \beta : \beta \rangle`
Унікальні оператори, n_1	4
Унікальні операнди, n_2	2
Загальна кількість операторів, N_1	5
Загальна кількість операндів, N_2	4
Об’єм V	23.265
Складність D	4
Зусилля E	93.059
Оператори, частоти	Divide×1, Equal×1, To×1, AngleBracket×2
Операнди, частоти	s_{13}×1, β×3

Таблиця 5.20 – Результати аналізу виразу правила s14 (перемички)

Параметр	Значення
Вираз	$s_{14} = \langle \beta \to \beta \cdot \beta \rangle$

Продовження таблиці 5.20

Параметр	Значення
LaTeX-представлення	`s_{14} = \langle \beta \rightarrow \beta \cdot \beta \rangle`
Унікальні оператори, n_1	4
Унікальні операнди, n_2	2
Загальна кількість операторів, N_1	5
Загальна кількість операндів, N_2	4
Об’єм V	23.265
Складність D	4
Зусилля E	93.059
Оператори, частоти	Multiply×1, Equal×1, To×1, AngleBracket×2
Операнди, частоти	s_{14}×1, β×3

Таблиця 5.21 – Результати аналізу виразу правила s15 (мережа)

Параметр	Значення
Вираз	$s_{15} = \langle \beta \rightarrow f \rangle$
LaTeX-представлення	`s_{15} = \langle \beta \rightarrow f \rangle`
Унікальні оператори, n_1	3
Унікальні операнди, n_2	3
Загальна кількість операторів, N_1	4
Загальна кількість операндів, N_2	3

Продовження таблиці 5.21

Параметр	Значення
Об’єм V	18.095
Складність D	1.5
Зусилля E	27.142
Оператори, частоти	Equal×1, To×1, AngleBracket×2
Операнди, частоти	s_{15}×1, β×1, f×1

Таблиця 5.22 – Результати аналізу виразу правила s16 (мережа + поїзд)

Параметр	Значення
Вираз	$s_{16} = \langle \beta \rightarrow t \cdot f \rangle$
LaTeX-представлення	`s_{16} = \langle \beta \rightarrow t \cdot f \rangle`
Унікальні оператори, n_1	4
Унікальні операнди, n_2	4
Загальна кількість операторів, N_1	5
Загальна кількість операндів, N_2	4
Об’єм V	27
Складність D	2
Зусилля E	54
Оператори, частоти	Multiply×1, Equal×1, To×1, AngleBracket×2
Операнди, частоти	s_{16}×1, β×1, t×1, f×1

Таблиця 5.23 – Результати аналізу виразу l1

Параметр	Значення
Вираз	$l_1 = \alpha \cdot \beta \cdot \alpha$
LaTeX-представлення	`l_{1} = \alpha \cdot \beta \cdot \alpha`
Унікальні оператори, n_1	2
Унікальні операнди, n_2	3
Загальна кількість операторів, N_1	3
Загальна кількість операндів, N_2	4
Об’єм V	16.253
Складність D	1.333
Зусилля E	21.671
Оператори, частоти	Multiply×2, Equal×1
Операнди, частоти	l_1×1, α×2, β×1

Таблиця 5.24 – Результати аналізу виразу l2

Параметр	Значення
Вираз	$l_2 = \alpha \cdot \beta \cdot (\alpha \times (\beta \cdot \alpha))$
LaTeX-представлення	`l_{2} = \alpha \cdot \beta \cdot (\alpha \times (\beta \cdot \alpha))`
Унікальні оператори, n_1	3
Унікальні операнди, n_2	3
Загальна кількість операторів, N_1	9
Загальна кількість операндів, N_2	6

Продовження таблиці 5.24

Параметр	Значення
Об’єм V	38.774
Складність D	3
Зусилля E	116.323
Оператори, частоти	Multiply×4, Equal×1, Parenthesis×4
Операнди, частоти	l_2×1, α×3, β×2

Таблиця 5.25 – Результати аналізу виразу l3 (розгалуження)

Параметр	Значення
Вираз	$l_3 = \alpha \cdot \beta \cdot (\alpha \times (\beta \cdot (\beta \cdot \alpha) \times (\beta \cdot \alpha)))$
LaTeX-представлення	`l_{3} = \alpha \cdot \beta \cdot (\alpha \times (\beta \cdot (\beta \cdot \alpha) \times (\beta \cdot \alpha)))`
Унікальні оператори, n_1	3
Унікальні операнди, n_2	6
Загальна кількість операторів, N_1	16
Загальна кількість операндів, N_2	9
Об’єм V	64.624
Складність D	4.5
Зусилля E	290.808
Оператори, частоти	Multiply×7, Equal×1, Parenthesis×8
Операнди, частоти	l_3×1, α×4, β×4

Таблиця 5.26 – Результати аналізу виразу l5 (з накопичувачем)

Параметр	Значення
Вираз	$l_5 = ((e \cdot x) \times \gamma) \cdot \beta \cdot (\alpha \times (\beta \cdot (\beta \cdot \alpha) \times (\beta \cdot \alpha)))$
LaTeX-представлення	`l_{5} = ((e \cdot x) \times \gamma) \cdot \beta \cdot (\alpha \times (\beta \cdot (\beta \cdot \alpha) \times (\beta \cdot \alpha)))`
Унікальні оператори, n_1	3
Унікальні операнди, n_2	6
Загальна кількість операторів, N_1	22
Загальна кількість операндів, N_2	11
Об’єм V	104.608
Складність D	2.75
Зусилля E	287.671
Оператори, частоти	Multiply×9, Equal×1, Parenthesis×12
Операнди, частоти	l_5×1, e×1, x×1, γ×1, β×4, α×3

Таблиця 5.27 – Результати аналізу виразу l8 (з інвертором)

Параметр	Значення
Вираз	$l_8 = ((e \cdot x) \times (a \cdot b) \cdot ((c \cdot h) : (d \cdot x))) \cdot \beta \cdot (\alpha \times (\beta \cdot (\beta \cdot \alpha) \times (\beta \cdot \alpha)))$
LaTeX-представлення	`l_{8} = ((e \cdot x) \times (a \cdot b) \cdot ((c \cdot h) : (d \cdot x))) \cdot \beta \cdot (\alpha \times`

Продовження таблиці 5.27

Параметр	Значення
	`(\beta \cdot (\beta \cdot \alpha) \times (\beta \cdot \alpha)))`
Унікальні оператори, n_1	4
Унікальні операнди, n_2	10
Загальна кількість операторів, N_1	35
Загальна кількість операндів, N_2	16
Об’єм V	194.175
Складність D	3.2
Зусилля E	621.36
Оператори, частоти	Multiply×13, Divide×1, Equal×1, Parenthesis×20
Операнди, частоти	l_8×1, e×1, x×2, a×1, b×1, c×1, h×1, d×1, β ×4, α×3

Таблиця 5.28 – Результати аналізу виразу l30 (повна схема)

Параметр	Значення
Вираз	$l_{30} = ((e \cdot x) \times (a \cdot b) \cdot ((c \cdot h) : (d \cdot x))) \cdot (f : (f \cdot t \cdot f)) \cdot ((a \cdot b \cdot c) \times (f : f) \cdot (t \cdot f \cdot (c : (d \cdot x) \cdot b \cdot a)) \times (f \cdot c \cdot b \cdot a)))$
LaTeX-представлення	`l_{30} = ((e \cdot x) \times (a \cdot b) \cdot ((c \cdot h) : (d \cdot x))) \cdot (f : (f \cdot t \cdot f)) \cdot ((a \cdot b \cdot c) \times (f : f) \cdot (t \cdot f \cdot (c : (d \cdot`

Продовження таблиці 5.28

Параметр	Значення
	`x) \cdot b \cdot a)) \times (f \cdot c \cdot b \cdot a)))`
Унікальні оператори, n_1	4
Унікальні операнди, n_2	10
Загальна кількість операторів, N_1	59
Загальна кількість операндів, N_2	29
Об’єм V	335.047
Складність D	5.8
Зусилля E	1943.274
Оператори, частоти	Multiply×23, Divide×4, Equal×1, Parenthesis×31
Операнди, частоти	l_{30}×1, e×1, x×3, a×4, b×4, c×4, h×1, d×2, f×7, t×2

5.7 Узагальнення та інтерпретація результатів

Отримані значення метрик дозволяють розглядати вибірку як сукупність формул різного рівня деталізації: від коротких означень і продукційних правил до розгорнутих виразів, що описують повні схеми. Для зручності інтерпретації результати доцільно розглядати через три “зрізи”:

1. Об’єм V. Розподіл на рис. 5.1 показує, що основна частина формул має невеликі значення V. Це характерно для означень, відображень і продукцій, де запис короткий, а повторюваність елементів невисока. Пікові значення формуються формулами, у яких велика довжина запису N поєднується з повтором однотипних операцій (передусім множення та групування дужками). Саме цей ефект найбільш виражений у формулі `l30`, де кількість операторів зростає за рахунок великої кількості дужок і множень.

2. Складність D. Графік на рис. 5.2 демонструє, що D у цій вибірці не зростає пропорційно V. Це очікувано, оскільки D залежить не тільки від довжини запису, а й від співвідношення операторів і операндів, а також від того, наскільки різноманітними є операнди (n_2). Підвищені значення D мають не лише найдовші формули, а й частина правил, у яких присутні операції над структурами (наприклад, об'єднання множин), а також перетворення з використанням кутових дужок і відображень.
3. Зусилля E. Розподіл на рис. 5.3 є найбільш контрастним. Це пояснюється визначенням E як добутку D та V: якщо формула одночасно довга і структурно складна, показник зусилля різко зростає. У результаті формується чітка трійка лідерів за складністю читання/аналізу: `l30`, `l8`, `l3`. Для практичного застосування це означає, що саме ці формули є першочерговими кандидатами для перевірки читабельності, уточнення позначень, а також для можливого розбиття на підвирази у методичних матеріалах.

5.8 Порівняння формул, що описують різні варіанти схеми

У вибірці присутні формули, які відображають різні рівні деталізації або різні варіанти побудови схеми.

- `l8 (З інвертором)` і `l30 (Повна схема)` мають однакові n_1 та n_2, але відрізняються загальною кількістю операторів і операндів (N_1, N_2). Це означає, що набір "типів" операцій і сутностей у цих формулах подібний, однак повна схема містить значно більше повторів, додаткових блоків і групувань. Саме повторюваність та деталізація дають найбільший внесок у зростання V і, як наслідок, E.
- Для правил типу `s1`–`s12` характерна близька структура: кутові дужки, стрілка перетворення та одна додаткова операція (здебільшого об'єднання). У таких формулах метрики часто згруповані в вузький діапазон, що підтверджує їх функцію "структурних" продукцій. Вони добре підходять для швидкого порівняння варіантів структури КПС: навіть невеликі зміни в записі (додавання множин, параметрів, атрибутів) відображаються у D та E.
- Короткі означення (`f1`, `d1`, `m1`) демонструють мінімальні або близькі до мінімальних значення метрик. Це відповідає їх призначенню: вони вводять позначення та типи відображень, не перевантажуючи формульний опис операціями.

5.9 Вплив формату джерела та обмеження експерименту

У межах цього експерименту вибірка сформована з документа `.docx`. Для такого джерела ключовим фактором є перетворення математичного подання у LaTeX-форму, придатну для подальшого аналізу. Практичний досвід показує, що перетворення зазвичай коректне, однак можливі артефакти (зайві символи, некоректні командні послідовності, відмінності у групуванні), які ускладнюють подальший розбір.

Валідація та нормалізація зменшують вплив таких випадків, проте повністю усунути їх неможливо без ручної перевірки окремих формул. Саме тому в дослідженні використовується підхід “контрольованої обробки”: проблемні фрагменти не приховуються, а позначаються як такі, що потребують уваги.

Обмеження експерименту:

- результати отримані на конкретній вибірці формул з одного джерельного документа; інші документи можуть мати інший стиль запису та іншу частку нестандартних конструкцій;
- метрики Холстеда відображають структурно-інформаційну складність запису, але не описують “фізичну” складність об’єкта моделювання; складний об’єкт може бути описаний короткою формулою, і навпаки;
- інтерпретація V, D, E є порівняльною: найбільш корисний результат – ранжування та виявлення “пікових” формул, а не абсолютне значення метрики як самодостатній показник.

5.10 Можливі напрями розвитку підходу

Під час аналізу результатів виявлено декілька напрямів, які можуть підвищити інформативність порівняння:

- розширення правил класифікації операторів для окремих математичних конструкцій КПМ (наприклад, окремий облік операторів композиції, кванторів або специфічних відношень, якщо вони регулярно зустрічаються у матеріалах);
- введення додаткових узагальнених показників на рівні документа: частка формул з високими значеннями E, “щільність” складних формул, а також розбиття за типами (означення/правила/виведення);
- порівняння результатів для одних і тих самих формул у джерелах `.tex` та `.docx`, щоб кількісно оцінити вплив перетворення на стабільність метрик.

5.11 Висновки до розділу 5

Експериментальне дослідження виконано на вибірці формул, що описують елементи КПС/КПМ зон рекуперації постійного струму, із розрахунком метрик Холстеда для кожної формули.

Розподіли V, D, E показали, що основна частина формул має помірні значення метрик, тоді як декілька формул формують виражені максимуми, що відповідає найбільш розгорнутим описам схеми.

Показник V найбільше реагує на довжину запису та повторюваність операцій, тоді як D відображає структурну насиченість і співвідношення операторів/операндів.

Найбільш "важкими" за інтегральною оцінкою E виявилися формули `l30`, `l8` та `l3`, що узгоджується з їх змістом: вони містять значну кількість групувань, множень і вкладених фрагментів.

Правила продукцій типу `s1`–`s12` формують компактний клас записів зі схожою структурою та близькими метриками, що зручно для порівняння варіантів опису структури.

Застосування `.docx` як джерела підтвердило важливість контролю коректності перетворення математичних об'єктів у LaTeX-подання; можливі артефакти не зупиняють аналіз, але мають бути явно позначені.

Отримані результати підтверджують придатність метрик Холстеда як кількісного інструмента для порівняння формалізованих описів КПМ/КПС та виявлення формул, що потребують підвищеної уваги під час аналізу й супроводу.

ЗАГАЛЬНІ ВИСНОВКИ

У кваліфікаційній роботі виконано дослідження складності конструктивно-продукційних моделей зон рекуперації постійного струму та показано практичну доцільність кількісного оцінювання формалізованих описів у вигляді формул. У межах аналізу предметної області уточнено, що складність формульного запису впливає на читабельність, перевірку та супровід моделей, а отже потребує відтворюваних критеріїв оцінювання, які дозволяють порівнювати різні варіанти побудови КПМ/КПС.

Теоретичну основу роботи становлять метрики Холстеда. У роботі наведено їх визначення та інтерпретацію для аналізу математичних формул, а також сформульовано правила класифікації елементів формалізованого запису на оператори й операнди. Це дало можливість перейти від суто описового порівняння формул до обчислюваних показників n_1, n_2, N_1, N_2 та похідних метрик V, D, E, які використовуються для ранжування і виявлення фрагментів підвищеної складності.

Практичним результатом роботи стала розробка методики обробки формул з джерел `.tex` і `.docx`, що включає вилучення математичних фрагментів, перевірку коректності та нормалізацію запису, семантичний розбір у деревоподібне подання та подальше обчислення метрик. На основі цієї методики спроєктовано й реалізовано настільний програмний засіб на базі Electron з графічним інтерфейсом користувача. Застосунок підтримує імпорт документів, редагування формул, запуск аналізу, візуалізацію результатів, сортування/фільтрацію та збереження робочих сесій у форматі `.fmx`, що робить процес дослідження повторюваним і зручним для практичного використання.

Окрему увагу приділено інженерним рішенням, які забезпечують стабільність і придатність застосунку до роботи з реальними документами. Обчислення виконуються без блокування інтерфейсу, а взаємодія між частинами застосунку реалізована через механізми IPC із врахуванням вимог безпеки. Для контролю коректності реалізації застосовувалося тестування: модульні та інтеграційні перевірки додавалися під час розробки й використовувалися для фіксації критичних прикладів вилучення, валідації LaTeX та обчислення метрик, що дозволило зменшити кількість регресій при розширенні функціональності.

Експериментальне дослідження на підготовленій вибірці формул підтвердило працездатність запропонованого підходу й продемонструвало, що метрики V, D, E чутливо реагують на збільшення довжини запису, кількості операцій та рівня вкладеності. Побудовані розподіли показали, що основна частина формул має помірні значення метрик, тоді як окремі розгорнуті вирази формують виражені максимуми та закономірно виділяються як найскладніші. Це створює підґрунтя для практичного застосування результатів: кількісні оцінки дозволяють обґрунтовано порівнювати варіанти формалізованих описів моделей і виявляти фрагменти, які потребують першочергової перевірки або уточнення.

Разом з тим зафіксовано обмеження підходу, зокрема залежність від якості математичного подання та особливостей перетворення формул з `.docx`. У подальшій роботі доцільним є розширення правил класифікації операторів та операндів для специфічних конструкцій КПМ, запровадження додаткових узагальнених показників на рівні документа та виконання порівняльного аналізу стабільності оцінок для джерел різних форматів на більш широкій вибірці.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Khodaparastan, M., Mohamed, A. A., Brandauer, W. Recuperation of Regenerative Braking Energy in Electric Rail Transit Systems [Електронний ресурс] / M. Khodaparastan, A. A. Mohamed, W. Brandauer // IEEE Transactions on Intelligent Transportation Systems – 2019. – Т. 20. – № 8. – С. 2831–2847. – DOI: 10.1109/TITS.2018.2886809.
2. Шинкаренко, В. І., Іванов, О. П., Гармаш, О. І. Конструктивно-продукційне моделювання структури системи електропостачання тяги постійного струму [Електронний ресурс] / В. І. Шинкаренко, О. П. Іванов, О. І. Гармаш // Системні технології – 2024. – № 6(155). – С. 145–158. – Режим доступу: https://journals.nmetau.edu.ua/index.php/st/article/view/1922. – Конструктивно-продукційне моделювання структури системи електропостачання тяги постійного струму. – DOI: 10.34185/1562-9945-6-155-2024-14.
3. Атрибутивне насичення конструктивно-продукційної моделі ділянки системи електропостачання тяги постійного струму [Електронний ресурс] // Системні технології – 2025. – № 3(158). – С. 96–109. – Режим доступу: https://journals.nmetau.edu.ua/index.php/st/article/view/2012. – Атрибутивне насичення конструктивно-продукційної моделі ділянки системи електропостачання тяги постійного струму. – DOI: 10.34185/1562-9945-3-158-2025-10.
4. Конструктивно-продукційне моделювання розподілу енергії рекуперації на основі нечіткої логіки [Електронний ресурс] // Системні технології – 2025. – № 4(159). – С. 200–211. – Режим доступу: https://journals.nmetau.edu.ua/index.php/st/article/view/2054. – Конструктивно-продукційне моделювання розподілу енергії рекуперації на основі нечіткої логіки. – DOI: 10.34185/1562-9945-4-159-2025-20.
5. Halstead, M. H. Elements of Software Science [Текст] / M. H. Halstead. – New York : Elsevier, 1977. – 142 с.
6. Fitzsimmons, A., Love, T. A Review and Evaluation of Software Science [Електронний ресурс] / A. Fitzsimmons, T. Love // ACM Computing Surveys – 1978. – Т. 10. – № 1. – С. 3–18. – DOI: 10.1145/356715.356717.

