ДОДАТОК А

Функції попередньої обробки тексту для BERT моделі

```
def    prepare_dialogue(dialogue_turns,    tokenizer,    max_turns:    int,
max_seq_length: int):
    input_ids_list = []
    attention_mask_list = []
    for turn in dialogue_turns[:max_turns]:
        encoding = tokenizer.encode_plus(
            turn,
            add_special_tokens=True,
            max_length=max_seq_length,
            truncation=True,
            padding="max_length",
            return_attention_mask=True,
            return_tensors="pt"
        )
        input_ids_list.append(encoding["input_ids"])
        attention_mask_list.append(encoding["attention_mask"])
    # Pad with empty turns if fewer than max_turns
    while len(input_ids_list) < max_turns:
        encoding = tokenizer.encode_plus("",
            add_special_tokens=True,
            max_length=max_seq_length,
            truncation=True,
            padding="max_length",
            return_attention_mask=True,
            return_tensors="pt"
        )
        input_ids_list.append(encoding["input_ids"])
        attention_mask_list.append(encoding["attention_mask"])
    dialogue_input_ids = torch.stack(input_ids_list).squeeze(1)
    dialogue_attention_masks = torch.stack(attention_mask_list).squeeze(1)
    return dialogue_input_ids, dialogue_attention_masks
def load_all_dialogues(json_dir: str):
    all_dialogues = []
    for filename in os.listdir(json_dir):
        if filename.endswith(".json"):
            file_path = os.path.join(json_dir, filename)
            with open(file_path, "r") as f:
                data = json.load(f)
                if isinstance(data, list):
                    all_dialogues.extend(data)
                else:
                    all_dialogues.append(data)
    return all_dialogues
def load_multitask_data(json_dir: str, max_turns: int):
    all_dialogs = load_all_dialogues(json_dir)
    dialogues = []
    dialogue_labels_raw = []
    utterance_labels_all = []
    service_set = set()
    service_set.add("None")
        for dialogue in all_dialogs:
        services = dialogue.get("services", [])
        if not services:
```

```
            services = ["None"]
        dialogue_labels_raw.append(services)
        service_set.update(services)
        dialogue_turns = []
        utterance_labels = []
        for turn in dialogue.get("turns", []):
            speaker = turn.get("speaker", "")
            utterance = turn.get("utterance", "")
            dialogue_turns.append(f"{speaker}: {utterance}")
            turn_label = {}
            for frame in turn.get("frames", []):
                service = frame.get("service", "")
                active_intent = frame.get("state", {}).get("active_intent",
"NONE")
                if active_intent != "NONE":
                    turn_label[service] = 1
                    service_set.add(service)
            if not turn_label:
                turn_label = {"None": 1}
            utterance_labels.append(turn_label)
        dialogues.append(dialogue_turns)
        utterance_labels_all.append(utterance_labels)
    service2index    =    {service:    idx    for    idx,    service    in
enumerate(sorted(service_set))}
    num_labels = len(service2index)
    dialogue_labels = []
    for services in dialogue_labels_raw:
        vector = [0] * num_labels
        for s in services:
            if s in service2index:
                vector[service2index[s]] = 1
        dialogue_labels.append(vector)
    utterance_labels_final = []
    for turn_labels in utterance_labels_all:
        utt_labels = []
        for t in turn_labels[:max_turns]:
            vector = [0] * num_labels
            for s, flag in t.items():
                if s in service2index and flag:
                    vector[service2index[s]] = 1
            utt_labels.append(vector)
        while len(utt_labels) < max_turns:
            utt_labels.append([0] * num_labels)
        utterance_labels_final.append(utt_labels)
    return     dialogues,     dialogue_labels,     utterance_labels_final,
service2index
```

