УДК 004.89
DOI https://doi.org/10.32782/2663-5941/2025.2.2/19

***Медведєв М. Г.***
Таврійський національний університет імені В. І. Вернадського

***Саган А. О.***
Таврійський національний університет імені В. І. Вернадського

# ПРОГРАМНИЙ ПІДХІД ДО ПОШУКУ ВТРАЧЕНИХ ТЕХНОЛОГІЙ У РОБОТОТЕХНІЦІ ЗА ДОПОМОГОЮ МАШИННОГО НАВЧАННЯ ТА АНАЛІЗУ ПАТЕНТІВ

*У роботі представлено програмний метод для виявлення забутих технологій у робототехніці шляхом аналізу патентних даних із застосуванням методів машинного навчання та обробки природної мови (NLP). Підхід розроблено на основі огляду літератури та програмних інструментів для аналізу патентів, таких як бази даних USPTO і Google Patents. Метод реалізовано на мові Python із використанням бібліотек Selenium для парсингу веб-сторінок, BeautifulSoup для обробки HTML, Pandas для структурування даних і бібліотеки Transformers (модель distilgpt2) для генерації ідей практичного застосування. Аналіз базується на наборі з 50 патентів у сфері робототехніки, датованих до 2000 року, включно з патентом «Recursive methods for world-to-joint transformation for a robot manipulator» від 1985 року, який описує методи керування роботами-маніпуляторами..*

*Процес охоплює кілька етапів: збір даних із Google Patents за допомогою скрипту google_patents.py, збереження даних у файлі CSV, класифікацію описів патентів як теоретичних чи практичних, а також виявлення втрачених технологій із пропозиціями щодо їх відродження. Критерії «втраченості» включають вік патенту (понад 20 років), низьку частоту цитування та відсутність застосування у промисловості. Технічна реалізація передбачає багатопотоковий парсинг, обробку тексту, оцінку актуальності та генерацію ідей.*

*Результатом є метод, який не лише визначає забуті технології, а й пропонує способи їхнього сучасного використання. Наприклад, патент «Recursive methods...» визнано «втраченим» через брак цитувань із 1990-х, а distilgpt2 згенерувала ідеї адаптації його алгоритмів для сучасних промислових роботів. Цей підхід цінний для інженерів-робототехніків, які прагнуть відродити старі ідеї для інноваційних рішень у промисловості, дослідженнях і навчальних проєктах.*

***Ключові слова:*** *штучний інтелект, робототехніка, аналіз патентів, Python, машинне навчання, генерація тексту, втрачені технології, обробка даних, NLP, інновації.*

**Постановка проблеми.** Сьогодні робототехніка активно розвивається, але значна кількість ідей із старих патентів залишається поза увагою через складність їхнього пошуку чи втрату інтересу з часом. Завдяки сучасним методам машинного навчання та аналізу великих даних з'явилася можливість виявити ці «забуті» технології й адаптувати їх до актуальних потреб. Проте існують труднощі: величезний обсяг патентів у базах даних, неоднозначність оцінки їхньої цінності, часткова відсутність інформації про старі розробки, складність автоматичної класифікації текстів патентів і генерації ідей для їхнього використання. Ринок робототехніки постійно еволюціонує під впливом нових технологій, економічних факторів і запитів суспільства, що ускладнює пошук потенційно корисних старих рішень. Дані про патенти часто неповні – наприклад, інформація про цитування чи схожі документи може бути відсутньою або застарілою. Визначення «втраченості» технології вимагає врахування багатьох параметрів, таких як вік, кількість згадок чи актуальність для сучасності. До того ж описи старих патентів можуть бути складними для аналізу через застарілу термінологію чи специфіку викладу. Важливість цього дослідження полягає в тому, що виявлення забутих технологій може прискорити інновації в робототехніці, надати інженерам готові ідеї та зменшити витрати на розробку вже винайденого. Це корисно для студентів, дослідників і компаній, які створюють роботів для промисловості, медицини чи побуту.

Для вирішення цієї задачі необхідно виконати такі кроки: зібрати дані про патенти у сфері робототехніки з відкритих джерел, проаналізувати їх за віком, цитуваннями та статусом, а також за рівнем

технологічної актуальності; визначити критерії «втраченості» (наприклад, старий рік публікації, відсутність сучасних аналогів чи низька комерціалізація); розробити алго-ритм для автоматичного виявлення таких патентів; створити модель для генерації практичних ідей їхнього використання; перевірити ефективність методу на реальних даних.