7. Shepperd, M., Ince, M. An Evaluation of Software Product Metrics [Електронний ресурс] / M. Shepperd, M. Ince // Information & Software Technology – 1988. – Т. 30. – № 3. – С. 177–188. – DOI: 10.1016/0950-5849(88)90064-X.
8. MacFarlane, J. Pandoc User's Guide [Електронний ресурс] – Режим доступу: https://pandoc.org/MANUAL.html. – Pandoc User's Guide. – Дата звернення: 15.11.2025.
9. cortex-js/compute-engine [Електронний ресурс] – Режим доступу: https://github.com/cortex-js/compute-engine. – cortex-js/compute-engine. – Дата звернення: 12.11.2025.
10. Chowdhury, S., Holmes, R., Zaidman, A. Revisiting the Debate: Are Code Metrics Useful for Measuring Maintenance Effort? [Електронний ресурс] / S. Chowdhury, R. Holmes, A. Zaidman // Empirical Software Engineering Journal (EMSE) – 2022. – № 27(6):1-31. – Режим доступу: https://www.cs.ubc.ca/~rtholmes/papers/emse_2022_chowdhury_preprint.pdf. – Revisiting the Debate: Are Code Metrics Useful for Measuring Maintenance Effort?.
11. On the accuracy of code complexity metrics: A neuroscience-based guideline for improvement [Електронний ресурс] // Frontiers in Neuroscience – 2023. – Режим доступу: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2022.1065366/full. – On the accuracy of code complexity metrics: A neuroscience-based guideline for improvement. – DOI: 10.3389/fnins.2022.1065366.
12. Evaluating the effectiveness of decomposed Halstead Metrics in software fault prediction [Електронний ресурс] // PeerJ Computer Science – 2023. – Режим доступу: https://pmc.ncbi.nlm.nih.gov/articles/PMC10703020/. – Evaluating the effectiveness of decomposed Halstead Metrics in software fault prediction.
13. [MS-OI29500]: oMath and Math Arguments [Електронний ресурс] – 2024. – Режим доступу: https://learn.microsoft.com/en-us/openspecs/office_standards/ms-oi29500/dbc60737-c3cb-43b1-8fe8-cd591fa4472d. – [MS-OI29500]: oMath and Math Arguments. – Дата звернення: 11.12.2025.
14. OOXML Format Family - ISO/IEC 29500 and ECMA 376 [Електронний ресурс] – 2024. – Режим доступу: https://www.loc.gov/preservation/digital/formats/fdd/

fdd000395.shtml. – OOXML Format Family - ISO/IEC 29500 and ECMA 376. – Дата звернення: 28.11.2025.

15. MathJax Documentation [Електронний ресурс] – Режим доступу: https://docs.mathjax.org/. – MathJax Documentation. – Дата звернення: 09.12.2025.
16. KaTeX: Supported Functions [Електронний ресурс] – Режим доступу: https://katex.org/docs/supported.html. – KaTeX: Supported Functions. – Дата звернення: 12.12.2025.
17. tamuratak. latex-utensils [Електронний ресурс] – Режим доступу: https://github.com/tamuratak/latex-utensils. – latex-utensils. – Дата звернення: 20.11.2025.
18. Electron: Process Model [Електронний ресурс] – Режим доступу: https://electronjs.org/docs/latest/tutorial/process-model. – Electron: Process Model. – Дата звернення: 09.01.2026.
19. The TypeScript Handbook [Електронний ресурс] – Режим доступу: https://www.typescriptlang.org/docs/handbook/intro.html. – The TypeScript Handbook. – Дата звернення: 19.11.2025.
20. Svelte Docs: state [Електронний ресурс] – Режим доступу: https://svelte.dev/docs/svelte/%24state. – Svelte Docs: state. – Дата звернення: 16.11.2025.
21. Electron: Inter-Process Communication [Електронний ресурс] – Режим доступу: https://electronjs.org/docs/latest/tutorial/ipc. – Electron: Inter-Process Communication. – Дата звернення: 15.11.2025.
22. Electron: Security [Електронний ресурс] – Режим доступу: https://electronjs.org/docs/latest/tutorial/security. – Electron: Security. – Дата звернення: 15.11.2025.
23. Electron: Context Isolation [Електронний ресурс] – Режим доступу: https://electronjs.org/docs/latest/tutorial/context-isolation. – Electron: Context Isolation. – Дата звернення: 15.11.2025.
24. Electron API: contextBridge [Електронний ресурс] – Режим доступу: https://electronjs.org/docs/latest/api/context-bridge. – Electron API: contextBridge. – Дата звернення: 15.11.2025.

25. Electron: Using Preload Scripts [Електронний ресурс] – Режим доступу: https://electronjs.org/docs/latest/tutorial/tutorial-preload. – Electron: Using Preload Scripts. – Дата звернення: 15.11.2025.
26. Node.js Documentation: worker_threads [Електронний ресурс] – Режим доступу: https://nodejs.org/api/worker_threads.html. – Node.js Documentation: worker_threads. – Дата звернення: 26.11.2025.
27. Chart.js Documentation: Legend Configuration [Електронний ресурс] – Режим доступу: https://www.chartjs.org/docs/latest/configuration/legend.html. – Chart.js Documentation: Legend Configuration. – Дата звернення: 26.12.2025.
28. Chart.js Documentation: API [Електронний ресурс] – Режим доступу: https://www.chartjs.org/docs/latest/developers/api.html. – Chart.js Documentation: API. – Дата звернення: 26.12.2025.
29. Jest Documentation [Електронний ресурс] – Режим доступу: https://jestjs.io/docs/getting-started. – Jest Documentation. – Дата звернення: 28.12.2025.
30. ts-jest Documentation [Електронний ресурс] – Режим доступу: https://kulshekhar.github.io/ts-jest/docs/. – ts-jest Documentation. – Дата звернення: 28.12.2025.

ДОДАТОК А

Технічне завдання

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного університету
науки і технологій
_____ /Анатолій РАДКЕВИЧ/
Дата _____

ПРОГРАМА ДЛЯ ДОСЛІДЖЕННЯ СКЛАДНОСТІ КОНСТРУКТИВНО-ПРОДУКЦІЙНИХ МОДЕЛЕЙ ЗОН РЕКУПЕРАЦІЇ ПОСТІЙНОГО СТРУМУ

Технічне завдання

ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.1535-01 ЛЗ

Завідувач кафедри КІТ
Вадим ГОРЯЧКІН

Керівник розробки
проф. Віктор ШИНКАРЕНКО

Виконавець
Євген ШАПОВАЛ

Нормоконтролер
доц. Світлана ВОЛКОВА

2025

ЗАТВЕРДЖЕНО
44165850.1535-01-ЛЗ

ПРОГРАМА ДЛЯ ДОСЛІДЖЕННЯ СКЛАДНОСТІ КОНСТРУКТИВНО-ПРОДУКЦІЙНИХ МОДЕЛЕЙ ЗОН РЕКУПЕРАЦІЇ ПОСТІЙНОГО СТРУМУ

Технічне завдання
44165850.1535-01

Листів 11

2025

ЗМІСТ

ВВЕДЕННЯ

«Програмний засіб для аналізу складності конструктивно-продукційних моделей зон рекуперації постійного струму» – це настільний застосунок, призначений для кількісного оцінювання складності формалізованих описів КПМ/КПС, поданих у вигляді математичних формул. Застосунок забезпечує імпорт документів, вилучення формул, перевірку коректності запису, обчислення метрик Холстеда та візуалізацію результатів для порівняння різних варіантів моделей.

Основні терміни: конструктивно-продукційна модель, конструктивно-продукційна структура, зона рекуперації, система тягового електропостачання, метрики Холстеда, оператор, операнд, LaTeX, MathJSON, графічний інтерфейс користувача.

Причина виникнення необхідності розробки ПЗ: у задачах проєктування та аналізу зон рекуперації важливо порівнювати альтернативні формалізовані описи моделей та виявляти найбільш складні фрагменти, які ускладнюють перевірку, супровід і подальшу модифікацію. Використання кількісних метрик складності дозволяє перейти від суб’єктивної оцінки до відтворюваних показників, що дає змогу обґрунтовувати вибір структурних рішень на етапі роботи з моделлю.

Область застосування: дослідження та проєктування систем тягового електропостачання з урахуванням рекуперативного гальмування, формалізоване моделювання зон рекуперації, порівняльний аналіз варіантів КПМ/КПС і підготовка матеріалів для подальших досліджень.

ПІДСТАВА ДЛЯ РОЗРОБКИ

Розробка програмного забезпечення для аналізу складності конструктивно-продукційних моделей зон рекуперації постійного струму ведеться на підставі наказу ректора Українського державного університету науки і технологій «Про призначення керівників та затвердження тем бакалаврських робіт» за спеціальністю 121 «Інженерія програмного забезпечення» факультету «Комп'ютерних технологій і систем» по кафедрі «Комп'ютерні інформаційні технології».

Тема дипломної роботи – «Дослідження складності конструктивно-продукційних моделей зон рекуперації постійного струму».

ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення: програмний засіб призначений для автоматизованого вилучення та аналізу математичних формул, що використовуються у формалізованих описах КПМ/КПС зон рекуперації постійного струму. Застосунок має забезпечувати обчислення метрик Холстеда та формування результатів у вигляді таблиць і графіків для подальшого порівняння моделей.

Експлуатаційне призначення: надання досліднику або інженеру-розробнику зручного інструмента для роботи з набором формул з документів `.tex` і `.docx`, виконання повторюваного аналізу складності та збереження результатів дослідження у файлі сесії. Засіб призначений для використання під час підготовки дослідницьких матеріалів, перевірки узгодженості формалізованого запису та вибору варіантів опису моделей.

ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

4.1. Вимоги до функціональних характеристик

Вимоги до функціональних характеристик наступні:

- можливість імпортувати документи з формулами у форматах `.tex` та `.docx`;
- вилучення математичних формул (inline і display), формування переліку формул для аналізу;
- перевірка коректності та нормалізація LaTeX-подання формули;
- редагування формули в інтерфейсі (виправлення запису, перейменування, повторний аналіз);
- обчислення метрик Холстеда для кожної формули (n_1, n_2, N_1, N_2, V, D, E);
- візуалізація результатів та порівняльний аналіз (графіки, сортування/фільтрація);

4.2. Вимоги до надійності

- коректна обробка помилкових або неповних формул без аварійного завершення роботи;
- контроль введення та валідація формул, попередження про проблемні фрагменти;
- архівна копія тексту програми.

4.3. Умови експлуатації

Програмний продукт повинен використовуватись у приміщеннях, які відповідають умовам роботи ЕОМ та мають наступні кліматичні умови:

- 21-25 С;
- відносна вологість повітря 40-60%.

Працювати з програмою може людина, що має навички роботи з персональним комп’ютером та ознайомлена з керівництвом користувача.

4.4. Вимоги до складу і параметрів технічних засобів

Вимоги до складу і параметрів технічних засобів наступні:

- процесор 2.8 ГГц;
- 4 ГБ оперативної пам’яті;
- 1 ГБ вільного місця на диску;

- монітор з роздільною здатністю не менше 1024x768;
- клавіатура та маніпулятор «миша».

4.5. Вимоги до інформаційної і програмної сумісності

Для настільного застосунку:

- операційна система Windows або Linux;
- наявність середовища виконання, необхідного для запуску Electron-застосунку (постачається разом із застосунком у складі збірки);
- доступ до файлової системи для відкриття `.tex`/`.docx` та збереження `.fmx`.

Програмне забезпечення повинно підтримувати роботу на сучасних настільних платформах та забезпечувати коректне відображення формул і графіків у графічному інтерфейсі.

4.6. Вимоги до маркування та упаковки

Упаковка програмного продукту, включаючи документацію, повинна бути захищена від пошкоджень. На упаковці повинна бути вказана назва продукту, розробник, контакти, рік розробки.

На рис. 4.1 представлено приклад маркування

Програмний засіб для аналізу складності КПМ
зон рекуперації постійного струму
розробник: Шаповал Євген
УДУНТ.КІТ, кафедра КІТ
2025

Рисунок 4.1 – Приклад маркування

4.7. Вимоги до транспортування та зберігання

Транспортування повинне забезпечувати збереження програмного продукту, його цілісність і запобігання несанкціонованого доступу до нього. Програмний виріб міститься на оптичному носії даних типу CDR і повинно мати відповідну упаковку для захисту від механічних ушкоджень та атмосферного впливу (пластиковий футляр).

ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу програмної документації мають входити:

- специфікація;
- текст програми;
- опис програми;
- керівництво користувача.

Вся документація до програмного продукту повинна задовольняти вимоги до програмної документації [1].

СТАДІЇ І ЕТАПИ РОЗРОБКИ

В таблиці 6.1 приведені стадії та етапи розробки.

Таблиця 6.1 Стадії та етапи розробки

Стідія	Зміст робіт	Строки виконання
Технічне завдання	Постановка задачі аналізу складності КПМ/КПС, збір предметної інформації та визначення критеріїв оцінювання. Вибір метрик і підходів до обробки формул. Визначення вимог до технічних засобів та сумісності. Узгодження та затвердження технічного завдання.	04.10.2025 – 27.10.2025
Робочий проект	Реалізація настільного застосунку та інтеграція обчислень метрик	28.11.2025 – 12.12.2025
	Тестування застосунку та перевірка результатів на контрольних формулах	15.12.2025 – 15.01.2026
	Розробка, узгодження та затвердження програмної документації	25.12.2025 – 04.01.2026

ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Контроль за виконанням робіт з розробки програмного засобу для аналізу складності конструктивно-продукційних моделей зон рекуперації постійного струму здійснює керівник проекту.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.: Ю. М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. - 38 с.

ДОДАТОК Б

Текст програми

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного університету
науки і технологій
_____ /Анатолій РАДКЕВИЧ/
Дата _____

ПРОГРАМА ДЛЯ ДОСЛІДЖЕННЯ СКЛАДНОСТІ КОНСТРУКТИВНО-ПРОДУКЦІЙНИХ МОДЕЛЕЙ ЗОН РЕКУПЕРАЦІЇ ПОСТІЙНОГО СТРУМУ

Текст програми

ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.1535-12 ЛЗ

Завідувач кафедри КІТ
Вадим ГОРЯЧКІН

Керівник розробки
проф. Віктор ШИНКАРЕНКО

Виконавець
Євген ШАПОВАЛ

Нормоконтролер
доц. Світлана ВОЛКОВА

2026 рік

ЗАТВЕРДЖЕНО
44165850.1535-12

ПРОГРАМА ДЛЯ ДОСЛІДЖЕННЯ СКЛАДНОСТІ КОНСТРУКТИВНО-ПРОДУКЦІЙНИХ МОДЕЛЕЙ ЗОН РЕКУПЕРАЦІЇ ПОСТІЙНОГО СТРУМУ

Текст програми
44165850.1535-12

Листів 37

2026

ЗМІСТ

ОПИС СТРУКТУРИ ПРОЄКТУ

Проєкт `electron-svelte-latex` є настільним застосунком на базі Electron з інтерфейсом на Svelte 5. Репозиторій організований так, щоб розділити код головного процесу Electron, код інтерфейсу та допоміжні матеріали для збірки, пакування і тестування.

1.1. Основні каталоги

- `electron/` — частина Electron, яка працює в головному процесі. Тут зосереджені модулі, що мають доступ до файлової системи, створюють вікно застосунку, організовують IPC-взаємодію та запускають фонові задачі.
- `electron/resources/` — ресурси, які додаються до збірки під час пакування. У проєкті використовується підкаталог із Pandoc, який додається як `extraResource` для роботи з `.docx`.
- `src/` — рендерер, тобто інтерфейс застосунку:
  - `src/components/` — UI-компоненти (панелі, картки формул, елементи керування, діаграми, повідомлення).
  - `src/lib/` — допоміжні модулі для роботи застосунку (утиліти, робота зі станом, обробка даних, локалізація).
- `test/` — тести (модульні та інтеграційні), що запускаються через Jest.
- `scripts/` — службові сценарії для збірки, зокрема для підготовки зовнішніх інструментів перед пакуванням.

1.2. Конфігураційні файли

- `package.json` — метадані проєкту, залежності та сценарії запуску. Вказує тип модулів `module` та вхідний файл Electron, який формується після збірки.
- `forge.config.cjs` — конфігурація Electron Forge: режим пакування `asar`, makers для створення інсталяційних артефактів, підключення Vite-плагіна та включення ресурсів `electron/resources/pandoc` до збірки.
- `vite.config.ts` — конфігурація Vite для рендерера, включно з Svelte-плагіном і параметрами dev-сервера.
- `svelte.config.js` — базові налаштування Svelte, зокрема `vitePreprocess()` для коректної обробки Svelte-файлів.

1.3. Використані технології та бібліотеки

- Платформа застосунку: Electron, Node.js.
- Інтерфейс: Svelte 5, Vite, Vanilla CSS.
- Візуалізація: Chart.js для діаграм, KaTeX для відображення формул.
- Робота з формулами та поданнями: `@cortex-js/compute-engine`, `latex-utensils`, `mathml-to-latex`, `xslt`.
- Пакування: Electron Forge (Squirrel, ZIP та інші makers).
- Контроль якості: ESLint, Prettier, `svelte-check`, Husky та lint-staged.
- Тестування: Jest, JSDOM, Testing Library для Svelte, `ts-jest`.

РОЗШИРЕНИЙ ОПИС ТЕКСТУ ПРОГРАМИ

2.1. Опис файлу `components/App.svelte`

`App.svelte` є головним компонентом інтерфейсу застосунку. У ньому зосереджений стан програми: список імпортованих файлів і формул, поточний вибір, видимість окремих формул для порівняння, а також службові повідомлення про хід обробки та помилки. Компонент ініціює дії через `window.api` (відкриття файлів, читання та збереження `.fmx`, запуск вилучення і аналізу), підписується на події прогресу та команди меню. Відображення організоване як робоча область зі списком формул (з рендерингом KaTeX та можливістю редагування) і панеллю аналітики, де показуються узагальнені показники та діаграми порівняння.

2.2. Опис файлу `components/Chart.svelte`

`Chart.svelte` є тонкою обгорткою над Chart.js. Компонент створює екземпляр графіка в елементі `<canvas>`, приймає підготовлені дані ззовні та оновлює відображення при зміні вхідного набору. Передбачено коректну ініціалізацію навіть за відсутності даних та очищення графіка, коли дані скидаються.

2.3. Опис файлу `main.ts`

`main.ts` є точкою входу рендерера. Файл монтує `App.svelte` у DOM-елемент із ідентифікатором `app` та запускає інтерфейс без додаткової логіки.

2.4. Опис файлу `lib/halstead.ts`

`halstead.ts` реалізує обчислення метрик Холстеда для формульних виразів. Модуль приводить LaTeX до придатного для обробки вигляду, виконує семантичний розбір через Compute Engine і формує структури підрахунку операторів та операндів. На основі лічильників обчислюються похідні показники (обсяг, складність, зусилля) та формується результат, зручний для відображення в інтерфейсі, включно з частотними словниками та підсумками для набору формул.