ДОДАТОК Б

Функції попередньої обробки тексту для HMM моделі

```
def load_all_dialogues(train_folder="train"):
    all_dialogues = []
    json_files = glob.glob(os.path.join(train_folder, "*.json"))
    print(f"Found {len(json_files)} JSON files in '{train_folder}'.")
    for idx, json_file in enumerate(json_files):
        with open(json_file, "r") as f:
            data = json.load(f)
            all_dialogues.extend(data)
        print(f"Loaded file {idx+1}/{len(json_files)}:
{os.path.basename(json_file)}")
    return all_dialogues
def extract_service_label(turn):
    for frame in turn.get("frames", []):
        state = frame.get("state", {})
        active_intent = state.get("active_intent", None)
        service = frame.get("service", "").lower()
        if active_intent and active_intent.upper() != "NONE" and service in
VALID_SERVICES:
            return service
    return "None"
def extract_utterances_and_services(dialogues):
    dialogue_utterances = []
    dialogue_service_labels = []
    for dialogue in dialogues:
        curr_utts = []
        curr_labels = []
        last_valid = None
        services = dialogue.get("services", [])
        if services:
            initial = services[0].lower()
            if initial in VALID_SERVICES:
                last_valid = initial
        for turn in dialogue.get("turns", []):
            utt = turn.get("utterance", "")
            label = extract_service_label(turn)
            if label.lower() == "none" and last_valid is not None:
                label = last_valid
            elif label.lower() != "none":
                last_valid = label
            curr_utts.append(utt)
            curr_labels.append(label)
        dialogue_utterances.append(curr_utts)
        dialogue_service_labels.append(curr_labels)
    return dialogue_utterances, dialogue_service_labels
```