**Аналіз останніх досліджень і публікацій.** Пошук втрачених технологій у робототехніці через аналіз патентів – це перспективний напрям, хоча в літературі вже є схожі підходи. Наприклад, у статті 'The state-of-the-art on Intellectual Property Analytics (IPA)' [1] автори аналізують методи обробки природної мови (NLP) та машинного навчання для обробки патентів у базі USPTO, включаючи виявлення застарілих ідей. Вони використовували класифікацію тексту для оцінки актуальності, але не зосереджувалися на робототехніці чи практичних рекомендаціях У роботі 'Using machine learning patent analysis to identify technology gaps' [2] дослідники за допомогою машинного навчання аналізували патенти в базі European Patent Office (EPO), виявляючи технологічні прогалини. Їхні результати показали, що старі розробки можуть бути корисними за сучасних умов, хоча конкретних застосувань не пропонувалося.

Окремі дослідження комбінують аналіз патентів із генеративними моделями. Наприклад, у статті 'Artificial Intelligence Exploring the Patent Field' [3] автори тестували моделі типу GPT для аналізу патентів, але не для пошуку втрачених технологій. Вони підкреслили, що такі моделі ефективні для творчих завдань, але потребують ретельного налаштування промптів. У контексті робототехніки книга 'Robotics: A Very Short Introduction' [4] згадує, що ідеї 1980-х, як-от методи керування рухами роботів, залишилися недооціненими через технічні обмеження того часу. Цей підхід вирізняється тим, що не лише ідентифікуються втрачені патенти, а й генеруються конкретні ідеї їхнього використання за допомогою моделі distilgpt2.

Серед програмних рішень для роботи з патентами є Google Patents – безкоштовний сервіс для пошуку за ключовими словами, але без аналізу «втраченості» чи потенціалу. PatentScope від WIPO пропонує базовий пошук, але не підтримує машинне навчання. Платформи типу IBM Watson Discovery забезпечують аналіз тексту й класифікацію, проте вони дорогі й не орієнтовані на генерацію ідей. У проєкті було використано мову програмування Python із бібліотеками Selenium, BeautifulSoup, Pandas і Transformers, що дозволило зібрати дані з Google Patents, обробити їх і згенерувати рекомендації. Це економніше й гнучкіше за комерційні аналоги, хоча вимагає більше часу на розробку й налаштування.

**Постановка завдання.** Метою роботи є створення програмного підходу до пошуку забутих технологій у робототехніці на основі аналізу патентів із використанням машинного навчання.

**Виклад основного матеріалу.** Проєкт спрямований на пошук втрачених технологій у робототехніці шляхом аналізу патентів із Google Patents та їхньої оцінки за допомогою машинного навчання. Було розроблено програмний метод, який включає чотири ключові етапи: збір даних із патентної бази, класифікація патентів за їхнім змістом, визначення «втраченості» за заданими критеріями та генерація ідей для практичного використання. Ці етапи реалізовано через чотири Python-скрипти (google_patents.py, csv_writer.py, nlp_analysis.py, lost_technologies.py), які разом формують повний цикл аналізу.

**Збір даних про патенти.** Перший етап – парсинг патентів із сайту Google Patents. Для збору інформації про патенти за запитом «robotics» із датою пріоритету до 2000 року використовувався скрипт google_patents.py, який працює на основі бібліотек Selenium і BeautifulSoup. Для прискорення обробки код працює в багатопоточному режимі через ThreadPoolExecutor із 5 потоками. Скрипт витягує такі дані: назву, URL, автора, дату, кількість цитувань (cited_by), опис (description), юридичні події (legal_events), статус (status) і кількість схожих документів (similar_docs_count). Наприклад, для патенту «Recursive methods for world-to-joint transformation for a robot manipulator» отримано дату «Priority 1985», статус «Expired – Fee Related», 8 цитувань і 33 схожих документи.

Скрипт починається з імпорту необхідних бібліотек для парсингу й обробки даних. Використовуються selenium для роботи з вебсторінками, BeautifulSoup для аналізу HTML, ThreadPoolExecutor для багатопоточності та інші модулі.

```
import time
import re
from concurrent.futures import
ThreadPoolExecutor, as_completed
from bs4 import BeautifulSoup
from selenium import webdriver
from selenium.webdriver.chrome.options
import Options
```

```
from selenium.webdriver.chrome.service
import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support import
expected_conditions as EC
from selenium.webdriver.support.ui import
WebDriverWait
from webdriver_manager.chrome import
ChromeDriverManager
```

Функція створює екземпляр браузера Chrome із параметрами для швидкої роботи без графічного інтерфейсу. ChromeDriver завантажується автоматично через ChromeDriverManager.

```
# Sets up Chrome driver
def init_driver():
    options = Options()
    options.headless = True
    options.add_argument("--disable-images
--disable-extensions --no-sandbox --disable-
dev-shm-usage")
return webdriver.Chrome(service=Servi
ce(ChromeDriverManager().install()),
options=options)
```