2.5. Опис файлу `lib/extraction.ts`

`extraction.ts` відповідає за вилучення формул із LaTeX-тексту. Для цього використовується `latex-utensils`, який будує синтаксичне дерево документа; далі виконується обхід дерева й відбір вузлів, що відповідають математичним режимам і типовим математичним середовищам. На виході повертається перелік

знайдених формул та список помилок, якщо під час розбору виявлено проблемні ділянки.

2.6. Опис файлу `lib/translations.ts`

`translations.ts` містить словники локалізації інтерфейсу для мов `en` та `ua` і допоміжну функцію `t`, яка повертає рядок за ключем та підставляє параметри в шаблони повідомлень. Модуль використовується в компонентах інтерфейсу для підписів кнопок, заголовків, статусів обробки та повідомлень про помилки.

2.7. Опис файлу `lib/types.ts`

`types.ts` визначає спільні типи даних, які використовуються між інтерфейсом і логікою: структури прогресу, результати аналізу, метадані формули, формат файлу сесії `.fmx`. Окремо описано інтерфейс `ElectronAPI`, який відповідає набору методів, доступних у `window.api` через preload-шар, а також типи обробників подій меню та прогресу.

2.8. Вміст файлу components/App.svelte

```
<script lang="ts">
  import * as katex from 'katex';
  import 'katex/dist/katex.min.css';
  import { t, type Language } from '../lib/
translations';
  import Chart from './Chart.svelte';
  import { SvelteSet } from 'svelte/reactivity';
  import { writable } from 'svelte/store';
  import type {
    ProgressData,
    AnalysisProgressData,
    AnalysisRow,
    AnalysisResults,
    FormulaMetadata,
    FormulaFile,
  } from '../lib/types';

  let filePath = $state<string | null>(null);
  let formulas = $state<string[]>([]);
  let formulaMetadata =
$state<FormulaMetadata[]>([]);
  let fileMetadata =
$state<FormulaFile['metadata']>({
    created: new Date().toISOString(),
    modified: new Date().toISOString(),
  });
  let raw = $state('');
  let extractionErrors = $state<string[]>([]);
  let debugMode = $state(false);
  let currentLang = $state<Language>('ua');
  type SortBy = 'name' | 'volume' | 'difficulty'
| 'effort' | 'created';
  let sortBy = $state<SortBy>('created');
  let sortOrder = $state<'asc' |
'desc'>('desc');
  let comparisonFormulaIds = new
SvelteSet<string>();
  let hiddenFormulaIds = new
SvelteSet<string>();
  let isProcessing = $state(false);
  let processingProgress = $state<string>('');
  let progressValue = $state(0);
  let totalFiles = $state(0);
  let processedFiles = $state(0);
  let editingNameId = $state<string |
null>(null);
  let editingLatexId = $state<string |
null>(null);
  let editingText = $state<string>('');
  let deletingFormulaId = $state<string |
null>(null);
  let statusMessage = $state<string>('');
  let statusVisible = $state(false);
  let statusTimeout: number;

  // Prepare chart data for a specific metric
  function prepareMetricChartData(
    rows: AnalysisRow[],
    formulas: FormulaMetadata[],
    metric: 'Volume' | 'Difficulty' | 'Effort'
  ) {
    // Only include formulas that are
in comparisonFormulaIds, in the current sorted
order
    const comparisonList = sortedFormulas.
filter((f) => comparisonFormulaIds.has(f.id));
    if (comparisonList.length === 0) return
null;

    const datasets = comparisonList.
map((formula) => {
      const rowIndex = formulaMetadata.
indexOf(formula);
      const row = rows[rowIndex];
      const isSuccess = row?.success;
      // Get index from original formulaMetadata
for stable color assignment
      const originalIdx = formulaMetadata.
findIndex((f) => f.id === formula.id);

      return {
        label: formula.name,
        data: [isSuccess ? row.halstead[metric]
|| 0 : 0],
        backgroundColor: `hsl(${(originalIdx *
360) / formulaMetadata.length}, 70%, 50%)`,
        borderColor: `hsl(${(originalIdx *
360) / formulaMetadata.length}, 70%, 40%)`,
        borderWidth: 1,
        hidden: hiddenFormulaIds.has(formula.
id),
      };
    });

    return {
      labels: [t(currentLang, metric.
toLowerCase())],
      datasets,
    };
  }

  let analysisResults = writable(null as
AnalysisResults | null);

  let sortedFormulas = $derived.by(() => {
    const list = [...formulaMetadata];
    list.sort((a, b) => {
      let valA: any = a.name;
      let valB: any = b.name;

      if (sortBy === 'created') {
        valA = a.created;
```

```
        valB = b.created;
      } else if (sortBy === 'volume') {
        valA = a.analysis?.Volume ?? -1;
        valB = b.analysis?.Volume ?? -1;
      } else if (sortBy === 'difficulty') {
        valA = a.analysis?.Difficulty ?? -1;
        valB = b.analysis?.Difficulty ?? -1;
      } else if (sortBy === 'effort') {
        valA = a.analysis?.Effort ?? -1;
        valB = b.analysis?.Effort ?? -1;
      } else if (sortBy === 'name') {
        valA = a.name.toLowerCase();
        valB = b.name.toLowerCase();
      }

      if (valA < valB) return sortOrder ===
'asc' ? -1 : 1;
      if (valA > valB) return sortOrder ===
'asc' ? 1 : -1;
      return 0;
    });
    return list;
  });

  let volumeChartData = $derived(
    $analysisResults && comparisonFormulaIds.
size > 0
      ? prepareMetricChartData($analysisResults.
rows, formulaMetadata, 'Volume')
      : null
  );
  let difficultyChartData = $derived(
    $analysisResults && comparisonFormulaIds.
size > 0
      ? prepareMetricChartData($analysisResults.
rows, formulaMetadata, 'Difficulty')
      : null
  );
  let effortChartData = $derived(
    $analysisResults && comparisonFormulaIds.
size > 0
      ? prepareMetricChartData($analysisResults.
rows, formulaMetadata, 'Effort')
      : null
  );

  let effortRange = $derived.by(() => {
    if (!$analysisResults) return { min: 0, max:
0 };
    const efforts = $analysisResults.rows
      .filter((r) => r && r.success)
      .map((r) => r.halstead.Effort);
    if (efforts.length === 0) return { min: 0,
max: 0 };
    return {
      min: Math.min(...efforts),
      max: Math.max(...efforts),
    };
  });

  function getEffortColor(effort: number) {
    const { min, max } = effortRange;
    if (max === min) return 'rgba(39, 174, 96,
0.3)';

    // Normalize effort between 0 and 1 (0 is
best, 1 is worst)
    const ratio = (effort - min) / (max - min);

    // HSL: 120 (Green) to 0 (Red)
    const hue = 120 * (1 - ratio);
    return `hsla(${hue}, 70%, 50%, 0.3)`;
  }

  let selectedOverall = $derived.by(() => {
    if (!$analysisResults) return null;
    const selectedRows = formulaMetadata
      .filter((f) => comparisonFormulaIds.has(f.
id) && !hiddenFormulaIds.has(f.id))
      .map((f) => {
        const idx = formulaMetadata.indexOf(f);
        return $analysisResults.rows[idx];
      })
      .filter((r) => r && r.success);

    if (selectedRows.length === 0) return null;

    const allOps: Record<string, number> = {};
    const allOprnds: Record<string, number> =
{};

    for (const row of selectedRows) {
      if (!row.success) continue;
      for (const [op, count] of Object.
entries(row.halstead.operators)) {
        allOps[op] = (allOps[op] || 0) + count;
      }
      for (const [op, count] of Object.
entries(row.halstead.operands)) {
        allOprnds[op] = (allOprnds[op] || 0)
+ count;
      }
    }

    const n1 = Object.keys(allOps).length;
    const n2 = Object.keys(allOprnds).length;
    const N1 = Object.values(allOps).reduce((a,
b: number) => a + b, 0);
    const N2 = Object.values(allOprnds).
reduce((a, b: number) => a + b, 0);
    const n = n1 + n2;
    const N = N1 + N2;
    const Volume = n > 0 ? N * Math.log2(n) : 0;
    const Difficulty = n2 > 0 ? (n1 / 2) * (N2 /
```

```
n2) : 0;
    const Effort = Difficulty * Volume;

    return {
      n1,
      n2,
      N1,
      N2,
      Volume,
      Difficulty,
      Effort,
      count: selectedRows.length,
    };
  });

  // Listen for extraction progress
  $effect(() => {
    const handleProgress = (_event: unknown,
data: ProgressData) => {
      if (data.status === 'starting') {
        processingProgress = t(currentLang,
'processing', { file: data.file });
        processedFiles = Math.
max(processedFiles, 0); // Reset if needed
      } else if (data.status === 'completed') {
        processedFiles += 1;
        progressValue = (processedFiles /
totalFiles) * 100;
        processingProgress = t(currentLang,
'completed', {
          file: data.file,
          current: processedFiles,
          total: totalFiles,
        });
        if (processedFiles >= totalFiles) {
          isProcessing = false;
          processingProgress = '';
          progressValue = 0;
        }
      } else if (data.status === 'error') {
        processedFiles += 1;
        progressValue = (processedFiles /
totalFiles) * 100;
        processingProgress = t(currentLang,
'errorProcessing', { file: data.file });
        if (processedFiles >= totalFiles) {
          isProcessing = false;
          processingProgress = '';
          progressValue = 0;
        }
      }
    };
    window.api.onProgress(handleProgress);
    return () => window.api.
offProgress(handleProgress);
  });

  // Listen for analysis progress
  $effect(() => {
    const handleAnalysisProgress = (_event:
unknown, data: AnalysisProgressData) => {
      if (data.status === 'analyzing') {
        if (data.current != null && data.total !
= null) {
          progressValue = (data.current / data.
total) * 100;
          processingProgress = t(currentLang,
'analyzingFormulas', {
            current: data.current,
            total: data.total,
          });
        } else {
          progressValue = 0;
          processingProgress = t(currentLang,
'analyzing');
        }
      } else if (data.status === 'completed') {
        isProcessing = false;
        processingProgress = '';
        progressValue = 0;
      } else if (data.status === 'error') {
        isProcessing = false;
        processingProgress = t(currentLang,
'analysisFailed');
        progressValue = 0;
      }
    };
    window.
api.onAnalysisProgress(handleAnalysisProgress);
    return () => window.api.
offAnalysisProgress(handleAnalysisProgress);
  });

  // Listen for native menu events
  $effect(() => {
    const handleOpen = () => openAndParse();
    const handleSave = () =>
saveFormulasToFile();
    const handleDebug = (_: unknown, enabled:
boolean) => (debugMode = enabled);
    const handleLangEn = () => (currentLang =
'en');
    const handleLangUa = () => (currentLang =
'ua');

    window.api.onMenuOpen(handleOpen);
    window.api.onMenuSave(handleSave);
    window.api.onMenuDebug(handleDebug);
    window.api.onMenuLanguageEn(handleLangEn);
    window.api.onMenuLanguageUa(handleLangUa);

    return () => {
      window.api.offMenuOpen(handleOpen);
      window.api.offMenuSave(handleSave);
```

```
      window.api.offMenuDebug(handleDebug);
      window.api.
offMenuLanguageEn(handleLangEn);
      window.api.
offMenuLanguageUa(handleLangUa);
    };
  });

  function toggleFormula(id: string) {
    if (comparisonFormulaIds.has(id)) {
      if (hiddenFormulaIds.has(id)) {
        hiddenFormulaIds.delete(id);
      } else {
        hiddenFormulaIds.add(id);
      }
    } else {
      comparisonFormulaIds.add(id);
    }
  }

  function toggleLegendItem(id: string) {
    if (hiddenFormulaIds.has(id)) {
      hiddenFormulaIds.delete(id);
    } else {
      hiddenFormulaIds.add(id);
    }
  }

  function showStatus(message: string) {
    statusMessage = message;
    statusVisible = true;
    if (statusTimeout) window.
clearTimeout(statusTimeout);
    statusTimeout = window.setTimeout(() => {
      statusVisible = false;
    }, 10000);
  }

  async function openAndParse() {
    const selectedFiles = await window.api.
openFile();
    if (!selectedFiles || selectedFiles.length
=== 0) return;

    // Check if the first file is an .fmx file
    const firstFile = selectedFiles[0];
    const ext = firstFile.split('.').pop()?.
toLowerCase();

    if (ext === 'fmx') {
      // Load .fmx file as saved formulas
      try {
        const fileData = await window.api.
readFmxFile(firstFile);

        // Validate format version
        if (!fileData.version ||
parseFloat(fileData.version) > 1.0) {
          console.warn('Unknown file format
version:', fileData.version);
        }

        // Load formulas but ignore analysis
data for future compatibility
        formulaMetadata = fileData.formulas.
map((f: FormulaMetadata) => ({
          id: f.id,
          name: f.name,
          latex: f.latex,
          created: f.created,
          modified: f.modified,
          tags: f.tags || [],
          // Explicitly ignore analysis - will
be recalculated
        }));

        fileMetadata = fileData.metadata;
        filePath = fileMetadata.sourceFile ||
null;

        // Re-analyze all formulas with current
metrics implementation
        isProcessing = true;
        processingProgress = t(currentLang,
'analyzing');
        statusMessage = '';

        try {
          const formulas = formulaMetadata.
map((f) => f.latex);
          const result = await window.api.
analyzeFormulas(formulas);

          // Update metadata with fresh analysis
          formulaMetadata.forEach((f, i) => {
            if (result.rows[i]?.success) {
              f.analysis = result.rows[i].
halstead;
            }
          });

          // Re-select all formulas
          comparisonFormulaIds.clear();
          hiddenFormulaIds.clear();
          formulaMetadata.forEach((f) =>
comparisonFormulaIds.add(f.id));

          // Set analysis results for chart
          analysisResults.set(result);

          showStatus(t(currentLang,
'formulaLoaded'));
        } catch (error) {
          console.error('Failed to re-analyze
```

```
formulas:', error);
          showStatus(t(currentLang,
'analysisFailed'));
        } finally {
          isProcessing = false;
          processingProgress = '';
        }
      } catch (error) {
        extractionErrors = [t(currentLang,
'failedToProcess', { error: String(error) })];
        showStatus(t(currentLang,
'errorLoading'));
      }
      return;
    }

    // Handle .tex and .docx files through
extraction
    isProcessing = true;
    totalFiles = selectedFiles.length;
    processedFiles = 0;
    progressValue = 0;
    processingProgress = t(currentLang,
'startingExtraction');
    statusMessage = '';

    try {
      const results = await window.api.
extractFormulas(selectedFiles);

      // For simplicity, take the first file's
results
      const firstResult = results[0];
      filePath = firstResult.file;
      formulas = firstResult.formulas;
      extractionErrors = firstResult.errors;
      raw = ''; // Not applicable for word docs

      // If there are multiple files, maybe
aggregate or something, but for now just first
      if (results.length > 1) {
        extractionErrors.push(t(currentLang,
'processedMultiple', { count: results.
length }));
      }

      comparisonFormulaIds.clear();
      hiddenFormulaIds.clear();

      // Analyze formulas with worker - convert
Proxy to plain array for IPC
      const plainFormulas = [...formulas];
      const analysisResult = await window.api.
analyzeFormulas(plainFormulas);
      analysisResults.set(analysisResult);

      // Migrate to formula metadata
      migrateFormulasToMetadata(formulas,
analysisResult);
      fileMetadata.sourceFile = filePath ||
undefined;

      showStatus(
        t(currentLang, 'fileLoaded', {
          file: filePath?.split(/[\\/]/).pop()
|| '',
          count: formulas.length,
        })
      );
    } catch (error) {
      extractionErrors = [t(currentLang,
'failedToProcess', { error: String(error) })];
      formulas = [];
      raw = '';
      showStatus(t(currentLang,
'errorLoading'));
    } finally {
      isProcessing = false;
      processingProgress = '';
    }
  }

  function cancelProcessing() {
    window.api.cancelExtraction();
    isProcessing = false;
    processingProgress = t(currentLang,
'cancelled');
  }

  // Formula editing functions
  function startEditName(id: string,
currentName: string) {
    editingNameId = id;
    editingText = currentName;
    // Auto-select text on next tick
    window.setTimeout(() => {
      const input = document.
querySelector(`input[data-edit-name="${id}"]`);
      if (input instanceof HTMLInputElement)
input.select();
    }, 0);
  }

  function saveNameEdit(id: string) {
    if (editingNameId !== id) return;
    const formula = formulaMetadata.find((f) =>
f.id === id);
    if (formula) {
      formula.name =
        editingText.trim() ||
        `${t(currentLang, 'formula')}
${formulaMetadata.indexOf(formula) + 1}`;
      formula.modified = new Date().
toISOString();
```

```
    }
    editingNameId = null;
    editingText = '';
  }

  function cancelNameEdit() {
    editingNameId = null;
    editingText = '';
  }

  function startEditLatex(id: string,
currentLatex: string) {
    editingLatexId = id;
    editingText = currentLatex;
    // Auto-select text on next tick
    window.setTimeout(() => {
      const input = document.
querySelector(`input[data-edit-latex="${id}"]`);
      if (input instanceof HTMLInputElement)
input.select();
    }, 0);
  }

  function saveLatexEdit(id: string) {
    if (editingLatexId !== id) return;
    const formula = formulaMetadata.find((f) =>
f.id === id);
    if (formula && editingText.trim()) {
      formula.latex = editingText.trim();
      formula.modified = new Date().
toISOString();

      // Recalculate all metrics to update
overall stats
      recalculateAllMetrics();
    }
    editingLatexId = null;
    editingText = '';
  }

  function cancelLatexEdit() {
    editingLatexId = null;
    editingText = '';
  }

  // Formula naming

  // File operations
  async function saveFormulasToFile() {
    console.log('[SAVE] Starting save process..
.');

    // Convert to plain objects to avoid IPC
serialization issues
    const plainFormulas = formulaMetadata.
map((f) => {
      // Deep clone to remove any Svelte proxies
or non-serializable objects
      const plain = {
        id: f.id,
        name: f.name,
        latex: f.latex,
        created: f.created,
        modified: f.modified,
        analysis: f.analysis ? JSON.parse(JSON.
stringify(f.analysis)) : undefined,
        tags: f.tags ? [...f.tags] : [],
      };
      return plain;
    });

    const plainMetadata = {
      created: fileMetadata.created,
      modified: new Date().toISOString(),
      sourceFile: fileMetadata.sourceFile,
      description: fileMetadata.description,
    };

    const fileData: FormulaFile = {
      version: '1.0',
      metadata: plainMetadata,
      formulas: plainFormulas,
    };

    try {
      const savedPath = await window.api.
saveFormulas(fileData);
      if (savedPath) {
        fileMetadata.modified = new Date().
toISOString();
        showStatus(t(currentLang,
'formulaSaved'));
      }
    } catch (error) {
      console.error('[SAVE] Error during save:',
error);
      showStatus(t(currentLang,
'errorLoading')); // Reusing errorLoading for
simplicity or add errorSaving
    }
  }

  async function recalculateAllMetrics() {
    if (formulaMetadata.length === 0) {
      analysisResults.set(null);
      return;
    }

    const formulas = formulaMetadata.map((f) =>
f.latex);
    try {
      const result = await window.api.
analyzeFormulas(formulas);
```

```
      // Update metadata with fresh analysis
      formulaMetadata.forEach((f, i) => {
        if (result.rows[i]?.success) {
          f.analysis = result.rows[i].halstead;
        }
      });

      // Update analysis results store (this
updates the chart and overall metrics)
      analysisResults.set(result);
    } catch (error) {
      console.error('Failed to recalculate
metrics:', error);
    }
  }

  function addNewFormula() {
    const newFormula: FormulaMetadata = {
      id: crypto.randomUUID(),
      name: `${t(currentLang, 'formula')}
${formulaMetadata.length + 1}`,
      latex: '',
      created: new Date().toISOString(),
      modified: new Date().toISOString(),
      tags: [],
    };

    formulaMetadata = [...formulaMetadata,
newFormula];

    // Add empty row to analysis results
temporarily
    const currentResults: AnalysisResults =
$analysisResults || {
      overall: {
        n1: 0,
        n2: 0,
        N1: 0,
        N2: 0,
        Volume: 0,
        Difficulty: 0,
        Effort: 0,
      },
      rows: [],
      errors: [],
    };

    currentResults.rows.push({
      formula: '',
      halstead: {
        operators: {},
        operands: {},
        n1: 0,
        n2: 0,
        N1: 0,
        N2: 0,
        n: 0,
        N: 0,
        Volume: 0,
        Difficulty: 0,
        Effort: 0,
      },
      success: false,
      error: { message: t(currentLang,
'notYetAnalyzed') },
    });
    analysisResults.set(currentResults);

    // Automatically start editing the new
formula
    startEditLatex(newFormula.id, '');
  }

  function migrateFormulasToMetadata(formulas:
string[], analysisResults: AnalysisResults) {
    formulaMetadata = formulas.map((latex,
index) => ({
      id: crypto.randomUUID(),
      name: `${t(currentLang, 'formula')}
${index + 1}`,
      latex,
      created: new Date().toISOString(),
      modified: new Date().toISOString(),
      analysis: analysisResults.rows[index]?.
success
        ? analysisResults.rows[index].halstead
        : undefined,
      tags: [],
    }));

    // Select all for comparison initially after
migration
    formulaMetadata.forEach((f) =>
comparisonFormulaIds.add(f.id));
  }

  // Delete formula
  function startDeleteFormula(id: string) {
    deletingFormulaId = id;
  }

  function confirmDeleteFormula(id: string) {
    const index = formulaMetadata.findIndex((f)
=> f.id === id);
    if (index >= 0) {
      comparisonFormulaIds.delete(id);
      hiddenFormulaIds.delete(id);
      formulaMetadata = formulaMetadata.
filter((f) => f.id !== id);

      // Recalculate metrics to update overall
stats
      recalculateAllMetrics();
    }
```

```
    deletingFormulaId = null;