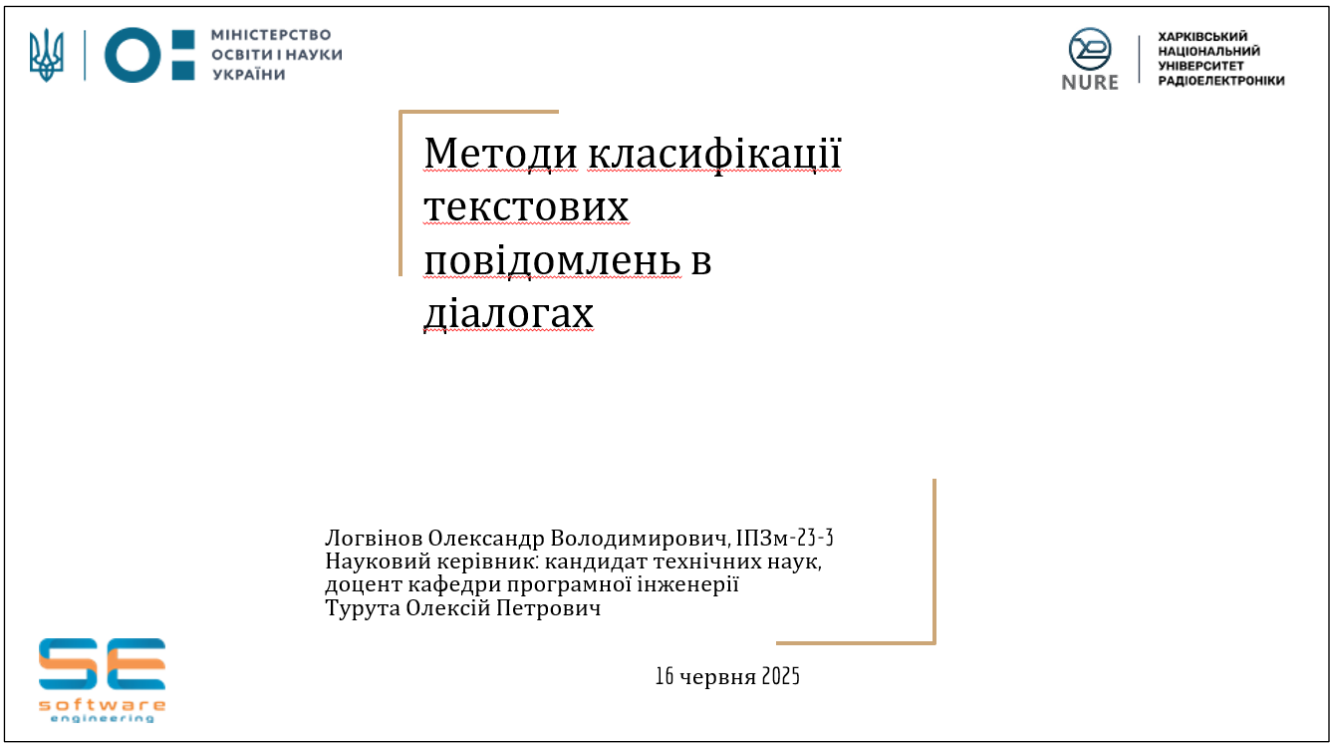
ДОДАТОК В

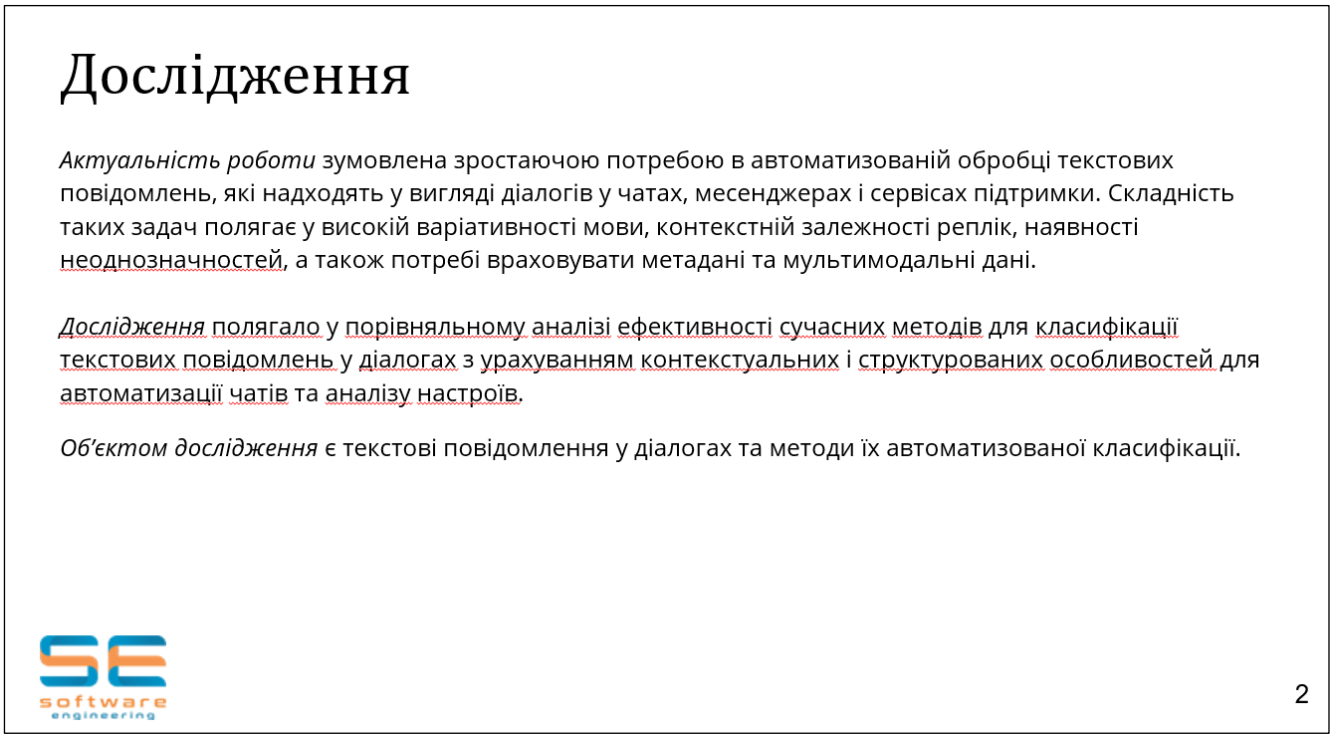
Функції попередньої обробки тексту для наївного Байєса

```
def extract_utterance_label(turn):
    frames = turn.get("frames", [])
    labels = set()
    has_active_intent = False
    for frame in frames:
        state = frame.get("state", {})
        active_intent = state.get("active_intent", "NONE")
        if active_intent != "NONE":
            has_active_intent = True
            break
        service = frame.get("service")
        if service in allowed_labels:
            labels.add(service)
    if has_active_intent or (not labels and not frames):
        return ["None"]
    else:
        return list(labels)

dialogs = []
json_files = glob.glob(os.path.join("train", "*.json"))
for file in json_files:
    with open(file, "r", encoding="utf-8") as f:
        data = json.load(f)
        dialogs.extend(data)

utterance_texts = []
utterance_labels = []
dialog_texts = []
dialog_labels = []

for dialog in dialogs:
    dialog_services = dialog.get("services", [])
    dialog_labels.append(dialog_services)
    dialog_text = " ".join(turn.get("utterance", "") for turn in
dialog.get("turns", []))
    dialog_texts.append(dialog_text)
    for turn in dialog.get("turns", []):
        text = turn.get("utterance", "")
        label = extract_utterance_label(turn)
        utterance_texts.append(text)
        utterance_labels.append(label)
```

ДОДАТОК Г

Слайди презентації





Огляд літератури

Основні джерела:

1. Ashish Vaswani et al., "Attention Is All You Need" (2017)

- Запропонована архітектура трансформера, яка стала основою для сучасних моделей обробки тексту, таких як BERT
- Використання механізму самоуваги дозволяє моделі враховувати контекст у межах тексту.