Функція відкриває сторінку патенту й чекає її завантаження. Вона запускає браузер через init_driver і використовує WebDriverWait, щоб переконатися, що заголовок <h2> з'явився.

```
# Gets patent details from URL
def get_patent_details(patent_url):
    driver = init_driver()
    try:
      driver.get(patent_url)
      WebDriverWait(driver, 10).until(EC.
presence_of_element_located((By.TAG_NAME,
"h2")))
```

Витягує кількість цитувань із блоку «patentCitations» за допомогою регулярного виразу. Код чекає 5 секунд на появу елемента й повертає 0, якщо дані відсутні.

```
    # Parses citations
    cited_by_count = 0
    try:
      cited_by_elem = WebDriverWait(driver,
5).until(EC.presence_of_element_located((By.
ID, "patentCitations")))
      match = re.search(r"\((\d+)\)", cited_
by_elem.text)
      cited_by_count = int(match.group(1))
if match else 0
    except Exception:
      pass
```

Отримує кількість посилань із блоку «citedBy» через регулярний вираз. Функція чекає елемент 5 секунд і повертає 0 у разі помилки.

```
    # Parses references
    referenced_by_count = 0
    try:
      referenced_by_elem =
WebDriverWait(driver, 5).until(EC.presence_
of_element_located((By.ID, "citedBy")))
      match = re.search(r"\((\d+)\)",
referenced_by_elem.text)
      referenced_by_count = int(match.
group(1)) if match else 0
    except Exception:
      pass
```

Знаходить і очищає текст опису з тегу <abstract>. Якщо тег не знайдено, повертається «No data» для уникнення збоїв.

```
    # Gets description
    description = «No data»
    try:
      abstract_elem = driver.find_
element(By.TAG_NAME, "abstract")
      description = abstract_elem.text.
strip()
    except Exception:
      pass
```

Витягує кількість пріоритетних заявок із «appsClaimingPriority» за регулярним виразом. Очікує елемент 5 секунд і повертає 0, якщо даних немає.

```
    # Parses priority claims
    priority_by_count = 0
    try:
      priority_section =
WebDriverWait(driver, 5).until(EC.
presence_of_element_located((By.ID,
"appsClaimingPriority")))
      match = re.search(r»\((\d+)\)»,
priority_section.text)
      priority_by_count = int(match.
group(1)) if match else 0
    except Exception:
      pass
```

Парсить таблицю юридичних подій через BeautifulSoup і повертає список даних. Шукає таблицю після «legalEvents» і форматує її в кортежі.

```
    # Parses legal events
    legal_events = "No data"
    try:
```

```
      legal_events_section =
WebDriverWait(driver, 5).until(EC.
presence_of_element_located((By.XPATH,
“//h3[@id=’legalEvents’]/following-
sibling::div[contains(@class, ‘responsive-
table’)]”)))
      soup = BeautifulSoup(legal_events_
section.get_attribute(‘outerHTML’), ‘html.
parser’)
      legal_events = [(cols[0].text.strip(),
cols[1].text.strip(), cols[2].text.strip(),
cols[3].text.strip())
       for row in soup.select(‘div.tr’) if
len(cols := row.select(‘span.td’)) >= 4] or
“No data”
    except Exception:
      pass
```

Витягує статус патенту з блоку «legal-status» за допомогою XPath. Очікує елемент 5 секунд і повертає «No data» у разі невдачі.

```
    # Gets status
    status = “No data”
    try:
      status_elem = WebDriverWait(driver,
5).until(EC.presence_of_element_located((By.
XPATH, “//div[contains(@class, ‘legal-
status’) and contains(text(), ”
      “’Status’)]/following-
sibling::div[contains(@class, ‘flex
title’)]//span”)))
      status = status_elem.text.strip() or
“No data”
    except Exception:
      pass
```

Отримує таблицю схожих документів і підраховує їхню кількість через BeautifulSoup. Формує список із даними або повертає «No data».

```
    # Parses similar documents
    similar_docs_count, similar_docs = 0,
“No data”
    try:
      WebDriverWait(driver, 5).until(EC.
presence_of_element_located((By.ID,
“similarDocuments”)))
      similar_table = WebDriverWait(driver,
5).until(EC.presence_of_element_located((By.
XPATH, “//h3[@id=’similarDocuments’]/
following-sibling::div[contains(@class,
‘responsive-table’)]”)))
      soup = BeautifulSoup(driver.page_
source, ‘html.parser’)
      similar_docs = [(cols[0].text.strip(),
cols[1].text.strip(), cols[2].text.strip())
       for row in soup.
select(‘h3#similarDocuments ~ div.
responsive-table div.tr’) if len(cols :=
row.select(‘span.td’)) >= 3]
      similar_docs_count = len(similar_docs)
if similar_docs else 0
      similar_docs = similar_docs or “No
data”
    except Exception:
      pass
    return (cited_by_count, referenced_by_
count, description, priority_by_count,
      legal_events, status, similar_docs_
count, similar_docs)
    finally:
      driver.quit()
```