  }

  function cancelDeleteFormula() {
    deletingFormulaId = null;
  }
</script>

<div class="app-container">
  <div class="status-bar"
class:visible={statusVisible}>
    {statusMessage}
  </div>

  <div class="panels-wrapper">
    <section class="left-panel">
      {#if extractionErrors.length}
        <details>
          <summary class="error-summary"
            >{t(currentLang,
'extractionErrors')} ({extractionErrors.
length})</summary
          >
          <ul class="error-list">
            {#each extractionErrors as error
(error)}
              <li>{error}</li>
            {/each}
          </ul>
        </details>
      {/if}

      <div class="metrics-header">
        <h2>{t(currentLang, 'totalFormulas',
{ count: formulaMetadata.length })}</h2>
        <div class="sorting-controls">
          <select bind:value={sortBy}
class="sort-select" title={t(currentLang,
'sortBy')}>
            <option
value="created">{t(currentLang, 'date')}</
option>
            <option value="name">{t(currentLang,
'sortName')}</option>
            <option
value="volume">{t(currentLang, 'volume')}</
option>
            <option
value="difficulty">{t(currentLang,
'difficulty')}</option>
            <option
value="effort">{t(currentLang, 'effort')}</
option>
          </select>
          <select
bind:value={sortOrder} class="sort-select"
title={t(currentLang, 'sortOrder')}>
            <option value="asc">↑
{t(currentLang, 'asc')}</option>
            <option value="desc">↓
{t(currentLang, 'desc')}</option>
          </select>
        </div>
      </div>

      {#if $analysisResults && $analysisResults.
errors.length}
        <details>
          <summary class="warning-summary"
            >{t(currentLang, 'analysisErrors')}
({$analysisResults.errors.length})</summary
          >
          <ul class="warning-list">
            {#each $analysisResults.errors as
error (error)}
              <li><code>{error.formula ||
'Unknown'}</code>: {error.message}</li>
            {/each}
          </ul>
        </details>
      {/if}

      <ul class="formula-list">
        {#each sortedFormulas as formula
(formula.id)}
          {@const originalIdx = formulaMetadata.
indexOf(formula)}
          <li class="formula-card">
            {#if deletingFormulaId === formula.
id}
              <div class="delete-confirm-
buttons">
                <button
                  class="confirm-delete-btn"
                  onclick={() =>
confirmDeleteFormula(formula.id)}
                  title={t(currentLang,
'confirmDelete')}
                >
                  ✓
                </button>
                <button
                  class="cancel-delete-btn"
                  onclick={() =>
cancelDeleteFormula()}
                  title={t(currentLang,
'cancelDelete')}
                >
                  ×
                </button>
              </div>
            {:else}
              <button
                class="delete-btn"
```

```
                onclick={() =>
startDeleteFormula(formula.id)}
                title={t(currentLang,
'deleteFormula')}
              >
                🗑️
              </button>
            {/if}
            <div class="formula-header">
              <input
                type="checkbox"
                checked={comparisonFormulaIds.
has(formula.id) && !hiddenFormulaIds.has(formula.
id)}
                onchange={() =>
toggleFormula(formula.id)}
                title={t(currentLang,
'includeInComparison')}
              />

              {#if editingNameId === formula.id}
                <input
                  type="text"
                  bind:value={editingText}
                  data-edit-name={formula.id}
                  class="formula-name-input
editing"
                  placeholder={t(currentLang,
'formulaName')}
                  onblur={() =>
saveNameEdit(formula.id)}
                  onkeydown={(e) => {
                    if (e.key === 'Enter')
saveNameEdit(formula.id);
                    if (e.key === 'Escape')
cancelNameEdit();
                  }}
                />
              {:else}
                <div class="editable-field">
                  <span
                    class="formula-name-text"
                    style:background-
color={$analysisResults?.rows[originalIdx]?.
success
                      ?
getEffortColor($analysisResults.
rows[originalIdx].halstead.Effort)
                      : 'transparent'}
                    style="border-radius: 4px;"
                  >
                    {formula.name}
                  </span>
                  <button
                    class="edit-icon"
                    onclick={() =>
startEditName(formula.id, formula.name)}
                    title={t(currentLang,
'editName')}
                  >
                    ✎
                  </button>
                </div>
              {/if}
            </div>

            <details class="formula-details">
              <summary>
                <!-- eslint-disable-next-line
svelte/no-at-html-tags -->
                {@html katex.
renderToString(formula.latex, { throwOnError:
false })}
              </summary>
              <div>
                <div class="latex-source-line">
                  <strong>{t(currentLang,
'rawLatex')}:</strong>
                  {#if editingLatexId ===
formula.id}
                    <div class="latex-
edit-container">
                      <input
                        type="text"
                        bind:value={editingText}
                        data-edit-latex={formula.
id}
                        class="latex-edit-
input editing"
                        placeholder={t(currentLang, 'enterLatex')}
                        onblur={() =>
saveLatexEdit(formula.id)}
                        onkeydown={(e) => {
                          if (e.key === 'Enter')
saveLatexEdit(formula.id);
                          if (e.key ===
'Escape') cancelLatexEdit();
                        }}
                      />
                    </div>
                  {:else}
                    <div class="editable-
field inline">
                      <code class="latex-code-
text">{formula.latex}</code>
                      <button
                        class="edit-icon"
                        onclick={() =>
startEditLatex(formula.id, formula.latex)}
                        title={t(currentLang,
'editFormula')}
                      >
                        ✎
```

```
                    </button>
                  </div>
                {/if}
              </div>
              {#if $analysisResults?.
rows[originalIdx]?.success}
                <h4>{t(currentLang,
'metrics')}</h4>
                <dl class="metrics-grid">
                  <dt title={t(currentLang,
'tooltipN1')}>
                    {t(currentLang,
'uniqueOperators')}:
                  </dt>
                  <dd>{$analysisResults.
rows[originalIdx].halstead.n1}</dd>
                  <dt title={t(currentLang,
'tooltipN2')}>{t(currentLang,
'uniqueOperands')}:</dt>
                  <dd>{$analysisResults.
rows[originalIdx].halstead.n2}</dd>
                  <dt title={t(currentLang,
'tooltipN1Total')}>
                    {t(currentLang,
'totalOperators')}:
                  </dt>
                  <dd>{$analysisResults.
rows[originalIdx].halstead.N1}</dd>
                  <dt title={t(currentLang,
'tooltipN2Total')}>
                    {t(currentLang,
'totalOperands')}:
                  </dt>
                  <dd>{$analysisResults.
rows[originalIdx].halstead.N2}</dd>
                  <dt title={t(currentLang,
'tooltipVolume')}>{t(currentLang, 'volume')}:</
dt>
                  <dd>{$analysisResults.
rows[originalIdx].halstead.Volume.toFixed(3)}</
dd>
                  <dt title={t(currentLang,
'tooltipDifficulty')}>
                    {t(currentLang,
'difficulty')}:
                  </dt>
                  <dd>{$analysisResults.
rows[originalIdx].halstead.Difficulty.
toFixed(3)}</dd>
                  <dt title={t(currentLang,
'tooltipEffort')}>{t(currentLang, 'effort')}:</
dt>
                  <dd>{$analysisResults.
rows[originalIdx].halstead.Effort.toFixed(3)}</
dd>
                </dl>
                <div style="margin-top:
1rem;">
                  <strong>{t(currentLang,
'operators')}:</strong>
                  <div class="token-list">
                    {#each Object.
entries($analysisResults.rows[originalIdx].
halstead.operators) as [op, count], idx (op)}
                      {op}×{count}{idx <
                      Object.
entries($analysisResults.rows[originalIdx].
halstead.operators)
                        .length -
                        1
                        ? ', '
                        : ''}
                    {/each}
                  </div>
                  <strong>{t(currentLang,
'operands')}:</strong>
                  <div class="token-list">
                    {#each Object.
entries($analysisResults.rows[originalIdx].
halstead.operands) as [op, count], idx (op)}
                      {op}×{count}{idx <
                      Object.
entries($analysisResults.rows[originalIdx].
halstead.operands)
                        .length -
                        1
                        ? ', '
                        : ''}
                    {/each}
                  </div>
                </div>
              {:else}
                <strong class="error-
text">{t(currentLang, 'failed')}:</strong>
                  {$analysisResults?.
rows[originalIdx]?.error?.message || 'Unknown
error'}
              {/if}
            </div>
          </details>
        </li>
      {/each}
    </ul>

    <div class="add-formula-container">
      <button onclick={addNewFormula}
class="add-btn">
        + {t(currentLang, 'addFormula')}
      </button>
    </div>
  </section>

  <section class="right-panel">
    {#if selectedOverall}
```

```
        <div class="charts-container">
          {#if volumeChartData}
            <div class="chart-section">
              <h4>{t(currentLang, 'volume')}
</h4>
              <Chart data={volumeChartData} />
            </div>
          {/if}

          {#if difficultyChartData}
            <div class="chart-section">
              <h4>{t(currentLang, 'difficulty')}
</h4>
              <Chart
data={difficultyChartData} />
            </div>
          {/if}

          {#if effortChartData}
            <div class="chart-section">
              <h4>{t(currentLang, 'effort')}
</h4>
              <Chart data={effortChartData} />
            </div>
          {/if}
        </div>

        <div class="legend-container">
          <h3>{t(currentLang,
'metricComparison')}</h3>
          <div class="custom-legend">
            {#each sortedFormulas.filter( (f)
=> comparisonFormulaIds.has(f.id) ) as formula
(formula.id)}
              {@const originalIdx
= formulaMetadata.findIndex((f) => f.id ===
formula.id)}
              {@const isHidden =
hiddenFormulaIds.has(formula.id)}
              <button
                class="legend-item"
                class:hidden={isHidden}
                onclick={() =>
toggleLegendItem(formula.id)}
                title={isHidden ? t(currentLang,
'show') : t(currentLang, 'hide')}
              >
                <span
                  class="color-box"
                  style:background-
color={isHidden
                    ? '#ccc'
                    : `hsl(${(originalIdx *
360) / formulaMetadata.length}, 70%, 50%)`}
                ></span>
                <span class="legend-
label">{formula.name}</span>
              </button>
            {/each}
          </div>
        </div>

        <details class="overall-metrics-section"
open>
          <summary class="overall-metrics-
summary">
            <h3
              class="overall-metrics-title"
              title="{t(currentLang,
'successful')}
{t(currentLang, 'halsteadAnalysis')}"
            >
              {t(currentLang, 'overallMetrics')}
            </h3>
          </summary>
          <div class="overall-metrics-body">
            <p class="overall-metrics-subtitle">
              ({selectedOverall.count}
              {t(currentLang, 'successful').
toLowerCase()}
              {t(currentLang,
'halsteadAnalysis').toLowerCase()})
            </p>
          </div>
          <dl class="metrics-grid overall">
            <dt title={t(currentLang,
'tooltipN1')}>{t(currentLang,
'uniqueOperators')}:</dt>
            <dd>{selectedOverall.n1}</dd>
            <dt title={t(currentLang,
'tooltipN2')}>{t(currentLang,
'uniqueOperands')}:</dt>
            <dd>{selectedOverall.n2}</dd>
            <dt title={t(currentLang,
'tooltipN1Total')}>{t(currentLang,
'totalOperators')}:</dt>
            <dd>{selectedOverall.N1}</dd>
            <dt title={t(currentLang,
'tooltipN2Total')}>{t(currentLang,
'totalOperands')}:</dt>
            <dd>{selectedOverall.N2}</dd>
            <dt title={t(currentLang,
'tooltipVolume')}>{t(currentLang, 'volume')}:</
dt>
            <dd>{selectedOverall.Volume.
toFixed(3)}</dd>
            <dt title={t(currentLang,
'tooltipDifficulty')}>
              {t(currentLang, 'difficulty')}:
            </dt>
            <dd>{selectedOverall.Difficulty.
toFixed(3)}</dd>
            <dt title={t(currentLang,
'tooltipEffort')}>{t(currentLang, 'effort')}:</
```

```
dt>
            <dd>{selectedOverall.Effort.
toFixed(3)}</dd>
          </dl>
        </details>
      {/if}

      {#if debugMode}
        <h2>{t(currentLang, 'debugInfo')}</h2>

        {#if raw}
          <h3>{t(currentLang,
'rawLatexContent')}</h3>
          <pre class="debug-pre">{raw}</pre>
        {/if}

        {#if $analysisResults}
          <h3>{t(currentLang,
'analysisResultsRaw')}</h3>
          <details>
            <summary>{t(currentLang,
'clickToViewRaw')}</summary>
            <pre class="debug-pre">{JSON.
stringify($analysisResults, null, 2)}</pre>
          </details>
        {/if}
      {/if}
    </section>
  </div>
</div>

{#if isProcessing}
  <div class="processing-overlay">
    <div class="processing-modal">
      <div style="margin-bottom: 1rem;">
        <div class="spinner"></div>
      </div>
      <p>{processingProgress || t(currentLang,
'processing', { file: '' })}</p>
      {#if progressValue > 0}
        <div style="margin: 1rem 0;">
          <progress value={progressValue}
max="100" style="width: 100%;"></progress>
          <p style="margin-top: 0.5rem; font-
size: 0.9em; color: #666;">
            {t(currentLang, 'percentComplete',
{ percent: Math.round(progressValue) })}
          </p>
        </div>
      {/if}
      <button onclick={cancelProcessing}
class="cancel-btn">{t(currentLang, 'cancel')}</
button>
    </div>
  </div>
{/if}
```

```
<style>
  .app-container {
    display: flex;
    flex-direction: column;
    height: 100vh;
    background: #fcfcfc;
  }

  .panels-wrapper {
    display: grid;
    grid-template-columns: 0.6fr 1fr;
    flex: 1;
    overflow: hidden;
  }

  .left-panel,
  .right-panel {
    padding: 1rem;
    overflow-y: auto;
  }

  .left-panel {
    border-right: 1px solid #eee;
  }

  .status-bar {
    position: fixed;
    top: 1rem;
    right: 1rem;
    background: #e8f5e9;
    color: #2e7d32;
    padding: 0.75rem 1.25rem;
    border-radius: 8px;
    border: 1px solid #c8e6c9;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);
    z-index: 2000;
    opacity: 0;
    transform: translateY(-20px);
    transition: all 0.4s cubic-bezier(0.175, 0.
885, 0.32, 1.275);
    pointer-events: none;
  }

  .status-bar.visible {
    opacity: 1;
    transform: translateY(0);
    pointer-events: auto;
  }

  .metrics-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 1rem;
    border-bottom: 2px solid #eee;
    padding-bottom: 0.5rem;
  }
```

```
.sorting-controls {
  display: flex;
  gap: 0.5rem;
}

.sort-select {
  padding: 0.25rem 0.5rem;
  border-radius: 4px;
  border: 1px solid #ccc;
  background: white;
  font-size: 0.9rem;
  cursor: pointer;
  outline: none;
}

.sort-select:hover {
  border-color: #bbb;
}

.formula-list {
  list-style: none;
  padding: 0;
}

.formula-card {
  margin-bottom: 1em;
  border: 1px solid #ddd;
  border-radius: 4px;
  padding: 0.75rem;
  background: #fafafa;
  position: relative;
}

.formula-header {
  display: flex;
  align-items: center;
  gap: 0.5rem;
  margin-bottom: 0.5rem;
}

/* Inline edit styles */
.editable-field {
  display: inline-flex;
  align-items: center;
  gap: 0.25rem;
  position: relative;
}

.editable-field.inline {
  display: inline-flex;
}

.formula-name-text {
  font-weight: 500;
  padding: 0.25rem 0.5rem;
  min-width: 100px;
}

.latex-code-text {
  padding: 0.25rem 0.5rem;
}

.edit-icon {
  opacity: 0;
  background: transparent;
  border: none;
  color: #3498db;
  cursor: pointer;
  padding: 0.25rem;
  font-size: 14px;
  transition:
    opacity 0.2s,
    color 0.2s;
  line-height: 1;
}

.editable-field:hover .edit-icon {
  opacity: 1;
}

.edit-icon:hover {
  color: #2980b9;
}

.formula-name-input {
  width: 200px;
  padding: 0.25rem 0.5rem;
  border: 1px solid #ccc;
  border-radius: 3px;
  font-weight: 500;
  transition: border-color 0.2s;
}

.formula-name-input.editing {
  border: 2px solid #3498db;
  outline: none;
}

.latex-source-line {
  margin-bottom: 1rem;
}

.latex-edit-container {
  margin-top: 0.5rem;
}

.latex-edit-input {
  width: 100%;
  padding: 0.5rem;
  border: 1px solid #ccc;
  border-radius: 3px;
  font-family: monospace;
  font-size: 0.9em;
```

```
  transition: border-color 0.2s;
}

.latex-edit-input.editing {
  border: 2px solid #3498db;
  outline: none;
}

.delete-btn {
  position: absolute;
  top: 0.5rem;
  right: 0.5rem;
  color: white;
  border: none;
  border-radius: 3px;
  width: 24px;
  height: 24px;
  cursor: pointer;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 14px;
  font-weight: bold;
  z-index: 10;
  background: #95a5a6;
  opacity: 0;
  transition: all 0.2s;
}

.formula-card:hover .delete-btn {
  opacity: 1;
}

.delete-btn:hover {
  background: #34495e !important;
}

.delete-confirm-buttons {
  position: absolute;
  top: 0.5rem;
  right: 0.5rem;
  display: flex;
  gap: 0.25rem;
  z-index: 10;
}

.confirm-delete-btn,
.cancel-delete-btn {
  color: white;
  border: none;
  border-radius: 3px;
  width: 24px;
  height: 24px;
  cursor: pointer;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 16px;
  font-weight: bold;
  transition: all 0.2s;
}

.confirm-delete-btn {
  background: #27ae60;
}

.confirm-delete-btn:hover {
  background: #229954;
}

.cancel-delete-btn {
  background: #e74c3c;
}

.cancel-delete-btn:hover {
  background: #c0392b;
}

.metrics-grid {
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: 0.5rem;
  margin: 0;
}

.metrics-grid dd {
  border-bottom: 1px solid #eee;
  margin: 0;
}

.metrics-grid.overall {
  background: #f0f4c3;
  padding: 1rem;
  border-radius: 4px;
}

.charts-container {
  display: flex;
  flex-direction: column;
  gap: 1rem;
  margin-bottom: 1.5rem;
}

.legend-container {
  margin-bottom: 1.5rem;
  border-top: 1px solid #eee;
  padding-top: 0.75rem;
}

.custom-legend {
  display: flex;
  flex-wrap: wrap;
  gap: 0.4rem;
  margin-bottom: 1rem;
```

```
  padding: 0.5rem;
  background: #fdfdfd;
  border-radius: 6px;
  border: 1px solid #eee;
}

.legend-item {
  display: flex;
  align-items: center;
  gap: 0.35rem;
  padding: 0.25rem 0.5rem;
  background: white;
  border: 1px solid #ddd;
  border-radius: 4px;
  cursor: pointer;
  font-size: 0.85rem;
  transition: all 0.2s;
}

.legend-item:hover {
  border-color: #bbb;
  transform: translateY(-1px);
  box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}

.legend-item.hidden {
  background: #f5f5f5;
  color: #888;
  border-color: #eee;
  box-shadow: none;
}

.legend-item.hidden:hover {
  background: #eee;
}

.color-box {
  width: 12px;
  height: 12px;
  border-radius: 50%;
  flex-shrink: 0;
}

.chart-section {
  background: white;
  padding: 0.5rem 1rem;
  border-radius: 8px;
  border: 1px solid #eee;
  height: 28vh;
  display: flex;
  flex-direction: column;
}

.chart-section h4 {
  margin-top: 0;
  margin-bottom: 0.25rem;
  text-align: center;
  color: #333;
  border-bottom: 1px solid #f0f0f0;
  padding-bottom: 0.25rem;
  font-size: 0.9rem;
}

.overall-metrics-section {
  margin-top: 2rem;
  margin-bottom: 0;
}

.overall-metrics-summary {
  cursor: pointer;
  margin-bottom: 0.5rem;
  padding: 0.25rem 0;
}

.overall-metrics-title {
  margin: 0;
  display: inline;
  vertical-align: middle;
}

.overall-metrics-body {
  margin-bottom: 1rem;
}

.overall-metrics-subtitle {
  font-size: 0.9em;
  color: #7f8c8d;
  margin: 0 0 1rem 1.5rem; /* Indent to align
with text under triangle */
  font-style: italic;
  font-weight: 500;
}

.token-list {
  font-size: 0.9em;
  color: #444;
  margin-left: 1em;
}

.add-formula-container {
  margin-top: 1rem;
  margin-bottom: 2rem;
}

.add-btn {
  padding: 0.5rem 1rem;
  background: #27ae60;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  width: 100%;
}
```

```
.processing-overlay {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: rgba(0, 0, 0, 0.5);
  display: flex;
  align-items: center;
  justify-content: center;
  z-index: 1000;
}

.processing-modal {
  background: white;
  padding: 2rem;
  border-radius: 8px;
  text-align: center;
  min-width: 300px;
}

.spinner {
  border: 4px solid #f3f3f3;
  border-top: 4px solid #3498db;
  border-radius: 50%;
  width: 40px;
  height: 40px;
  animation: spin 2s linear infinite;
  margin: 0 auto;
}

.cancel-btn {
  margin-top: 1rem;
  padding: 0.5rem 1rem;
  background: #dc3545;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

.debug-pre {
  background: #f5f5f5;
  padding: 1rem;
  overflow: auto;
  max-height: 200px;
}

.error-summary,
.error-list {
  color: #d32f2f;
}
.warning-summary,
.warning-list {
  color: #f57c00;
}
.error-text {
  color: #f57c00;
}

@keyframes spin {
  0% {
    transform: rotate(0deg);
  }
  100% {
    transform: rotate(360deg);
  }
}

:global(body) {
  overflow: hidden;
  margin: 0;
  padding: 0;
}
</style>
```