2. Adar Kahana, Oren Elisha, "MESSAGENET"

- Дослідження ефективності поєднання тексту з метаданими (час, автор, вкладення) у задачах класифікації
- Показано покращення точності класифікації завдяки врахуванню додаткових структурованих атрибутів

Недоліки у наявних дослідженнях

- Класифікація часто виконується на рівні окремих повідомлень без урахування контексту діалогу як послідовності реплік
- Існує обмежена кількість робіт, що порівнюють трансформери, HMM та мультимодальні підходи в багатокласових і багатоміткових задачах.
- Відсутній цілісний підхід, який би поєднував різні типи моделей і даних з урахуванням ресурсної ефективності

3

Постановка задачі

Проблема полягає у відсутності ефективних методів класифікації текстових повідомлень у діалогах, які б одночасно враховували контекст реплік, мультимодальні дані та працювали в умовах багатокласових задач

Очікується, що моделі на основі трансформерної архітектури забезпечать найвищі показники точності класифікації порівняно зі статистичними та лінійними підходами (зокрема, HMM та Naive Bayes), які демонструватимуть нижчі результати. У підсумку буде сформовано впорядкований перелік моделей із зазначенням їх точності та обчислювальних витрат, що дозволить зіставити ефективність і ресурсну складність кожного підходу

4

Методологія

У процесі виконання роботи було проведено **теоретичне дослідження**, на основі якого реалізовано **практичне дослідження**

Під час *теоретичного дослідження* моделі було порівняно за допомогою методу лінійної адитивної згортки з використанням вагових коефіцієнтів. У якості оціночних параметрів розглядалися: ***кількість параметрів моделі***, ***здатність враховувати контекст***, ***обсяг необхідної пам'яті***, ***розмір словника***, ***масштабованість*** та ***інтерпретованість***.

Практичне дослідження було виконане на мові програмування ***Python*** з використанням бібліотек ***HuggingFace Transformers***, ***TensorFlow*** та ***PyTorch***

Теоретичне дослідження

Досліджувані моделі

Моделі на основі трансформерів

BERT, **CLIP**

Вловлюють глибокі контекстуальні зв'язки завдяки механізмам уваги; дуже точні, але потребують багато ресурсів.

Статистичні моделі

Naive Bayes, **HMM**

Використовують ймовірності та припущення; легкі та інтерпретовані, але мають обмеження у роботі зі складним контекстом.

Лінійні моделі

Softmax Regression

Прості, інтерпретовані моделі з використанням зважених вхідних ознак; працюють швидко, але не враховують контекст.

Моделі на основі відстаней

SVM, **KNN**

Класифікують на основі схожості або меж; ефективні для малих/середніх даних, але можуть мати труднощі з масштабними або послідовними даними

Ансамблеві моделі

Random Forest

Поєднують кілька дерев рішень для підвищення надійності прогнозів; добре справляються з нелінійностями, але важче інтерпретуються.

Теоретичне дослідження

Критерії порівняння

	Кількість параметрів	Врахування контексту (токенів)	Необхідна пам'ять (ГБ)	Розмір словника	Масштабованість	Інтерпретованість
Transformers (BERT)	110 млн	512	11	50000	6.35	2
Multimodal Models (CLIP)	112 млн	512	4	49408	5.33	2
HMM	112 млн	512	1	50001	1.60	1
Softmax Regression	112 млн	0	1	0	2.96	2
Naive Bayes	112 млн	0	0.5	0	4.44	3
SVM	112 млн	0	12	0	2.00	1
Random Forest	112 млн	0	12	0	3.57	2
KNN	0	0	12	0	1.28	3

Для оцінки масштабованості було проаналізовано потенціал розпаралелювання обчислень моделі згідно із законом Амдаля

$$S = \frac{1}{(1-p) + \frac{p}{s}}$$, де p – частка задачі, яка може бути паралелізована,
s – кількість процесорів.