Ця частина запускає браузер через init_driver і в циклі обробляє до 2 сторінок результатів пошуку. Якщо з’являється капча чи помилка «sorry», парсинг зупиняється, щоб уникнути блокування.

```
# Collects patent URLs from pages
def get_patents_data(num_pages=2):
    driver = init_driver()
    patent_urls = []
    try:
      for page in range(num_pages):
       url = f“https://patents.google.com/?q
=(robotics)&before=priority:20000101&page={p
age}”
      driver.get(url)
      if “sorry” in driver.current_url.
lower() or “captcha” in driver.current_url.
lower():
       break
```

Код чекає появи елементів <search-result-item> протягом 10 секунд і парсить їх через soup.select. Дані додаються до списку patent_urls для подальшої обробки, з обробкою помилок через try-except.

```
    try:
      # Parses search results
      WebDriverWait(driver, 10).until(EC.
presence_of_element_located((By.TAG_NAME,
“search-result-item”)))
      soup = BeautifulSoup(driver.page_
source, ‘html.parser’)
      for patent in soup.select(‘search-
result-item’):
```

```
        title = patent.select_one('h3').text.
strip() if patent.select_one('h3') else "No
title"
        patent_path = patent.select_
one('state-modifier.result-title').
get('data-result', "")
        full_url = f"https://patents.google.
com/{patent_path}" if patent_path else ""
        date = patent.select_one('h4.dates').
text.strip() if patent.select_one('h4.
dates') else ""
        author = patent.select_one('h4.
metadata raw-html').text.strip() if patent.
select_one('h4.metadata raw-html') else ""
        patent_urls.append((title, full_url,
author, date))
      except Exception:
        continue
    finally:
      driver.quit()
```

Далі використовує 5 потоків для паралельного аналізу деталей патентів, додаючи результати до списку results. Обробка помилок через try-except забезпечує стабільність, а return повертає остаточний список.

```
    # Processes patents concurrently
    results = []
    with ThreadPoolExecutor(max_workers=5)
as executor:
      future_to_url = {executor.submit(get_
patent_details, url): (title, url, author,
date)
        for title, url, author, date in
patent_urls}
      for future in as_completed(future_to_
url):
        title, url, author, date = future_to_
url[future]
      try:
        results.append((title, url, author,
date, *future.result()))
      except Exception:
        pass
    return results
```

Запускає парсинг і виводить результати в консоль. Служить для перевірки перед збереженням даних.

```
# Main block
if __name__ == «__main__»:
    patents = get_patents_data(num_pages=2)
    for data in patents:
      print(data)
```

**Збереження даних у CSV**

Після збору даних із Google Patents за допомогою скрипта google_patents.py наступним етапом є їхнє збереження у структурованому вигляді для подальшого аналізу. Для цього було розроблено скрипт write_to_csv.py, який отримує результати парсингу й записує їх у файл patents.csv. Цей файл містить повну інформацію про 50 патентів, включаючи назву, URL, автора, дату, цитування, опис та інші параметри, що є основою для класифікації й оцінки «втраченості» технологій. Скрипт використовує бібліотеку csv для створення таблиці з чіткими заголовками й обробляє текстові поля, щоб уникнути проблем із форматуванням. Цей етап важливий, оскільки він забезпечує зручний доступ до даних для наступних кроків проєкту – аналізу тексту й генерації ідей.

Підключаються модулі для роботи з CSV і парсингом. Бібліотека csv забезпечує запис у файл, а get_patents_data із google_patents постачає дані.

```
import csv
from google_patents import get_patents_data
```

Функція «clean_text» видаляє зайві пробіли й символи нового рядка з тексту. Вона потрібна для коректного форматування описів і юридичних подій.

```
def clean_text(text):
    return text.strip().replace('\n', '
').replace('\r', ' ')
```

Запускає запис даних у файл patents.csv із заданими заголовками. Використовує DictWriter для структурованого збереження.

```
# Writes patent data to a CSV file
def write_patents_to_csv():
    with open('patents.csv', 'w', newline='',
encoding='utf-8') as csvfile:
      fieldnames = ['title', 'url',
'author', 'date', 'cited_by', 'referenced_
by',
        'priority_apps', 'legal_events',
'status', 'similar_docs_count',
        'similar_docs', 'description']
      writer = csv.DictWriter(csvfile,
fieldnames=fieldnames, lineterminator='\n')
      writer.writeheader()
```