2.9. Вміст файлу components/Chart.svelte

```
<script lang="ts">
  import Chart from 'chart.js/auto';

  let { data } = $props();

  let canvas = $state<HTMLCanvasElement>();
  let chart = $state<Chart | null>(null);

  $effect(() => {
    if (canvas && !chart) {
      // Create chart even with empty data
      const emptyData = {
        labels: [],
        datasets: [],
      };
      chart = new Chart(canvas, {
        type: 'bar',
        data: data || emptyData,
        options: {
          responsive: true,
          maintainAspectRatio: false,
          plugins: {
            legend: {
              display: false,
            },
          },
          scales: {
            y: {
              beginAtZero: true,
            },
          },
        },
      });
    }
  });
```

```
  $effect(() => {
    if (chart) {
      if (data) {
        chart.data = data;
        chart.update();
      } else {
        // Clear the chart
        chart.data = { labels: [], datasets:
[] };
        chart.update();
      }
    }
  });
</script>

<div style="flex: 1; min-height: 0;">
  <canvas bind:this={canvas} style="width: 100%;
height: 100%;"></canvas>
</div>
```

2.10. Вміст файлу `main.ts`

```
import App from './components/App.svelte';
import { mount } from 'svelte';

mount(App, { target: document.
getElementById('app')! });
```

2.11. Вміст файлу `lib/halsted.ts`

```
// src/halstead.ts
import { ComputeEngine } from '@cortex-js/
compute-engine';

export type Halstead = {
  n1: number;
  n2: number;
  N1: number;
  N2: number;
  n: number;
  N: number;
  Volume: number;
  Difficulty: number;
  Effort: number;
  operators: Record<string, number>;
  operands: Record<string, number>;
};

export type AnalysisError = {
  type: 'latex_parse' | 'math_parse' |
'extraction' | 'calculation';
  message: string;
  formula?: string;
  details?: unknown;
};

export type AnalysisResult =
    | {
        success: true;
        halstead: Halstead;
      }
    | {
        success: false;
        error: AnalysisError;
      };

// Simple logging system
const logger = {
  debug: (message: string, data?: unknown) => {
    if (process.env.NODE_ENV === 'development')
{
      console.log(`[DEBUG] ${message}`, data ||
'');
    }
  },
  warn: (message: string, data?: unknown) => {
    console.warn(`[WARN] ${message}`, data ||
'');
  },
  error: (message: string, data?: unknown) => {
    console.error(`[ERROR] ${message}`, data ||
'');
  },
};

const ce = new ComputeEngine();

/** Preprocess LaTeX to normalize complex
constructs for better parsing */
function preprocessLatex(latex: string): string
{
  let processed = latex;

  // Normalize matrix environments to array
notation
  processed = processed.replace(/\
\begin\{[pbv]matrix\}/g, '\\begin{bmatrix}');
  processed = processed.replace(/\
\end\{[pbv]matrix\}/g, '\\end{bmatrix}');

  // Normalize cases where matrices might be
written differently
  processed = processed.replace(/\
\begin\{array\}\{[^}]*\}/g, '\\begin{bmatrix}');
  processed = processed.replace(/\\end\{array\}/
g, '\\end{bmatrix}');

  // Normalize some common function notations
  processed = processed.replace(/\
\operatorname\{([^}]+)\}/g, '$1');

  // CRITICAL FIX: Convert colon ratio operator
to division
  // ComputeEngine doesn't support : as an
```

```
operator, so convert to /
  // This must be done BEFORE normalizing LaTeX
spacing commands
  processed = processed.replace(/\s*:\s*/g, ' /
');

  // Normalize spacing in some constructs (LaTeX
spacing commands: \, \: \;)
  processed = processed.replace(/\\,/g, ' '); //
thin space to regular space
  processed = processed.replace(/\\\s*:/g, '
'); // medium space (\:) to regular space
  processed = processed.replace(/\\;/g, ' '); //
thick space to regular space

  // Normalize some differential operators
  processed = processed.replace(/\\partial/g,
'∂');
  processed = processed.replace(/\\nabla/g,
'∇');

  // Handle some complex function notations
  processed = processed.replace(/\
\lim_\{([^}]+)\}/g, '\\lim_{$1}');

  return processed;
}

/** Remove outer math delimiters if present */
function stripDelimiters(s: string): string {
  let t = s.trim();
  if (t.startsWith('$$') && t.endsWith('$$'))
return t.slice(2, -2).trim();
  if (t.startsWith('$') && t.endsWith('$'))
return t.slice(1, -1).trim();
  if (t.startsWith('\\(') && t.endsWith('\\)'))
return t.slice(2, -2).trim();
  if (t.startsWith('\\[') && t.endsWith('\\]'))
return t.slice(2, -2).trim();
  // envs (equation/align/etc.) should already
be stripped by the extractor
  return t;
}

/** Fallback parser for expressions that Compute
Engine can't handle */
function fallbackParse(latex: string): string[]
| null {
  const ops: string[] = [];
  const oprnds: string[] = [];

  // Detect LaTeX operators with proper pattern
matching and count ALL occurrences
  const opPatterns = [
    { pattern: /\\cdot/g, name: 'Multiply' },
    { pattern: /\\times/g, name: 'Multiply' },
    { pattern: /:/g, name: 'Divide' },
    { pattern: /\//g, name: 'Divide' },
    { pattern: /\\frac/g, name: 'Divide' },
    { pattern: /=/g, name: 'Equal' },
    { pattern: /\\neq/g, name: 'NotEqual' },
    { pattern: /\\le/g, name: 'LessEqual' },
    { pattern: /\\ge/g, name: 'GreaterEqual' },
    { pattern: /</g, name: 'Less' },
    { pattern: />/g, name: 'Greater' },
    { pattern: /\\approx/g, name: 'Approx' },
    { pattern: /\\sim/g, name: 'Similar' },
    { pattern: /\\equiv/g, name: 'Equivalent' },
    { pattern: /\\land/g, name: 'And' },
    { pattern: /\\lor/g, name: 'Or' },
    { pattern: /\\neg/g, name: 'Not' },
    { pattern: /\\implies/g, name: 'Implies' },
    { pattern: /\\iff/g, name: 'Equivalent' },
    { pattern: /\\mapsto/g, name: 'MapsTo' },
    { pattern: /\\colon/g, name: 'Colon' },
    { pattern: /\\to/g, name: 'To' },
    { pattern: /\\rightarrow/g, name: 'To' },
    { pattern: /\\leftarrow/g, name:
'LeftArrow' },
    { pattern: /\\oplus/g, name: 'CircleAdd' },
    { pattern: /\\otimes/g, name:
'CircleMultiply' },
    { pattern: /\\in/g, name: 'Element' },
    { pattern: /\\ni/g, name: 'Contains' },
    { pattern: /\\subset/g, name: 'Subset' },
    { pattern: /\\cup/g, name: 'Union' },
    { pattern: /\\cap/g, name: 'Intersection' },
    { pattern: /\\pm/g, name: 'PlusMinus' },
    { pattern: /\\mp/g, name: 'MinusPlus' },
    { pattern: /\+/g, name: 'Add' },
    { pattern: /\(/g, name: 'Parenthesis' },
    { pattern: /\)/g, name: 'Parenthesis' },
    { pattern: /\[/g, name: 'Bracket' },
    { pattern: /\]/g, name: 'Bracket' },
    { pattern: /\\langle/g, name:
'AngleBracket' },
    { pattern: /\\rangle/g, name:
'AngleBracket' },
    { pattern: /\\{/g, name: 'Brace' },
    { pattern: /\\}/g, name: 'Brace' },
    // Check for minus but avoid matching
negative signs in LaTeX commands
    { pattern: /(?<!\\)-/g, name: 'Subtract' },
  ];

  for (const { pattern, name } of opPatterns) {
    const matches = latex.match(pattern);
    if (matches) {
      for (let i = 0; i < matches.length; i++) {
        ops.push(name);
      }
    }
  }
```

```
  // Extract variables including subscripted
ones (e.g., l_{30}, x, \alpha, C_A)
  // Pattern matches: single letters, Greek
letters, and subscripted variables (with or
without braces)
  const varPattern = /\\?([a-zA-Z]+)
(?:_\{([^}]+)\}|_([a-zA-Z0-9]))?/g;
  let match;

  // Track the content we've already processed
as variables to avoid double-counting in numbers
  const processedRanges: Array<[number, number]>
= [];

  while ((match = varPattern.exec(latex)) !==
null) {
    const start = match.index;
    const end = varPattern.lastIndex;
    const baseName = match[1];
    const subscript = match[2] || match[3];

    // Skip LaTeX structural and formatting
commands
    const latexCommands = [
      'cdot',
      'times',
      'frac',
      'langle',
      'rangle',
      'begin',
      'end',
      'left',
      'right',
      'neq',
      'le',
      'ge',
      'approx',
      'sim',
      'equiv',
      'land',
      'lor',
      'neg',
      'implies',
      'iff',
      'oplus',
      'otimes',
      'pm',
      'mp',
      'in',
      'ni',
      'subset',
      'cup',
      'cap',
      'mapsto',
      'colon',
      'to',
      'rightarrow',
      'leftarrow',
      'lfloor',
      'rfloor',
      'lceil',
      'rceil',
      'partial',
      'nabla',
      'Delta',
      'infty',
      'text',
      'mathrm',
      'mathbf',
      'mathit',
      'mathsf',
      'mathtt',
    ];

    if (latexCommands.includes(baseName)) {
      continue;
    }

    // Build variable name with subscript if
present
    const varName = subscript ? `${baseName}
_${subscript}` : baseName;

    // Add to operands (allow duplicates for
correct Halstead N2 count)
    oprnds.push(varName);
    processedRanges.push([start, end]);
  }

  // Extract standalone numbers, avoiding those
already inside subscript variables
  const numPattern = /\d+(\.\d+)?/g;
  while ((match = numPattern.exec(latex)) !==
null) {
    const start = match.index;
    const end = numPattern.lastIndex;

    // Check if this number is inside a
previously processed variable/subscript
    const isInside = processedRanges.
some(([pStart, pEnd]) => start >= pStart && end
<= pEnd);
    if (!isInside) {
      oprnds.push(match[0]);
    }
  }

  // Return null if nothing found
  if (ops.length === 0 && oprnds.length === 0) {
    return null;
  }

  // Build a nested structure that includes ALL
operators
```

```
  // This allows collectFromJson to extract all operators
  // Structure: [FirstOp, [SecondOp, [ThirdOp, ...operands]]]
  let result: unknown = oprnds;

  // Build from the inside out, nesting each operator
  for (let i = ops.length - 1; i >= 0; i--) {
    if (i === ops.length - 1) {
      // Innermost: operator with all operands
      result = [ops[i], ...oprnds];
    } else {
      // Wrap previous result in next operator
      result = [ops[i], result];
    }
  }

  return result as string[];
}

/** Recursively check if a node contains an error or raw LaTeX commands that CE missed */
function isErrorNode(node: unknown): boolean {
  if (Array.isArray(node)) {
    if (node[0] === 'Error') return true;
    return node.some(isErrorNode);
  }
  if (typeof node === 'object' && node !== null) {
    return Object.values(node).some(isErrorNode);
  }
  if (typeof node === 'string') {
    // If we see a raw LaTeX command in a string node, it means CE didn't parse it correctly
    if (node.startsWith('\\')) return true;
    const rawCommands = new Set([
      'oplus',
      'otimes',
      'neq',
      'land',
      'lor',
      'mapsto',
      'colon',
      'subset',
      'in',
      'ni',
    ]);
    if (rawCommands.has(node)) return true;
  }
  return false;
}

/** Parse LaTeX into MathJSON (plain JSON) */
function latexToJson(
  latex: string
): { success: true; json: unknown } | { success: false; error: AnalysisError } {
  try {
    const stripped = stripDelimiters(latex);
    const preprocessed = preprocessLatex(stripped);
    logger.debug(`Parsing LaTeX: ${preprocessed}`);

    const parsed = ce.parse(preprocessed);
    if (!parsed || typeof parsed.json === 'undefined' || isErrorNode(parsed.json)) {
      logger.warn(`Compute Engine failed or returned errors: ${preprocessed}, trying fallback`);

      // Try fallback parsing
      const fallback = fallbackParse(preprocessed);
      if (fallback) {
        logger.debug(`Fallback parsing succeeded`, fallback);
        return { success: true, json: fallback };
      }
    }

    if (!parsed || typeof parsed.json === 'undefined') {
      return {
        success: false,
        error: {
          type: 'math_parse',
          message: 'Compute Engine could not parse the LaTeX expression',
          formula: latex,
          details: { stripped, preprocessed },
        },
      };
    }

    logger.debug(`Successfully parsed to MathJSON`, parsed.json);
    return { success: true, json: parsed.json };
  } catch (error) {
    logger.error(`Exception during LaTeX parsing: ${error}`, { latex, error });

    // Try fallback parsing even on exception
    try {
      const stripped = stripDelimiters(latex);
      const preprocessed = preprocessLatex(stripped);
      const fallback = fallbackParse(preprocessed);
      if (fallback) {
        logger.debug(`Fallback parsing succeeded
```

```
after exception`, fallback);
        return { success: true, json:
fallback };
      }
    } catch (fallbackError) {
      logger.error(`Fallback parsing also
failed: ${fallbackError}`);
    }

    return {
      success: false,
      error: {
        type: 'math_parse',
        message: `Parse error: ${error
instanceof Error ? error.message : 'Unknown
error'}`,
        formula: latex,
        details: error,
      },
    };
  }
}

/** Check if a head should be treated as a
function rather than an operator */
function isFunctionHead(head: string): boolean {
  // Common mathematical functions that take
arguments
  const functions = new Set([
    'Sin',
    'Cos',
    'Tan',
    'Log',
    'Ln',
    'Exp',
    'Sqrt',
    'Abs',
    'Derivative',
    'Integral',
    'Sum',
    'Product',
    'Limit',
    'Factorial',
    'Gamma',
    'Beta',
    'Zeta',
    'Binomial',
    'Min',
    'Max',
    'Gcd',
    'Lcm',
    'Floor',
    'Ceil',
    'Round',
    'Matrix',
    'Determinant',
    'Trace',
    'Transpose',
    'NotEqual',
    'Less',
    'Greater',
    'LessEqual',
    'GreaterEqual',
    'And',
    'Or',
    'Not',
    'Implies',
    'Equivalent',
    'Subset',
    'Intersection',
    'Union',
    'In',
    'Element',
    'Contains',
    'NotIn',
    'SetMinus',
    'MapsTo',
    'To',
    'LeftArrow',
    'Colon',
  ]);

  return functions.has(head) || head.
startsWith('Function');
}

/** Collect operators (heads) and operands
(symbols/numbers) from MathJSON */
function collectFromJson(node: unknown, ops:
string[], oprnds: string[]) {
  if (node == null) return;

  // Array form: [head, arg1, arg2, ...]
  if (Array.isArray(node)) {
    const head = node[0];
    if (typeof head === 'string' && head.length)
{
      // Classify head as operator or function
      if (isFunctionHead(head)) {
        // Functions are treated as operators
for Halstead metrics
        // but we might want to count their
arguments differently
        ops.push(head);
      } else {
        // Regular operators
        ops.push(head);
      }
    }

    // Special handling for matrix-like
structures
    if (head === 'Matrix' || head === 'List') {
      // For matrices and lists, count the
```

```
structure as an operator
      // and recursively process elements
      ops.push('Matrix');
      for (let i = 1; i < node.length; i++) {
        if (Array.isArray(node[i])) {
          // Matrix row - process each element
          (node[i] as unknown[]).
forEach((element: unknown) =>
            collectFromJson(element, ops,
oprnds)
          );
        } else {
          collectFromJson(node[i], ops, oprnds);
        }
      }
      return;
    }

    // Normal recursive processing
    for (let i = 1; i < node.length; i++)
collectFromJson(node[i], ops, oprnds);
    return;
  }

  // Numbers (operands)
  if (typeof node === 'number') {
    oprnds.push(node.toString());
    return;
  }

  // Symbols/identifiers are strings in MathJSON
  if (typeof node === 'string') {
    // Basic number check
    if (!isNaN(Number(node))) {
      oprnds.push(node);
      return;
    }

    // Explicitly check if it's a known operator
or structural name
    const knownOperators = new Set([
      'Add',
      'Subtract',
      'Multiply',
      'Divide',
      'Power',
      'Equal',
      'NotEqual',
      'Less',
      'Greater',
      'LessEqual',
      'GreaterEqual',
      'And',
      'Or',
      'Not',
      'Implies',
      'Equivalent',
      'Element',
      'In',
      'InElement',
      'NotIn',
      'Subset',
      'Union',
      'Intersection',
      'Tuple',
      'List',
      'Pair',
      'Sequence',
      'Function',
      'Sequence',
      'CircleAdd',
      'CircleMultiply',
      'PlusMinus',
      'MinusPlus',
      'MapsTo',
      'To',
      'LeftArrow',
      'Colon',
      'Approx',
      'Similar',
      'LatexString',
      'Error',
      'unexpected-command',
      'unexpected-operator',
      'Delimiter',
      'AngleBracket',
      'Parenthesis',
      'Bracket',
      'Brace',
      // Raw names that might leak from CE
      'oplus',
      'otimes',
      'neq',
      'land',
      'lor',
      'mapsto',
      'colon',
      'subset',
      'in',
      'ni',
      'cup',
      'cap',
      'subset',
      'in',
      'notin',
      'ni',
      'perp',
      'parallel',
      'cdot',
      'times',
      'sqrt',
      'frac',
      'partial',
      'nabla',
```

```
      'delta',
      'Delta',
    ]);

    if (knownOperators.has(node)) {
      ops.push(node);
      return;
    }

    // Constants treated as operands
    if (
      node === 'e' ||
      node === 'pi' ||
      node === 'ExponentialE' ||
      node === 'Infinity' ||
      node === 'NaN'
    ) {
      oprnds.push(node === 'ExponentialE' ?
'e' : node);
      return;
    }

    // Variables (including Greek letters which
are often returned as string names)
    if (/^[a-zA-Z][a-zA-Z0-9_]*$/.test(node)) {
      // If it's a Greek letter or variable-like
string, it's an operand
      oprnds.push(node);
    } else {
      // Anything else is treated as an
operator/head
      ops.push(node);
    }
  }

  // Objects can represent dictionaries (e.g.
, {"num": 1} or {"sym": "a"}) depending on CE
config.
  // Handle common cases defensively:
  if (typeof node === 'object' && node !== null)
{
    const n = node as Record<string, unknown>;
    if (typeof n.sym === 'string') {
      oprnds.push(n.sym);
      return;
    }
    if (typeof n.num === 'number') {
      oprnds.push(String(n.num));
      return;
    }
    // Handle matrix objects
    if (n.matrix) {
      ops.push('Matrix');
      collectFromJson(n.matrix, ops, oprnds);
      return;
    }
    // Generic walk over object values
    for (const v of Object.values(n))
collectFromJson(v, ops, oprnds);
  }
}

/** Compute Halstead metrics from occurrence
arrays */
function halsteadFrom(ops: string[], oprnds:
string[]): Halstead {
  const n1 = new Set(ops).size;
  const n2 = new Set(oprnds).size;
  const N1 = ops.length;
  const N2 = oprnds.length;
  const n = n1 + n2;
  const N = N1 + N2;
  const Volume = n > 0 && N > 0 ? N * Math.
log2(n) : 0;
  const Difficulty = n2 > 0 ? (n1 / 2) * (N2 /
n2) : 0;
  const Effort = Difficulty * Volume;

  const operators: Record<string, number> = {};
  const operands: Record<string, number> = {};
  for (const k of ops) operators[k] =
(operators[k] ?? 0) + 1;
  for (const k of oprnds) operands[k] =
(operands[k] ?? 0) + 1;

  return {
    n1,
    n2,
    N1,
    N2,
    n,
    N,
    Volume,
    Difficulty,
    Effort,
    operators,
    operands,
  };
}

/** Analyze a single LaTeX formula */
export function analyzeFormula(latex: string):
AnalysisResult {
  logger.debug(`Analyzing formula: ${latex}`);

  // Check for empty or whitespace-only formulas
  if (!latex || latex.trim().length === 0) {
    return {
      success: false,
      error: {
        type: 'extraction',
        message: 'Empty or whitespace-only
formula',
        formula: latex,
```

```
      },
    };
  }

  const parseResult = latexToJson(latex);
  if (!parseResult.success) {
    return parseResult;
  }

  try {
    const ops: string[] = [];
    const oprnds: string[] = [];
    collectFromJson(parseResult.json, ops,
oprnds);

    logger.debug(`Collected operators: ${ops.
length}, operands: ${oprnds.length}`);

    const halstead = halsteadFrom(ops, oprnds);
    logger.debug(`Calculated Halstead metrics`,
halstead);

    return { success: true, halstead };
  } catch (error) {
    logger.error(`Exception during Halstead
calculation: ${error}`, {
      latex,
      json: parseResult.json,
      error,
    });
    return {
      success: false,
      error: {
        type: 'calculation',
        message: `Calculation error: ${error
instanceof Error ? error.message : 'Unknown
error'}`,
        formula: latex,
        details: error,
      },
    };
  }
}

export type AnalysisRow = {
  formula: string;
} & AnalysisResult;

/** Analyze many formulas (aggregate correctly
by concatenating occurrences) */
export async function analyzeMany(
  formulas: string[],
  onProgress?: (current: number, total: number)
=> void
): Promise<{
  rows: AnalysisRow[];
  overall: Halstead;
  errors: AnalysisError[];
}> {
  logger.debug(`Analyzing ${formulas.length}
formulas`);

  const rows: AnalysisRow[] = [];
  const errors: AnalysisError[] = [];
  const successfulResults: Halstead[] = [];

  for (let i = 0; i < formulas.length; i++) {
    const formula = formulas[i];
    const result = analyzeFormula(formula);
    const row: AnalysisRow = { formula, ...
result };
    rows.push(row);

    if (onProgress) {
      onProgress(i + 1, formulas.length);
      // Allow UI to update
      await new Promise((resolve) =>
setTimeout(resolve, 0));
    }

    if (result.success) {
      successfulResults.push(result.halstead);
    } else {
      errors.push(result.error);
      logger.warn(`Failed to analyze formula:
${formula}`, result.error);
    }
  }

  // Aggregate only successful results
  const allOps: string[] = [];
  const allOperands: string[] = [];
  for (const halstead of successfulResults) {
    for (const [k, c] of Object.entries(halstead.
operators)) {
      for (let i = 0; i < c; i++) allOps.
push(k);
    }
    for (const [k, c] of Object.entries(halstead.
operands)) {
      for (let i = 0; i < c; i++) allOperands.
push(k);
    }
  }

  const overall = halsteadFrom(allOps,
allOperands);
  logger.debug(
    `Analysis complete: ${successfulResults.
length} successful, ${errors.length} failed`
  );

  return { rows, overall, errors };
}
```