	BERT	CLIP	HMM	Softmax Regression	Naive Bayes	SVM	Random Forest	KNN
p	0.95	0.90	0.50	0.70	0.85	0.60	0.80	0.30

Теоретичне дослідження

Розрахунки

1) нормування оцінок за шкалами

$$f = \frac{f_{вимір} - f_{min}}{f_{max} - f_{min}}$$

де f_{min} – мінімальна з оцінок критерію,
f_{max} – максимальна з оцінок критерію,
f_{max} – оцінка для якої відбувається нормалізація

	Кількість параметрів (млн)	Врахування контексту (токенів)	Збережена пам'ять (ГБ)	Розмір словника	Масштаб.	Інтер.
BERT	0	1	0	0.99	1	0.5
CLIP	1	1	0.68	0.98	0.78	0.5
HMM	1	1	0.7	1	0	0
Softmax Regression	1	0	0.7	0	0.28	0.5
Naive Bayes	1	0	1	0	0.59	1

2) Метод лінійної адитивної згортки з використанням вагових коефіцієнтів

$$Z^* = \max_{i=1,m} \sum_{j=1}^{n} a_j b_j a_{ij}$$, де a_j – нормуючий множник,
β_j – ваговий коефіцієнт.

нормуючий множник $a_i = \frac{1}{\sum_{i=1}^{m} a_{ij}}$

Кількість параметрів	Врахування контексту (токенів)	Необхідна пам'ять (ГБ)	Розмір словника	Масштаб.	Інтерпр.
3	5	4	3	5	2

Теоретичне дослідження **Результати**

	Згортка
Transformers (BERT)	0.94533
Multimodal Models (CLIP)	0.8466
HMM	0.5980
Softmax Regression	0.4786
Naive Bayes	0.4173

Визначено *три групи* моделей за ефективністю згортки

Перша група (висока ефективність): **BERT**, **CLIP**

- Добра обробка контексту (до 512 токенів).
- Висока масштабованість.
- Велика кількість параметрів (110–112 млн).
- Найкраще підходять для задач із високими вимогами до точності та обробки великих обсягів даних.

Друга група (середня ефективність): **HMM**

- Підходить для задач із ймовірнісною структурою даних.
- Працює з послідовностями та враховує контекст (512 токенів).
- Середня масштабованість.
- Високі вимоги до пам'яті (11 ГБ).

Третя група (низька ефективність): **Softmax Regression, Naive Bayes**

- Не враховують контекст (0 токенів).
- Низька/середня масштабованість (2.96 для Softmax, 4.44 для Naive Bayes).

Практичне дослідження

Для реалізації BERT

- Двонаправлена трансформерна модель з GRU-декодером.

- Режими:
  - *багатозначна класифікація*: репліки + повний діалог.
  - *однозначна класифікація*: кожна репліка має єдиний клас.

- Навчання з використанням HuggingFace.

Для реалізації CLIP

- Використання попередньо натренованої моделі CLIP.

- Створення класифікатора CLIPClassifier, який об'єднує вектори тексту та зображення.

- Навчання лише класифікаційного шару, freeze параметрів енкодерів

Для реалізації HMM

- Використання GaussianHMM з бібліотеки hmmlearn

- Векторизація через TF-IDF, зниження розмірності через TruncatedSVD

- Алгоритм Баума-Велша для навчання, алгоритм Вітербі для декодування

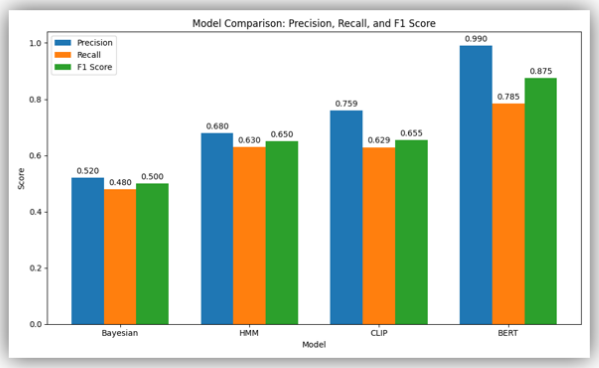
- Побудова мапи латентних станів на класи за допомогою compute_state_mapping

Для реалізації Naive Bayes

- Використання MultinomialNB у зв'язці з TF-IDF + MultiLabelBinarizer.

- Класифікація через One-vs-Rest.