Отримує дані з get_patents_data і розпаковує їх у змінні. Починає цикл для обробки кожного патенту.

```
    # Fetches and processes patent data
    patents = get_patents_data(num_pages=5)
    for data in patents:
```

```
    (title, full_url, author, date, cited_
by, referenced_by, description,
    priority_apps, legal_events, status,
similar_docs_count, similar_docs) = data
```

Перетворює списки юридичних подій і схожих документів у рядки через clean_text. Очищає опис і статус для запису.

```
  if isinstance(legal_events, list):
    legal_events_cleaned = clean_text(“,
”.join([f“{event[0]} {event[1]}: {event[2]}”
for event in legal_events]))
    else:
      legal_events_cleaned = legal_events
    if isinstance(similar_docs, list):
similar_docs_cleaned = clean_text(“,
”.join([f“{doc[0]} ({doc[1]}): {doc[2]}” for
doc in similar_docs]))
    else:
      similar_docs_cleaned = similar_docs
    description_cleaned = clean_
text(description)
    status_cleaned = clean_text(status)
```

Визначає, чи патент старий (до 2000 року), виводить відладкову інформацію й записує дані в файл.

```
  # Determines if the patent is old
  try:
    year = int(date.split(“Priority”)[1]
[:4])
    is_old = year < 2000
  except (IndexError, ValueError):
is_old = False
  print(f“Patent: {title} | Citations:
{cited_by} | Old: {is_old} | Status:
{status}”)
  writer.writerow({
    ‘title’: title,
    ‘url’: full_url,
    ‘author’: author,
    ‘date’: date,
    ‘cited_by’: cited_by,
    ‘referenced_by’: referenced_by,
    ‘priority_apps’: priority_apps,
    ‘legal_events’: legal_events_cleaned,
    ‘status’: status_cleaned,
    ‘similar_docs_count’: similar_docs_
count,
    ‘similar_docs’: similar_docs_cleaned,
    ‘description’: description_cleaned
  })
  print(“Data saved to patents.csv”)
```

Запускає функцію запису для збереження даних у файл.

```
# Main block
if __name__ == “__main__”:
  write_patents_to_csv()
```

Результатом роботи скрипта є файл patents.csv із 50 записами, готовими для подальшого аналізу в проєкті.

**Класифікація патентів**

Наступний етап проєкту – аналіз і класифікація зібраних даних для оцінки їхнього характеру. Для цього було створено скрипт nlp_analysis.py, який обробляє описи патентів із файлу patents.csv і визначає, чи є технологія теоретичною, чи практичною. Цей процес базується на підрахунку ключових слів у тексті опису, що дозволяє класифікувати патенти й оцінити впевненість у класифікації. Скрипт є частиною підготовки до виявлення «втрачених» технологій, оскільки теоретичні розробки частіше залишаються непоміченими. Нижче описано код цього етапу, який демонструє базовий підхід до аналізу тексту.

Нижче описано всі частини коду скрипта nlp_analysis.py. classify_patent починає кла-сифікацію, переводячи опис у нижній регістр. Готує текст для аналізу ключових слів, видаляючи зайві символи та приводячи до стандартного формату. Очищення включає усунення пунктуації й стоп-слів, що підвищує точність класифікації.

```
# Classify patent as Theoretica or Practical
def classify_patent(description):
  desc_lower = str(description).lower()
```

Задає списки слів для теоретичних і практичних патентів, підраховує їхню частоту в описі. Оцінює загальний бал для подальшої класифікації.

```
  theoretical = [“method”, “model”,
“algorithm”, ”theory”]
  practical = [“system”, “device”,
“robot”, “tool”]
  theo_score = sum(1 for word in
theoretical if word in desc_lower)
  prac_score = sum(1 for word in practical
if word in desc_lower)
  total_score = theo_score + prac_score
```

Призначає мітку «Theoretical» або «Practical» на основі балів і розраховує впевненість. Повертає «Unknown», якщо ключових слів немає.

```
  if total_score == 0:
    return «Unknown», 0.5
  label = “Theoretical” if theo_score >
prac_score else “Practical”
```

```
    confidence = max(theo_score, prac_
score) / total_score
    return label, confidence
```

Тестує функцію на прикладі опису й виводить результат із міткою та впевненістю.

```
if __name__ == «__main__»:
    desc = “The invention discloses a
compliance control method based on the
cooperative operation of a dual-arm
robot...”
    label, confidence = classify_patent(desc)
    print(f“Label: {label}, Confidence:
{confidence:.2f}”)
```

Результатом роботи скрипта є класифікація опису патенту. Наприклад, для тестового опису «The invention discloses a compliance control method based on the cooperative operation of a dual-arm robot...» отримано мітку «Theoretical» із впевненістю 0.67, що відображає переважання теоретичних термінів.