2.12. Вміст файлу `lib/extraction.ts`

```
import { latexParser } from 'latex-utensils';

export interface ExtractionResult {
  formulas: string[];
  errors: string[];
}

/**
 * Extract mathematical formulas from LaTeX
source code
 */

export function extractMath(src: string):
ExtractionResult {
  const out: string[] = [];
  const extractionErrors: string[] = [];

  try {
    // 1. Attempt to parse with latex-utensils
for robust AST-based extraction
    try {
      const ast = latexParser.parse(src);

      const sliceByLoc = (node: unknown): string
| null => {
        const n = node as Record<string,
unknown>;
        const loc = n?.location ?? n?.loc;
        const l = loc as Record<string,
unknown>;
        const start = (l?.start as
Record<string, unknown>)?.offset;
        const end = (l?.end as Record<string,
unknown>)?.offset;
        if (typeof start !== 'number' || typeof
end !== 'number') return null;
        return src.slice(start, end);
      };

      const stripDelims = (s: string): string
=> {
        let t = s.trim();
        // inline math delimiters
        if (t.startsWith('$$') && t.endsWith('$
$')) return t.slice(2, -2).trim();
        if (t.startsWith('$') && t.
endsWith('$')) return t.slice(1, -1).trim();
        if (t.startsWith('\\(') && t.endsWith('\
\)')) return t.slice(2, -2).trim();
        if (t.startsWith('\\[') && t.endsWith('\
\]')) return t.slice(2, -2).trim();
        // display math delimiters
        if (t.startsWith('\\begin{displaymath}')
&& t.endsWith('\\end{displaymath}')) {
          return t.slice(20, -17).trim();
        }
        if (t.startsWith('\\begin{math}') && t.
endsWith('\\end{math}')) {
          return t.slice(12, -9).trim();
        }
        // generic environment extraction
(exclude align environments)
        // First trim trailing backslashes
to handle cases where they interfere with \end
matching
        const trimmedForMatching = t.replace(/\\
+$/, '');
        const envMatch
= trimmedForMatching.match(/^\\begin\{([^}]+)\}
([\s\S]*?)\\end\{\1}/);
        if (envMatch && !envMatch[1].
startsWith('align'))
          return envMatch[2].trim().replace(/\\+
$/, '');
        return t.trim().replace(/\\+$/, '');
      };

      // Formula validation function
      const isValidFormula = (formula: string):
boolean => {
        if (!formula || formula.trim().length
=== 0) return false;
        if (formula.length > 10000) return
false; // Too long, probably not a formula

        // Check for double newlines (paragraph
breaks)
        if (/[\r\n]+\s*[\r\n]+/.test(formula))
return false;

        // Clean the formula first
        const cleaned = cleanFormula(formula);

        // Check for basic LaTeX math patterns
        const hasMathContent = /[a-zA-Z0-9+\-*/
=^_{}[\]\\]/.test(cleaned);

        // Additional validations
        return (
          hasMathContent &&
          isBalanced(cleaned) &&
          hasBalancedDelimiters(cleaned) &&
          hasValidCommands(cleaned)
        );
      };

      const pushNode = (node: unknown) => {
        try {
          const chunk = sliceByLoc(node);
          if (!chunk) {
            extractionErrors.push(`Could
not extract text for math node: ${JSON.
```

```
stringify(node)}`);
            return;
          }
          let inner = stripDelims(chunk);
          inner = cleanFormula(inner);
          if (inner && isValidFormula(inner)
&& !inner.startsWith('\\begin{align')) {
            out.push(inner);
          } else if (inner && !inner.
startsWith('\\begin{align')) {
            extractionErrors.push(`Invalid or
empty formula extracted: "${inner}"`);
          }
        } catch (error) {
          extractionErrors.push(`Error
extracting math node: ${error}`);
        }
      };

      const walk = (node: unknown) => {
        if (!node || typeof node !== 'object')
return;
        const n = node as Record<string,
unknown>;
        const k: string = (n.kind as string) ??
'';

        // Comprehensive math environment
detection
        const isMathNode =
          // Standard latex-utensils math nodes
          k === 'inlineMath' ||
          k === 'displayMath' ||
          (typeof k === 'string' && k.
startsWith('env.math.')) ||
          // Additional math environments
          k === 'math' ||
          k === 'env.math' ||
          k === 'env.displaymath' ||
          k === 'env.equation*' ||
          k === 'env.gather' ||
          k === 'env.gather*' ||
          k === 'env.multline' ||
          k === 'env.multline*' ||
          k === 'env.flalign' ||
          k === 'env.flalign*';

        if (isMathNode) {
          pushNode(node);
        }

        for (const key in n) {
          const v = n[key];
          if (Array.isArray(v)) v.forEach(walk);
          else if (v && typeof v === 'object')
walk(v);
        }
      };

      walk(ast);
    } catch (_parseError) {
      // If AST parsing fails, we rely entirely
on fallback extraction
      // This is common for partial or malformed
LaTeX
      // We don't necessarily report this as a
user error if fallback works
    }

    // 2. Fallback extraction: regex-based
search for missed math
    // This runs regardless of AST success
to catch things AST might miss (e.g. inside
comments or non-standard structures)
    const fallbackExtract = (text: string):
string[] => {
      const fallbackFormulas: string[] = [];
      const patterns = [
        // Display math: \[ ... \]
        /\\\[([\s\S]*?)\\\]/g,
        // Inline math: $ ... $ (careful with
non-greedy matching and escaped dollars)
        /(?<!\\)\$(?!$)([^$]+?)(?<!\\)\$/g,
        // Parentheses: \( ... \)
        /\\\(([\s\S]*?)\\\)/g,
        // Double dollar: $$ ... $$
        /\$\$([\s\S]*?)\$\$/g,
        // Common math environments
        /\\begin\{(equation|gather|multline|
flalign)(?:\*)?\}([\s\S]*?)\\end\{\1(?:\*)?\}/g,
        // Display math environment
        /\\begin\{displaymath\}([\s\S]*?)\
\end\{displaymath\}/g,
        // Math environment
        /\\begin\{math\}([\s\S]*?)\\end\{math\}/
g,
      ];

      patterns.forEach((pattern) => {
        let match;
        while ((match = pattern.exec(text)) !==
null) {
          // Handle different regex capture
groups: some have content in match[2], others in
match[1]
          const formula = (match[2] !==
undefined ? match[2] : match[1]) || match[0];
          if (isValidFormula(formula)) {
            // Check if this formula is already
extracted
            const normalized = formula.replace(/
[ \t\n]+/g, ' ').trim();
            if (!out.some((existing) =>
existing.replace(/[ \t\n]+/g, ' ').trim() ===
```

```
normalized)) {
              fallbackFormulas.push(formula.
trim());
            }
          }
        }
      });

      return fallbackFormulas;
    };

    // Apply fallback extraction
    const fallbackFormulas =
fallbackExtract(src);
    if (fallbackFormulas.length > 0) {
      out.push(...fallbackFormulas);
      // Note: Fallback is successful, not an
error
    }

    // Deduplication: remove duplicate formulas
(normalize whitespace for comparison)
    // We remove ALL whitespace for
deduplication purposes to handle "x+y" vs "x +
y"
    const normalizedFormulas = out.map((s) => s.
replace(/\s+/g, ''));
    const uniqueFormulas = out.filter((formula,
index) => {
      const normalized = formula.replace(/\s+/
g, '');
      return normalizedFormulas.
indexOf(normalized) === index;
    });

    // Filter out formulas that are substrings
of other formulas (nested environment handling)
    const filteredFormulas = uniqueFormulas.
filter((formula, index) => {
      // Skip very short formulas that might be
parts of larger ones
      if (formula.length < 1) return false;

      // Check if this formula is contained
within any other formula
      return !uniqueFormulas.some((otherFormula,
otherIndex) => {
        if (index === otherIndex) return false;
        // Only filter if the other formula is
significantly longer and contains this one
        // And ensure it's not just a common
substring like "x" in "x+y"
        return otherFormula.includes(formula) &&
otherFormula.length > formula.length + 10;
      });
    });

    return { formulas: filteredFormulas, errors:
extractionErrors };
  } catch (error) {
    return {
      formulas: [],
      errors: [`LaTeX extraction failed:
${error}`],
    };
  }
}

// --- Helper Functions (Shared Logic) ---

// Check if braces are balanced
const isBalanced = (str: string): boolean => {
  let count = 0;
  for (let char of str) {
    if (char === '{') count++;
    if (char === '}') count--;
    if (count < 0) return false;
  }
  return count === 0;
};

// Check for known KaTeX-supported commands
const hasValidCommands = (formula: string):
boolean => {
  // Common unsupported commands that should be
rejected
  const invalidPatterns = [
    /\\begin\{(theorem|lemma|proof|corollary|
definition|example|exercise)\}/i,
    /\\cite\{/i,
    /\\ref\{/i,
    /\\label\{/i,
    /\\bibliography\{/i,
    /\\documentclass/i,
    /\\usepackage/i,
    /\\title\{/i,
    /\\author\{/i,
    /\\maketitle/i,
  ];

  return !invalidPatterns.some((pattern) =>
pattern.test(formula));
};

// Check if LaTeX delimiters are balanced
// This function ONLY validates \left...\right
pairs
// Standalone delimiters like \langle...\rangle,
{}, [], () are allowed
const hasBalancedDelimiters = (formula: string):
boolean => {
  const delimiterPairs: Record<string, string>
= {
    '(': ')',
```

```
    '[': ']',
    '{': '}',
    '<': '>',
    '|': '|',
    '/': '/',
    '.': '.',
    '\\{': '\\}',
    '\\langle': '\\rangle',
  };

  const stack: string[] = [];
  const regex = /\\left|\\right/g;
  let match;

  while ((match = regex.exec(formula)) !== null)
{
    const command = match[0];
    const afterCommand = formula.slice(match.
index + match[0].length);
    let delimiter = '';

    // Check for escaped delimiters first
    if (afterCommand.startsWith('\\{')) {
      delimiter = '\\{';
    } else if (afterCommand.startsWith('\
\langle')) {
      delimiter = '\\langle';
    } else if (afterCommand.startsWith('\\}')) {
      delimiter = '\\}';
    } else if (afterCommand.startsWith('\
\rangle')) {
      delimiter = '\\rangle';
    } else {
      delimiter = afterCommand[0];
    }

    if (command === '\\left') {
      if (delimiterPairs[delimiter]) {
        stack.push(delimiter);
      } else {
        // Invalid delimiter after \left
        return false;
      }
    } else if (command === '\\right') {
      if (stack.length === 0) {
        // \right without matching \left
        return false;
      }
      const expected = stack.pop()!;
      if (delimiter !==
delimiterPairs[expected]) {
        // Mismatched delimiter
        return false;
      }
    }
  }

  // Stack should be empty if all \left have
matching \right
  return stack.length === 0;
};

// Clean formula by removing trailing
punctuation
const cleanFormula = (formula: string): string
=> {
  let cleaned = formula.trim();

  // Remove trailing punctuation outside braces
  const braceStack: string[] = [];
  let lastValidIndex = cleaned.length;

  for (let i = 0; i < cleaned.length; i++) {
    if (cleaned[i] === '{') braceStack.
push('{');
    else if (cleaned[i] === '}') braceStack.
pop();

    // If we're at top level (no braces) and
find punctuation, mark as potential cutoff
    if (braceStack.length === 0 && /[,.;:]$/.
test(cleaned.slice(0, i + 1))) {
      lastValidIndex = i;
    }
  }

  // Only trim if we found punctuation at the
end
  if (lastValidIndex < cleaned.length &&
braceStack.length === 0) {
    cleaned = cleaned.slice(0, lastValidIndex).
trim();
  }

  return cleaned;
};

// Formula validation function
const isValidFormula = (formula: string):
boolean => {
  if (!formula || formula.trim().length === 0)
return false;
  if (formula.length > 10000) return false; //
Too long, probably not a formula

  // Clean the formula first
  const cleaned = cleanFormula(formula);

  // Check for basic LaTeX math patterns
  const hasMathContent = /[a-zA-Z0-9+\-*/=^_{}
[\]\\]/.test(cleaned);

  // Additional validations
  return (
```

```
    hasMathContent &&
    isBalanced(cleaned) &&
    hasBalancedDelimiters(cleaned) &&
    hasValidCommands(cleaned)
  );
};
```