Результати експерименту



BERT продемонстрував найвищі результати за всіма метриками: **Precision – 0.990**, **Recall – 0.785**, **F1 Score – 0.875**. Це підтверджує його здатність глибоко аналізувати контекст у діалогах.

CLIP показав хорошу **точність (0.759)**, але нижчі **Recall (0.629)** та **F1 (0.655)**, що свідчить про ефективність мультимодального підходу при достатній кількості візуальних даних

HMM забезпечив **збалансовані метрики**: **Precision – 0.680**, **Recall – 0.630**, **F1 Score – 0.650**, демонструючи переваги для послідовних діалогових структур

Наївний баєсівський класифікатор показав найгірші результати (**F1 – 0.500**) через неврахування контексту, але залишився швидким і інтерпретованим базовим підходом.

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

$$F1 = 2 \times \frac{\text{Precision} \times Recall}{\text{Precision} + Recal}$$

Підсумки

Найвищу ефективність продемонстрували трансформери, зокрема модель BERT, яка має 110 млн параметрів, обробляє до 512 токенів контексту та забезпечує масштабованість 6.35

Мультимодальна модель CLIP також показала високу ефективність (112 млн параметрів, масштабованість 5.33) та є придатною для задач, де поєднується текст і зображення

Модель HMM продемонструвала середню ефективність, добре працює з послідовностями, має 50000 параметрів і обробляє до 512 токенів, але має низьку масштабованість (1.60)

Softmax Regression і Naive Bayes виявилися недостатньо ефективними для складних задач, мають низьку здатність враховувати контекст і лише середню масштабованість (2.96 і 4.44 відповідно), хоч і відзначаються високою інтерпретованістю

Моделі SVM, Random Forest і KNN показали найгіршу масштабованість (2.00, 3.57 та 1.28 відповідно), не враховують контекст та є ресурсоємними, тому не рекомендовані для використання

У практичних експериментах BERT підтвердив лідерство за точністю, Recall і F1-score на різних наборах даних

CLIP виявився ефективним у мультимодальних задачах, де потрібно аналізувати текст у зв'язці з візуальними ознаками.

HMM перевершив прості статистичні моделі, але не дотягує до трансформерів через обмежену масштабованість

Naive Bayes підтвердив свою слабку придатність до задач із великим обсягом даних і складним контекстом.

Для подальших досліджень рекомендовано використовувати BERT, CLIP та HMM, а інші моделі доцільно виключити через їхні обмеження

ДОДАТОК Д

Апробація результатів роботи

MIT@AIS-2025

19 - 22 may
Kharkiv-Yaremche 2025

Certificate

Oleksandr Lohvinov

particapated in
1st International Scientific and Practical Conference
"Modern Information Technologies and
Artificial Intelligence Systems"
MIT@AIS-2025

Chair of the conference *Yuri Romanenkov*

Kharkiv 2025

ДОДАТОК Е

Результат перевірки на академічний плагіат

StrikePlagiarism.com

Дата звіту 6/6/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_М_ПІ_ІПЗ-23-3_Логвінов_О_В_скорочений
Автор
Логвінов Олександр Володимирович
Науковий керівник / Експерт
Євген Кардаш
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

2.68% 2.68% КП 1

0.97% 0.97% КЦ

25
Довжина фрази для коефіцієнта подібності 2

14298
Кількість слів

109999
Кількість символів

ДОДАТОК Ж

Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008: 2015

Експертний висновок результатів перевірки кваліфікаційної роботи

студент (посада) | програмної інженерії (кафедра) | ІПЗм-23-3 (група)

Логвінов Олександр Володимирович
(прізвище, ім'я, по батькові)

Зауваження

Пункт ДСТУ 3008-2015	Зміст пункту	Сторінка кваліфікаційної роботи
1	2	3
	7.1 Загальні положення	
	7.3 Нумерація сторінок звіту	
	7.4 Нумерація розділів, підрозділів, пунктів, підпунктів	
	7.5 Рисунки	
	7.6 Таблиці	
	7.7 Переліки	
	7.8 Примітки	
	7.9 Виноски	
	7.10 Формули та рівняння	
	7.11 Посилання	
	7.13 Список авторів	
	7.14 Скорочення та умовні познаки	
	7.15 Додатки	

зауважень немає

Експерт _____ (підпис) Олена ОЛІЙНИК (прізвище, ініціали)

07.06.2025