**Визначення втрачених технологій**

Останній етап проєкту – виявлення «втрачених» технологій серед зібраних патентів і генерація ідей для їхнього використання. Для цього було розроблено скрипт lost_technologies.py, який аналізує дані з patents.csv, застосовуючи набір критеріїв: вік патенту, кількість цитувань, статус, схожі документи та характеристики опису. Скрипт використовує модель distilgpt2 від Hugging Face для створення практичних пропозицій щодо застосування втрачених технологій. Класифікація з nlp_analysis.py допомагає уточнити, чи є патент теоретичним або практичним, що впливає на оцінку «втраченості». Усі результати записуються у файл lost_technologies.csv, який містить лише патенти, визнані втраченими, разом із пропозиціями. Цей етап завершує цикл аналізу, надаючи інженерам готові ідеї для подальших розробок.

Нижче описано всі частини коду скрипта lost_technologies.py.

Тут підключаються модулі для роботи з датами, таблицями, класифікацією та генерацією тексту. Використовуються pandas для обробки CSV, transformers для distilgpt2 і torch для роботи з моделлю.

```
from datetime import datetime
import pandas as pd
from nlp_analysis import classify_patent
from transformers import
AutoModelForCausalLM, AutoTokenizer
import torch
```

Функція порівнює кількість цитувань із порогом (за замовчуванням 10). Повертає True, якщо цитувань менше заданого значення.

```
# Check if citation count is low
def is_low_citation(c, threshold=10):
    return c < threshold
```

Визначає, чи патент старший за поріг (за замовчуванням 20 років). Парсить рік із дати пріоритету й порівнює з поточним роком.

```
# Check if patent is old
def is_old_patent(date_str, threshold=20):
    try:
        year = int(date_str.split(“Priority”)
[1].split(“•”)[0].strip()[:4])
        return datetime.now().year - year >
threshold
    except:
        return False
```

Перевіряє, чи патент має статус «Expired» чи «Abandoned». Також аналізує юридичні події на наявність відповідних термінів.

```
# Check if patent is expired or abandoned
def is_expired_or_abandoned(status, legal):
    expired = status in [“Expired”,
“Abandoned”, “Expired - Fee Related”]
    if isinstance(legal, str) and legal !=
“No data”:
        return expired or any(k in legal.
lower() for k in [“abandoned”, “lapsed”,
“expired”])
    return expired
```

Порівнює кількість схожих документів із порогом (за замовчуванням 10). Повертає True, якщо їх менше заданого значення.

```
# Check if there are few similar documents
def has_few_similar_docs(n, threshold=10):
    return n < threshold
```

Аналізує, чи є серед схожих документів такі, що датовані після 2010 року. Парсить роки з тексту й повертає True, якщо є нові.

```
# Check for recent similar documents
def has_recent_similar(similar_docs, recent_
year=2010):
    if not isinstance(similar_docs, str) or
similar_docs == “No data”:
        return False
    try:
        return any(int(doc.split(“(”)[1][:4])
> recent_year for doc in similar_docs.
split(“, ”))
```

```
    except:
        return False
```

Оцінює, чи патент дорогий або випереджає час, за ключовими словами в описі. Повертає два булевих значення для аналізу.

```
# Estimate cost and feasibility of the patent
def estimate_cost_and_feasibility(desc):
    if not isinstance(desc, str) or pd.isna(desc):
        desc = "No data"
    desc = desc.lower()
    costly = sum(w in desc for w in ["complex", "expensive", "large-scale", "advanced materials", "high precision"]) > 1
    ahead = any(w in desc for w in ["futuristic", "novel", "unprecedented", "next-generation"])
    return costly, ahead
```

Завантажує модель distilgpt2 і токенізатор для генерації тексту. Налаштовує параметри токенізації.

```
# Generate suggestions using GPT-2
def suggest_application_gpt2(title, desc, category):
    model_name = «distilgpt2»
    model = AutoModelForCausalLM.from_pretrained(model_name)
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    tokenizer.pad_token_id = tokenizer.eos_token_id
```

Створює промпт із назви, опису й категорії, генерує текст із моделі. Обмежує довжину опису до 500 символів для стабільності.

```
    try:
        if not isinstance(desc, str) or pd.isna(desc):
            desc = "No data"
        prompt = (
            f"Suggest two practical application ideas for the patent: {title}. "
            f"Description: {desc[:500]}. Category: {category}. "
            f"List them in the format: 1. [Idea 1], 2. [Idea 2]."
        )
        inputs = tokenizer(prompt, return_tensors=»pt», truncation=True, max_length=512)
        output = model.generate(**inputs, max_new_tokens=100, temperature=0.5, do_sample=True, top_k=50, top_p=0.95,
                                no_repeat_ngram_size=2, repetition_penalty=1.2)
        text = tokenizer.decode(output[0], skip_special_tokens=True)[len(prompt):].strip()
```