2.13. Вміст файлу `lib/translations.ts`

```
export type Language = 'en' | 'ua';

export const translations: Record<Language,
Record<string, string>> = {
  en: {
    foundFormulas: 'Found {count} formulas',
    halsteadAnalysis: 'Halstead Analysis',
    successful: 'successful',
    extractionErrors: 'Extraction Errors',
    analysisErrors: 'Analysis Errors',
    rawLatex: 'Raw LaTeX',
    metrics: 'Metrics',
    uniqueOperators: 'Unique Operators',
    uniqueOperands: 'Unique Operands',
    totalOperators: 'Total Operators',
    totalOperands: 'Total Operands',
    volume: 'Volume',
    difficulty: 'Difficulty',
    effort: 'Effort',
    date: 'Date',
    sortName: 'Name',
    asc: 'Ascending',
    desc: 'Descending',
    sortBy: 'Sort by',
    operators: 'Operators',
    operands: 'Operands',
    overallMetrics: 'Overall Metrics',
    failed: 'FAILED',
    tooltipN1: 'n1: Number of unique operators
in the formula',
    tooltipN2: 'n2: Number of unique operands in
the formula',
    tooltipN1Total: 'N1: Total occurrences of
operators',
    tooltipN2Total: 'N2: Total occurrences of
operands',
    tooltipVolume: 'V: Program volume (size
complexity)',
    tooltipDifficulty: 'D: Program difficulty
(error proneness)',
    tooltipEffort: 'E: Effort required to
implement/maintain',
    openTexFile: 'Open .tex…',
    openFile: 'Open File…',
    debugMode: 'Debug Mode',
    file: 'File',
    language: 'Language',
    includeInComparison: 'Include in
comparison',
    metricComparison: 'Metric Comparison',
    selectMetrics: 'Select Metrics',
    formula: 'Formula',
    addFormula: 'Add Formula',
    saveFormulas: 'Save Formulas',
    formulaName: 'Formula Name',
    editFormula: 'Edit Formula',
    enterLatex: 'Enter LaTeX formula',
    formulaSaved: 'Formulas saved successfully',
    formulaLoaded: 'Formulas loaded
successfully',
    processing: 'Processing: {file}',
    completed: 'Completed: {file} ({current}/
{total})',
    errorProcessing: 'Error processing {file}',
    analyzingFormulas: 'Analyzing formulas:
{current}/{total}',
    analyzing: 'Analyzing formulas...',
    analysisFailed: 'Analysis failed',
    cancelled: 'Cancelled',
    startingExtraction: 'Starting extraction..
.',
    fileLoaded: 'File loaded: {file} ({count}
formulas found)',
    errorLoading: 'Error loading file.',
    processedMultiple: 'Processed {count} files,
showing results for the first one.',
    failedToProcess: 'Failed to process files:
{error}',
    notYetAnalyzed: 'Not yet analyzed',
    deleteFormula: 'Delete formula',
    confirmDelete: 'Confirm deletion',
    cancelDelete: 'Cancel deletion',
    editName: 'Edit name',
    totalFormulas: 'Total formulas: {count}',
    debugInfo: 'Debug Information',
    rawLatexContent: 'Raw LaTeX Content',
    analysisResultsRaw: 'Analysis Results
(Raw)',
    clickToViewRaw: 'Click to view raw results',
    cancel: 'Cancel',
    percentComplete: '{percent}% complete',
  },
  ua: {
    foundFormulas: 'Знайдено {count} формул',
    halsteadAnalysis: 'Аналіз Холстеда',
    successful: 'успішних',
    extractionErrors: 'Помилки вилучення',
    analysisErrors: 'Помилки аналізу',
    rawLatex: 'Оригінальний LaTeX',
    metrics: 'Метрики',
    uniqueOperators: 'Унікальні оператори',
    uniqueOperands: 'Унікальні операнди',
    totalOperators: 'Загальна кількість
операторів',
    totalOperands: 'Загальна кількість
```

```
операндів',
    volume: "Об'єм",
    difficulty: 'Складність',
    effort: 'Зусилля',
    date: 'Дата',
    sortName: 'Назва',
    asc: 'За зростанням',
    desc: 'За спаданням',
    sortBy: 'Сортувати за',
    operators: 'Оператори',
    operands: 'Операнди',
    overallMetrics: 'Загальні метрики',
    failed: 'НЕВДАЧА',
    tooltipN1: 'n1: Кількість унікальних
операторів у формулі',
    tooltipN2: 'n2: Кількість унікальних
операндів у формулі',
    tooltipN1Total: 'N1: Загальна кількість появ
операторів',
    tooltipN2Total: 'N2: Загальна кількість появ
операндів',
    tooltipVolume: "V: Об'єм програми (розмір
складності)",
    tooltipDifficulty: 'D: Складність програми
(схильність до помилок)',
    tooltipEffort: 'E: Зусилля, необхідні для
реалізації/підтримки',
    openTexFile: 'Відкрити .tex…',
    openFile: 'Відкрити файл…',
    debugMode: 'Режим налагодження',
    file: 'Файл',
    language: 'Мова',
    includeInComparison: 'Включити в
порівняння',
    metricComparison: 'Порівняння метрик',
    selectMetrics: 'Виберіть метрики',
    formula: 'Формула',
    addFormula: 'Додати формулу',
    saveFormulas: 'Зберегти формули',
    formulaName: 'Назва формули',
    editFormula: 'Редагувати формулу',
    enterLatex: 'Введіть формулу LaTeX',
    formulaSaved: 'Формули успішно збережено',
    formulaLoaded: 'Формули успішно
завантажено',
    processing: 'Обробка: {file}',
    completed: 'Завершено: {file} ({current}/
{total})',
    errorProcessing: 'Помилка обробки {file}',
    analyzingFormulas: 'Аналіз формул:
{current}/{total}',
    analyzing: 'Аналіз формул...',
    analysisFailed: 'Аналіз не вдався',
    cancelled: 'Скасовано',
    startingExtraction: 'Початок вилучення...',
    fileLoaded: 'Файл завантажено: {file}
(знайдено {count} формул)',
    errorLoading: 'Помилка завантаження файлу.',
    processedMultiple: 'Оброблено {count}
файлів, показано результати для першого.',
    failedToProcess: 'Не вдалося обробити файли:
{error}',
    notYetAnalyzed: 'Ще не проаналізовано',
    deleteFormula: 'Видалити формулу',
    confirmDelete: 'Підтвердити видалення',
    cancelDelete: 'Скасувати видалення',
    editName: 'Редагувати назву',
    totalFormulas: 'Всього формул: {count}',
    debugInfo: 'Інформація для налагодження',
    rawLatexContent: 'Сирий вміст LaTeX',
    analysisResultsRaw: 'Результати аналізу
(сирі)',
    clickToViewRaw: 'Натисніть, щоб переглянути
сирі результати',
    cancel: 'Скасувати',
    percentComplete: '{percent}% завершено',
  },
};

export function t(lang: Language, key: string,
params?: Record<string, string | number>):
string {
  let text = translations[lang][key] ||
translations.en[key] || key;
  if (params) {
    Object.entries(params).forEach(([k, v]) => {
      text = text.replace(`{${k}}`, String(v));
    });
  }
  return text;
}
```

2.14. Вміст файлу `lib/types.ts`

```
export interface ProgressData {
  file: string;
  status: 'starting' | 'completed' | 'error';
  result?: unknown;
  error?: string;
}

export interface AnalysisProgressData {
  status: 'analyzing' | 'completed' | 'error';
  current?: number;
  total?: number;
}

export interface AnalysisRow {
  formula: string;
  halstead: {
    operators: Record<string, number>;
    operands: Record<string, number>;
    n1: number;
    n2: number;
```

```
    N1: number;
    N2: number;
    n: number;
    N: number;
    Volume: number;
    Difficulty: number;
    Effort: number;
  };
  success: boolean;
  error?: { message: string };
}

export interface AnalysisResults {
  rows: AnalysisRow[];
  errors: Array<{ formula: string; message:
string }>;
  overall: {
    n1: number;
    n2: number;
    N1: number;
    N2: number;
    Volume: number;
    Difficulty: number;
    Effort: number;
  };
}

export interface FormulaMetadata {
  id: string;
  name: string;
  latex: string;
  created: string;
  modified: string;
  analysis?: AnalysisRow['halstead'];
  tags?: string[];
}

export interface FormulaFile {
  version: string;
  metadata: {
    created: string;
    modified: string;
    sourceFile?: string;
    description?: string;
  };
  formulas: FormulaMetadata[];
}

export interface ElectronAPI {
  openFile: () => Promise<string[]>;
  readFile: (filePath: string) =>
Promise<string>;
  readFileBuffer: (filePath: string) =>
Promise<Uint8Array>;
  extractWordFormulas: (filePath: string) =>
Promise<{ formulas: string[]; errors: string[] }
>;
  extractFormulas: (
    files: string[]
  ) => Promise<Array<{ file: string; formulas:
string[]; errors: string[] }>>;
  analyzeFormulas: (formulas: string[]) =>
Promise<AnalysisResults>;
  cancelExtraction: () => Promise<void>;
  saveFormulas: (data: FormulaFile) =>
Promise<string | null>;
  readFmxFile: (filePath: string) =>
Promise<FormulaFile>;
  onProgress: (callback: (event: unknown, data:
ProgressData) => void) => void;
  offProgress: (callback: (event: unknown, data:
ProgressData) => void) => void;
  onAnalysisProgress: (callback: (event:
unknown, data: AnalysisProgressData) => void) =>
void;
  offAnalysisProgress: (callback: (event:
unknown, data: AnalysisProgressData) => void) =>
void;

  // Menu events
  onMenuOpen: (callback: () => void) => void;
  offMenuOpen: (callback: () => void) => void;
  onMenuSave: (callback: () => void) => void;
  offMenuSave: (callback: () => void) => void;
  onMenuDebug: (callback: (event: unknown,
enabled: boolean) => void) => void;
  offMenuDebug: (callback: (event: unknown,
enabled: boolean) => void) => void;
  onMenuLanguageEn: (callback: () => void) =>
void;
  offMenuLanguageEn: (callback: () => void) =>
void;
  onMenuLanguageUa: (callback: () => void) =>
void;
  offMenuLanguageUa: (callback: () => void) =>
void;
}

declare global {
  interface Window {
    api: ElectronAPI;
  }
}
```

ДОДАТОК В

Керівництво користувача

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного університету
науки і технологій
_____ /Анатолій РАДКЕВИЧ/
Дата _____

ПРОГРАМА ДЛЯ ДОСЛІДЖЕННЯ СКЛАДНОСТІ КОНСТРУКТИВНО-ПРОДУКЦІЙНИХ МОДЕЛЕЙ ЗОН РЕКУПЕРАЦІЇ ПОСТІЙНОГО СТРУМУ

Керівництво користувача.
Керівництво з дослідження складності конструктивно-продукційних моделей

ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.1535-01-І3-01

Завідувач кафедри КІТ
Вадим ГОРЯЧКІН

Керівник розробки
проф. Віктор ШИНКАРЕНКО

Виконавець
Євген ШАПОВАЛ

Нормоконтролер
доц. Світлана ВОЛКОВА

2026

ЗАТВЕРДЖЕНО
44165850.1535-01-І3-01

ПРОГРАМА ДЛЯ ДОСЛІДЖЕННЯ СКЛАДНОСТІ КОНСТРУКТИВНО-ПРОДУКЦІЙНИХ МОДЕЛЕЙ ЗОН РЕКУПЕРАЦІЇ ПОСТІЙНОГО СТРУМУ

Керівництво користувача

Керівництво з дослідження складності конструктивно-продукційних моделей

44165850.1535-01-І3-01

листів 11

2026

ЗМІСТ

ВСТУП

Програма electron-svelte-latex призначена для автоматизованого аналізу складності формалізованих описів конструктивно-продукційних моделей зон рекуперації постійного струму за метриками Холстеда. Вона дозволяє завантажувати документи у форматах .tex та .docx, вилучати з них формули, виконувати перевірку й нормалізацію LaTeX-запису, формувати семантичне подання MathJSON, підраховувати оператори та операнди, обчислювати показники V, D, E та візуалізувати результати для порівняння формул між собою. Програма розроблена на базі Electron з використанням Svelte та TypeScript, працює на Windows 10+ та сучасних дистрибутивах Linux, і для коректної роботи рекомендується процесор рівня Intel Core i5 або аналогічний, 4 ГБ оперативної пам’яті та не менше 200 МБ вільного місця на диску (без урахування розміру вхідних документів).

ПРИЗНАЧЕННЯ І УМОВИ ВИКОРИСТАННЯ

Призначення: програмний засіб `electron-svelte-latex` призначений для автоматизованого аналізу складності формульних описів конструктивно-продукційних моделей зон рекуперації постійного струму з обчисленням метрик Холстеда (n_1, n_2, N_1, N_2, V, D, E) та формуванням візуальних матеріалів для порівняння і ранжування формул у межах вибірки.

Умови використання: ОС – Windows 10/11 (64-біт) або сучасні дистрибутиви Linux; вимоги – встановлений Node.js (LTS) для запуску з вихідних кодів або зібраний дистрибутив застосунку, не менше 200 МБ вільного місця на диску (без урахування розміру документів), 4 ГБ оперативної пам’яті та процесор рівня Intel Core i5 або аналогічний; дані – файли `.tex` з LaTeX-розміткою або `.docx` з формулами, що можуть бути конвертовані у LaTeX для подальшого аналізу; режим – офлайн, без обов’язкового доступу до інтернету; застосовувати для наукових, навчальних або прикладних цілей під час дослідження та порівняння формалізованих описів моделей. Заборонено комерційне поширення без дозволу автора.

ПІДГОТОВКА ДО РОБОТИ

Для роботи з програмним засобом `electron-svelte-latex` необхідно встановити застосунок та підготувати вхідні документи з формулами. Після встановлення виконайте такі дії:

1. Запустіть застосунок `electron-svelte-latex`.
2. Підготуйте вхідні матеріали для аналізу:
   - для LaTeX – файл `.tex`, у якому формули записані у математичних режимах (inline або display) та, за потреби, у середовищах `equation`, `align` тощо;
   - для Word – файл `.docx`, що містить формули, набрані стандартними засобами редактора (вбудовані математичні об’єкти).

ПОРЯДОК РОБОТИ З ПРОГРАМОЮ

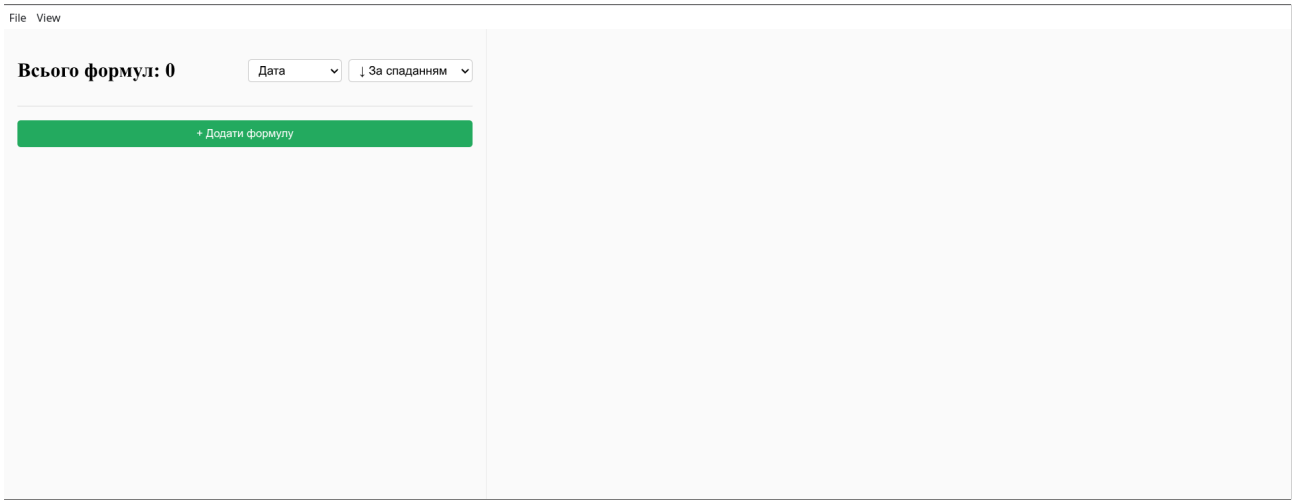

Рисунок 4.1 – Початковий вигляд застосунку після запуску.

На рис. 4.1 представлено початковий стан програми. Переконайтеся, що вікно відкрилося, а робоча область порожня. Для початку роботи відкрийте файл з формулами через меню.

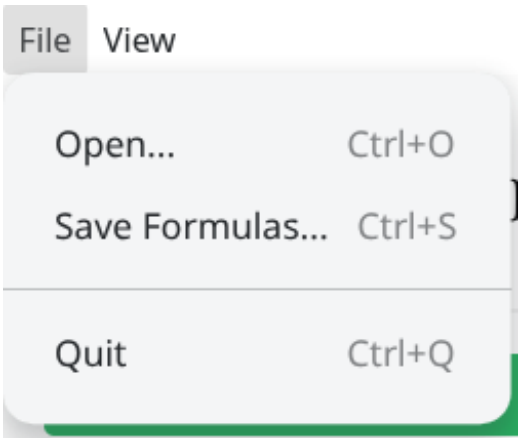

Рисунок 4.2 – Меню File для відкриття та збереження.