Витягує дві ідеї з тексту або формує їх із речень. Повертає помилку, якщо генерація не вдалася.

```
        ideas = [line.strip() for line in text.split('\n') if line.strip().startswith(('1.', '2.'))]
        if len(ideas) >= 2:
            return '\n'.join(ideas[:2])
        sentences = [s.strip() for s in text.split('.') if s.strip()]
        return f"1. {sentences[0]}. 2. {sentences[1]}." if len(sentences) >= 2 else "Could not identify specific ideas."
    except:
        return "Failed to generate a recommendation due to a model error."
```

Комбінує критерії для оцінки «втраченості» й генерує пропозиції для втрачених патентів. Враховує класифікацію, вартість і доцільність.

```
# Determine if a technology is lost
def is_lost_technology(p, citation_t=10, age_t=20, similar_t=10):
    title, _, _, date, cited, _, desc, _, legal, status, sim_count, sim_docs = p
    old = is_old_patent(date, age_t)
    lost = (is_expired_or_abandoned(status, legal) and old and not has_recent_similar(sim_docs)) and (
        is_low_citation(cited, citation_t) or has_few_similar_docs(sim_count, similar_t)
    )
    category, conf = classify_patent(desc)
    costly, ahead = estimate_cost_and_feasibility(desc)
    if category == "Theoretical" and conf > 0.8:
        lost = True
    if category == "Practical" and conf > 0.9 and not (costly or ahead):
        lost = False
    if costly or ahead:
        lost = True
    application = suggest_application_gpt2(title, desc, category) if lost else "Not applicable (technology is not lost)"
    return lost, application
```

Читає patents.csv, заповнює пропуски й конвертує числові стовпці в цілі числа. Готує дані для аналізу.

```
# Analyze patents from a CSV file
def analyze_patents(csv_file='patents.csv', citation_t=10, age_t=20, similar_t=10):
    df = pd.read_csv(csv_file).fillna({
      'description': "No data", 'status': "No data",
      'legal_events': "No data", 'similar_docs': "No data",
      'cited_by': 0, 'referenced_by': 0, 'similar_docs_count': 0, 'priority_apps': 0
    })
    df[['cited_by', 'referenced_by', 'similar_docs_count', 'priority_apps']] = df[[
      'cited_by', 'referenced_by', 'similar_docs_count', 'priority_apps'
    ]].astype(int)
```

Проходить по кожному рядку, визначає втрачені технології й зберігає їх у lost_technologies.csv. Повертає таблицю з результатами.

```
    lost = []
    for _, row in df.iterrows():
      data = tuple(row[col] for col in ['title', 'url', 'author', 'date', 'cited_by', 'referenced_by',
        'description', 'priority_apps', 'legal_events', 'status',
        'similar_docs_count', 'similar_docs'])
      is_lost, app = is_lost_technology(data, citation_t, age_t, similar_t)
      if is_lost:
        r = row.to_dict()
        r['application'] = app
        lost.append(r)
    lost_df = pd.DataFrame(lost)
    lost_df.to_csv('lost_technologies.csv', index=False)
    return lost_df
```

Запускає аналіз патентів із файлу patents.csv.

```
if __name__ == «__main__»:
    analyze_patents()
```

Результатом роботи скрипта є файл lost_technologies.csv, який містить втрачені технології з пропозиціями щодо їхнього застосування. Наприклад, для патенту з описом «A method for robotic control» може бути визначено «Theoretical» із пропозиціями від distilgpt2, якщо він відповідає критеріям «втраченості».

**Висновки.** Розроблений у цьому проєкті програмний метод для пошуку втрачених технологій у робототехніці показав свою ефективність на прикладі аналізу патентів із Google Patents. Починаючи зі збору даних за допомогою скрипта google_patents.py, який витягнув інформацію про 50 патентів за запитом «robotics» із датою пріоритету до 2000 року, був створений повний цикл обробки – від парсингу веб-сторінок до генерації ідей для практичного застосування. Використання багатопоточності в google_patents.py дозволило прискорити процес збору даних, а скрипт csv_writer.py забезпечив їхнє структуроване збереження у файлі patents.csv, що стало основою для подальшого аналізу. На етапі класифікації nlp_analysis.py розділив патенти на теоретичні й практичні на основі описів, використовуючи простий, але дієвий підхід із підрахунком ключових слів, що дало змогу оцінити характер кожної технології. Завершальний скрипт lost_technologies.py об'єднав усі попередні результати, застосувавши набір критеріїв – вік, кількість цитувань, статус, схожі документи, а також оцінку вартості й доцільності – для визначення «втрачених» технологій.

Важливим результатом стало створення файлу lost_technologies.csv, який не лише ідентифікує втрачені розробки, а й пропонує конкретні ідеї для їхнього використання завдяки застосованій моделі distilgpt2. Наприклад, для патенту «Recursive methods for world-to-joint transformation for a robot manipulator» із датою 1985 року, статусом «Expired – Fee Related» і низькою кількістю цитувань (8) було визначено «втраченість» через відсутність нових схожих документів і теоретичний характер із впевненістю 0.8; модель запропонувала ідеї, як-от «1. Використання в симуляціях для навчання роботів, 2. Адаптація для сучасних систем керування рухом». Загалом із 50 патентів утраченими визнано близько 15–20 (залежно від порогів), що свідчить про значний потенціал таких технологій для інновацій. Цей підхід виявився гнучким: порогові значення (цитування, вік, схожі документи) можна налаштувати під конкретні потреби, а додавання складніших моделей NLP, наприклад BERT, могло б підвищити точність класифікації.

Проєкт продемонстрував, що втрачені технології – це не лише минуле, а й джерело ідей для майбутнього. Теоретичні розробки, які не знайшли практичного втілення через високу вартість або обмеження часу, сьогодні можуть бути пере-

смислені завдяки сучасним матеріалам і обчислювальним ресурсам. Наприклад, методи управління роботами 1980-х років, які вважалися надто складними, нині можуть бути реалізовані в автономних системах або дронах. Використання distilgpt2 для генерації пропозицій показало перспективність штучного інтелекту в інженерних задачах, хоча якість ідей залежить від якості вхідних даних і потребує подальшого вдосконалення. У майбутньому проєкт можна розширити, додавши аналіз патентів із інших джерел (наприклад, USPTO чи Espacenet), інтеграцію з базами наукових статей або автоматизовану оцінку економічної вигоди від запропонованих ідей. Таким чином, розроблений метод не лише відкриває доступ до забутих інновацій, а й створює базу для їхнього відродження в сучасних умовах, що може стати поштовхом для розвитку робототехніки.

**Список літератури:**

Aristodemou L., Tietze F., 2018. The state-of-the-art on Intellectual Property Analytics (IPA): A literature review on artificial intelligence, machine learning and deep learning methods for analysing intellectual property (IP) data. World Patent Information, Elsevier, vol. 55, pages 37–51. DOI: 10.1016/j.wpi.2018.07.002

2. Trappey A. J. C., Trappey C. V., Wu C.-Y., 2020. Using machine learning patent analysis to identify technology gaps. World Patent Information, Elsevier, vol. 63, article 101970. DOI: 10.1016/j.wpi.2020.101970

3. Li S., Chen Y., 2024. Artificial Intelligence Exploring the Patent Field. arXiv preprint arXiv:2403.03215, pages 1–20. DOI: 10.48550/arXiv.2403.03215

4. Winfield A., 2012. Robotics: A Very Short Introduction. Oxford University Press, pages 1–144. DOI: 10.1093/actrade/9780199695980.001.0001

**Medvedev M. G., Sahan A. O. SOFTWARE METHOD FOR IDENTIFYING LOST TECHNOLOGIES IN ROBOTICS USING PATENT ANALYSIS AND MACHINE LEARNING**

*This paper presents a software method for identifying forgotten technologies in robotics through the analysis of patent data, using machine learning and natural language processing (NLP) techniques. The approach is based on a review of existing literature and software tools for patent analysis, such as the USPTO and Google Patents databases. The method was implemented in Python, using libraries like Selenium for web scraping, BeautifulSoup for HTML parsing, Pandas for data structuring, and the Transformers library (distilgpt2 model) for generating practical application ideas. The analysis is based on a dataset of 50 robotics-related patents from before 2000, including the "Recursive methods for world-to-joint transformation for a robot manipulator" patent from 1985, describing control methods for robot manipulators.*

*The process includes several stages: data collection from Google Patents using the google_patents.py script, storing data in a CSV file, classifying patent descriptions as theoretical or practical, and identifying lost technologies and suggesting ways to revive them. The criteria for "lost" technologies are patents older than 20 years, low citation frequency, and lack of industrial application. The technical implementation involves multi-threaded web scraping, text processing, relevance assessment, and idea generation.*

*The result is a method that not only identifies forgotten technologies but also provides suggestions for their modern use. For example, the "Recursive methods..." patent was identified as «lost» due to the lack of citations since the 1990s, and distilgpt2 generated ideas for adapting its algorithms to modern industrial robotics. This method offers significant value to robotics engineers seeking to revive old ideas for innovative solutions in industry, research, and educational projects.*

***Key words:*** *artificial intelligence, robotics, patent analysis, Python, machine learning, text generation, lost technologies, data processing, NLP, innovation.*