Відкрийте меню `File`, що зображено на рис. 4.2. Натисніть `Open...`, щоб вибрати документ `.tex`, `.docx` або збережену сесію `.fmx`. Для збереження поточного набору формул і результатів натисніть `Save Formulas...`.

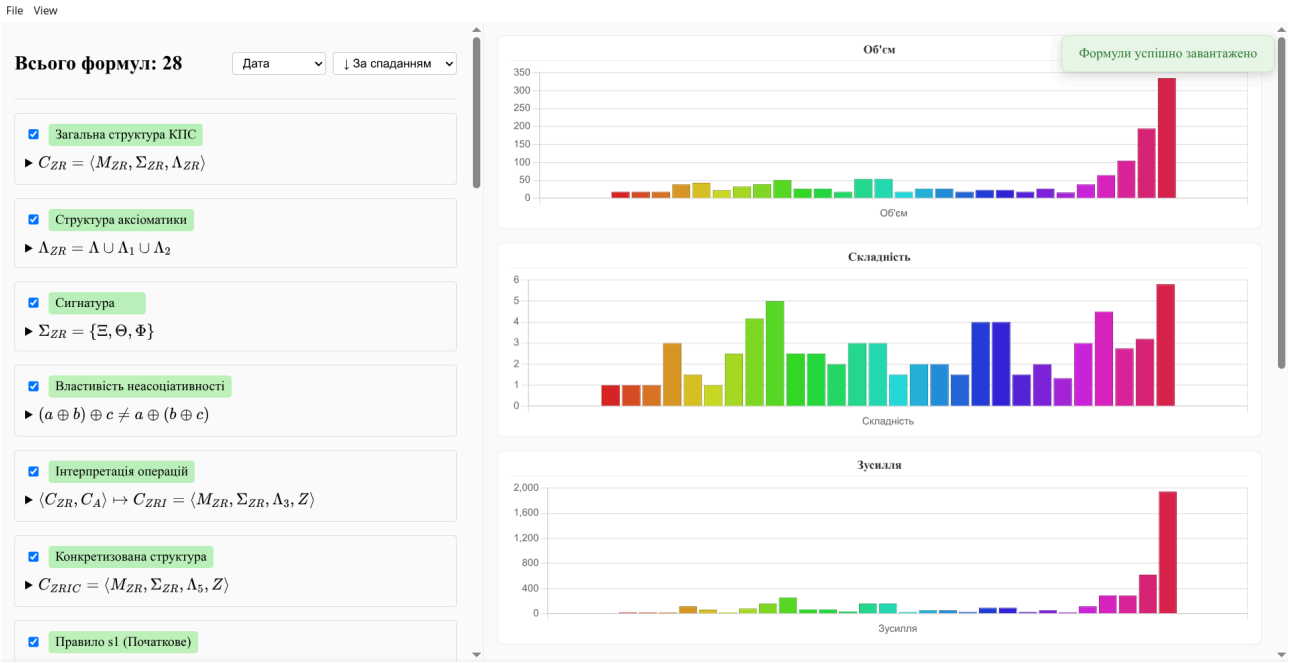

Рисунок 4.3 – Робоча область після завантаження формул.

Перевірте, що формули завантажилися: у лівій панелі має з’явитися список карток, а у правій – діаграми порівняння, що зображено на рис. 4.3. Зверніть увагу на верхній правий кут, де буде повідомлення про успішність завантаження формул з файлу.

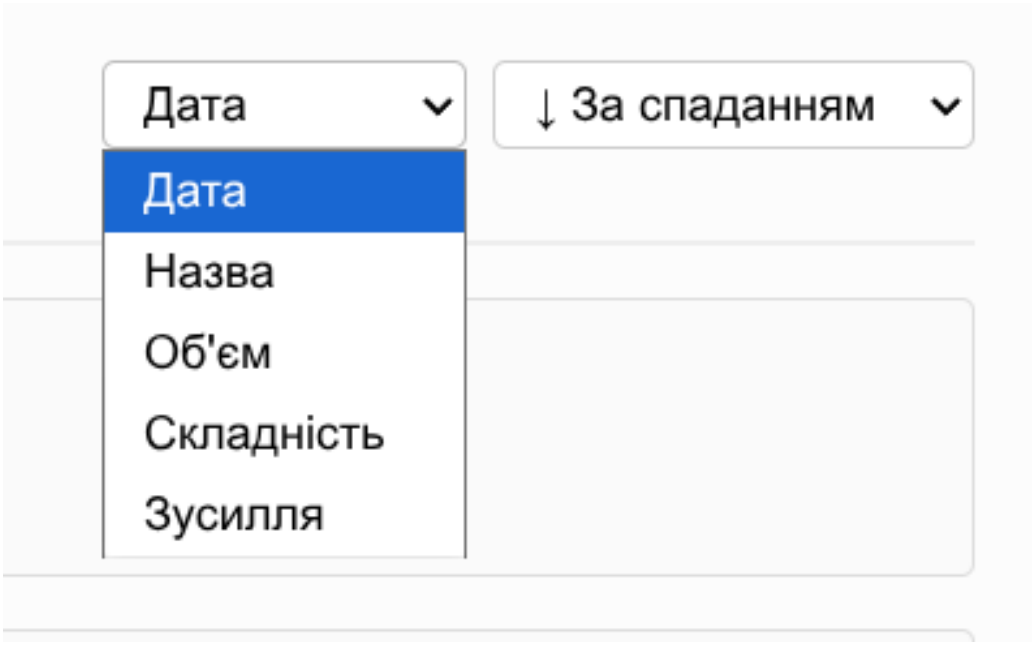

Рисунок 4.4 – Вибір критерію сортування списку формул.

Відкрийте лівий випадаючий список і виберіть критерій сортування (рис. 4.4). Встановіть `Дата`, щоб впорядкувати формули за часом створення, або виберіть `Назва`, `Об’єм`, `Складність`, `Зусилля`, щоб швидко знайти найважливіші формули за обраним показником.

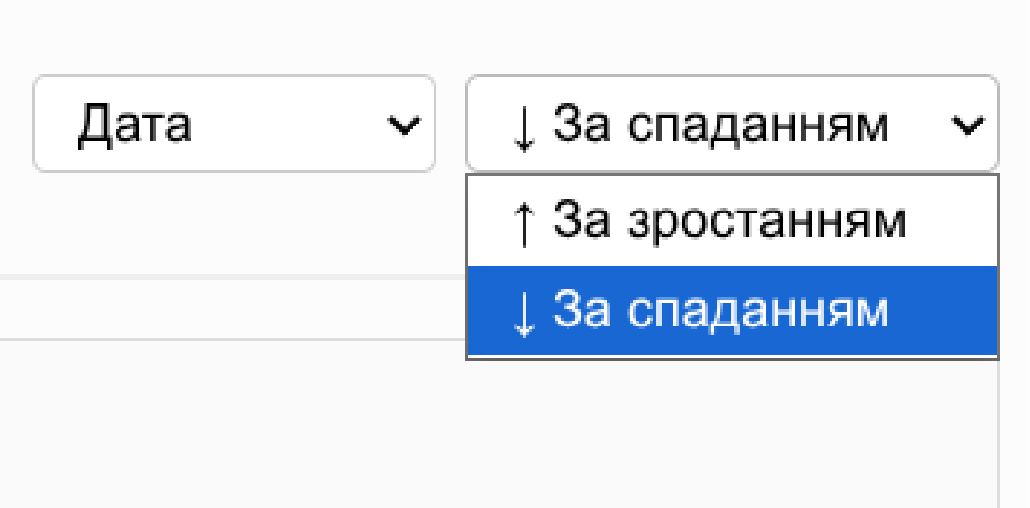

Рисунок 4.5 – Вибір напрямку сортування.

Відкрийте правий випадаючий список і виберіть напрям (рис. 4.5). Натисніть `За зростанням`, щоб показати найменші значення першими, або `За спаданням`, щоб одразу бачити формули з найбільшими значеннями метрик.

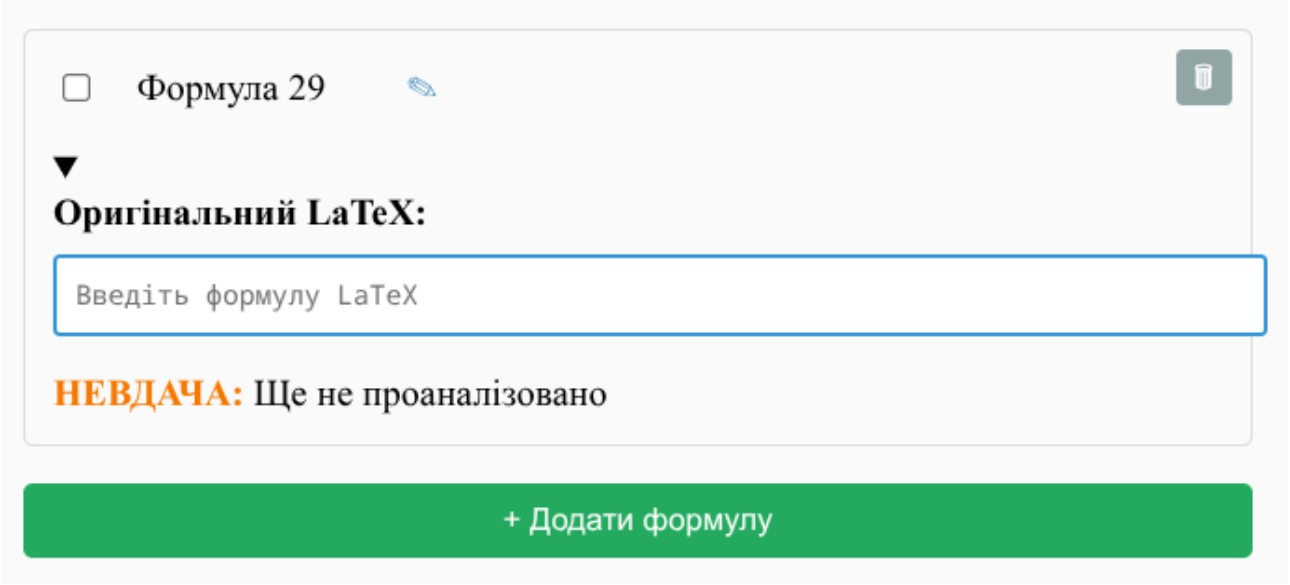

Рисунок 4.6 – Розгортання картки формули для редагування та перегляду результатів.

Розгорніть картку формули: натисніть на стрілку зліва від назви, щоб відкрити деталі. Введіть або виправте LaTeX у полі “Оригінальний LaTeX”, після чого дочекайтеся оновлення результатів. Перегляньте метрики для вибраної формули та за потреби видаліть її, натиснувши кнопку з іконкою кошика.

Щоб додати нову формулу вручну, натисніть кнопку `+ Додати формулу` у нижній частині списку. Щоб зберегти результат роботи, відкрийте меню `File` і натисніть `Save Formulas...`, після чого вкажіть місце збереження файлу `.fmx`.

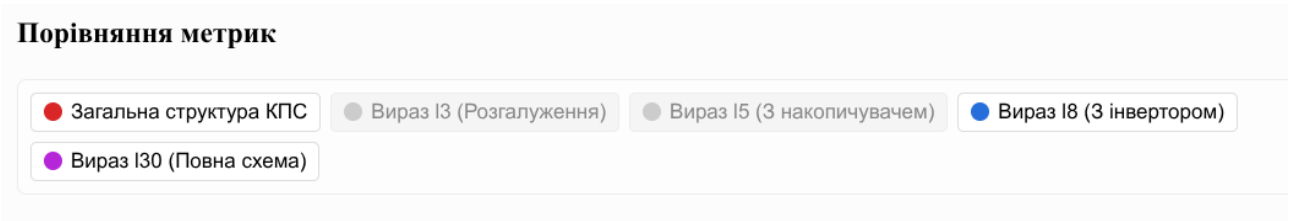

Рисунок 4.7 – Легенда діаграм для вмикання та вимикання окремих формул.

Керуйте порівнянням формул на діаграмах. Натисніть на назву формули в легенді, щоб приховати її на всіх діаграмах. Натисніть повторно, щоб повернути відображення. Використовуйте це, коли потрібно порівнювати лише кілька ключових формул без візуального перевантаження.

РЕКОМЕНДАЦІЇ

Під час роботи доцільно одразу визначити правила підготовки формул і однаковий підхід до інтерпретації результатів, оскільки метрики Холстеда найкраще працюють саме як засіб порівняння між формулами в межах однієї вибірки. Аналізуйте формули у вже нормалізованому вигляді: приберіть випадкові текстові вставки в математичному режимі, уникайте нестандартних макросів, стежте за балансом дужок та однаково оформлюйте індекси, дроби й вирівнювання. Якщо формула складна, почніть з базового запису і лише потім додавайте деталізацію, що допоможе відслідковувати, які саме зміни найбільше впливають на V, D та E.

Для документів `.docx` важливо підтримувати однорідний стиль набору формул у Word: використання стандартних математичних об’єктів зменшує ризик помилок під час перетворення у LaTeX та подальшого аналізу. Якщо в документі багато різнотипних формул, зручніше виконувати аналіз частинами та зберігати проміжні сесії у `.fmx`, щоб мати змогу швидко повертатися до попередніх результатів без повторного імпорту.

АВАРІЙНІ СИТУАЦІЇ

Помилка завантаження документа: якщо файл не відкривається або не імпортується, слід перевірити, що обрано підтримуваний формат `.tex` або `.docx`, файл не пошкоджений, а також що є права доступу до читання цього файлу. Для `.docx` додатково варто переконатися, що формули створені стандартними засобами Word, оскільки нестандартні вбудовані об’єкти можуть бути некоректно оброблені.

Помилка вилучення формул: якщо після імпорту програма не знаходить формули або знаходить їх неповністю, слід перевірити, чи присутні математичні режими у `.tex` (inline або display) та чи немає значних фрагментів, оформлених як звичайний текст. Для `.docx` типова причина – складні об’єкти або змішування формул із текстом у нестандартний спосіб; у такому разі доцільно перевірити документ на іншому ПК або спростити форматування формул у вихідному файлі.

Помилка валідації або рендерингу LaTeX: якщо певна формула не проходить перевірку або не відображається у попередньому перегляді, зазвичай причиною є незбалансовані дужки, некоректні команди, а також фрагменти, які залежать від користувацьких макросів. Рекомендовано відкрити формулу в режимі редагування, спростити запис (прибрати нестандартні команди, перевірити дужки, індекси та дроби) і повторити аналіз.

Неповний або некоректний розрахунок метрик: якщо метрики для формули не обчислюються або виглядають явно некоректно, це може означати, що семантичний розбір не зміг побудувати коректне дерево MathJSON. У такому випадку слід перевірити LaTeX-представлення формули, а також звернути увагу на складні конструкції на кшталт матриць, вирівнювання `align`, великої кількості вкладених груп або символів-розділювачів `&` і `\\`.

Зависання інтерфейсу або підвищене споживання пам’яті: при обробці дуже великих документів або великої кількості формул можливе зростання часу аналізу та використання оперативної пам’яті. Рекомендовано виконувати аналіз частинами (розбивати документ на логічні блоки), закривати сторонні застосунки, а також зберігати сесію `.fmx` перед повторними запусками аналізу.

ДОДАТОК Г

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ НАУКИ І ТЕХНОЛОГІЙ

ABSTRACTS
OF THE XIX INTERNATIONAL CONFERENCE
«MODERN INFORMATION AND COMMUNICATION TECHNOLOGIES ON A TRANSPORT, IN INDUSTRY AND EDUCATION»
18-19, December, 2025

СУЧАСНІ ІНФОРМАЦІЙНІ ТА КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ НА ТРАНСПОРТІ, В ПРОМИСЛОВОСТІ І ОСВІТІ

ПРИСВЯЧЕНО ПАМ'ЯТІ ПРОФЕСОРА ІГОРЯ ЖУКОВИЦЬКОГО

ТЕЗИ

XIX МІЖНАРОДНОЇ НАУКОВО-ПРАКТИЧНОЇ КОНФЕРЕНЦІЇ
18-19 ГРУДНЯ 2025

ДНІПРО
2025

Використання показників структурної складності конструктивно-продукційної моделі зони рекуперації тяги потягів постійного струму

Шаповал Є. В., Шинкаренко В. І., Український державний університет науки і технологій, Україна

Проблема підвищення енергоефективності на електрифікованому транспорті вимагає постійного вдосконалення як технічних засобів, так і алгоритмів керування зонами рекуперації. У цьому контексті вибір найбільш раціональної структурної конфігурації ділянки – це не лише питання вартості обладнання, але й питання її подальшої керованості та надійності.

В умовах структурної варіативності тягових мереж, де можуть застосовуватися різні комбінації пасивних та активних елементів, виникає необхідність у формалізованому, об’єктивному критерії оцінки складності. Такий критерій повинен дозволяти інженеру порівнювати альтернативні схеми не за їхньою графічною наочністю, а за їхньою внутрішньою логічною "вагою".

Для кількісного вимірювання структурної складності моделей зон рекуперації застосовується апарат конструктивно-продукційного моделювання (КПМ) у поєднанні з метриками Холстеда. КПМ забезпечує переведення інженерної схеми в математичну модель, яку можна аналізувати через призму операторів та операндів. Саме метрики Холстеда, зокрема, інтегральний показник «Зусилля» (E), надають чисельну оцінку складності розуміння та реалізації керуючих алгоритмів для даної конфігурації.

Складність D визначається як $D = \frac{2n_1}{n_2} \times \frac{N_2}{n_2}$, де n_1 і n_2 — кількість унікальних операторів та операндів відповідно.

Аналіз реальних та потенційних структурних схем виявляє важливу закономірність: структурна складність не зростає лінійно зі збільшенням кількості фізичних елементів. Було встановлено, що конфігурації, які включають активне обладнання (наприклад, системи накопичення енергії або інвертори), демонструють різке, стрибкоподібне збільшення показника E.

Цей ефект пояснюється тим, що такі елементи вводять до моделі нові, складніші логічні оператори керування, які вимагають набагато більшої кількості взаємодіючих операндів для підтримки динамічного енергетичного балансу. Фактично, високе значення E сигналізує не просто про велику схему, а про високу ймовірність виникнення складних, неочевидних взаємозв'язків та проблем з керованістю.

Виходячи з цього, показники структурної складності можуть виступати як вирішальний фактор при виборі раціональної конфігурації. При проектуванні нової або модернізації існуючої зони рекуперації інженер може використовувати наступний критерій: з кількох технічно можливих конфігурацій, які забезпечують однакову або близьку енергетичну ефективність (за критерієм E_{rec}), пріоритет слід надавати тій, яка має мінімальне значення метрики «Зусилля» (E).

Такий підхід дозволяє свідомо обирати менш "ризиковані" з точки зору складності управління схеми. Таким чином, структурна складність стає не просто аналітичним показником, а інженерним інструментом підтримки прийняття рішень, що забезпечує не лише економічну, але й експлуатаційну раціональність обраної конфігурації.