ระบบติดป้ายกำกับข้อมูลอัตโนมัติด้วย AI โดยใช้ Prolog และ Large Language Models

ระบบ Auto-Labeling Engine สำหรับข้อมูลตารางทั่วไป

December 5, 2025

Abstract

บทความนี้นำเสนอระบบติดป้ายกำกับข้อมูลอัตโนมัติที่ผลรวมความสามารถในการให้เหตุผลของ Prolog เข้ากับความเข้าใจภาษาธรรมชาติของ Large Language Models (LLMs) ระบบนี้ช่วยให้ผู้เชี่ยวชาญในสาขาต่างๆ สามารถแสดงกฎการติดป้ายกำกับด้วยภาษาธรรมชาติ ซึ่งจะถูกแปลงเป็นประพจน์ First-Order Logic Prolog โดยอัตโนมัติผ่าน Gemini API จากนั้นกฎเหล่านี้จะถูกนำไปใช้กับข้อมูลในรูปแบบ CSV โดยใช้ PySwip ซึ่งเป็นตัวเชื่อมระหว่าง Python และ Prolog ระบบสามารถรองรับกฎแบบลูกโซ่ (Chain Rules) สำหรับตัวตรรกะที่ซับซ้อน การจำแนกหลายป้ายกำกับพร้อมกัน (Multi-label) และมีระบบแสดงผลกราฟอัตโนมัติ เรานำเสนอ Auto-Labeling Engine ใหม่ที่ออกแบบมาเพื่อรองรับข้อมูลตารางทั่วไป (Tabular Data) และสามารถปรับใช้กับโดเมนต่างๆ ได้อย่างง่ายดายผ่านการตั้งค่า JSON โดยไม่ต้องเปลี่ยนโค้ด

1 บทนำ

การติดป้ายกำกับข้อมูล (Data Labeling) เป็นงานที่สำคัญแต่ใช้แรงงานมากในกระบวนการเตรียมข้อมูลสำหรับการเรียนรู้ของเครื่อง (Machine Learning) และการวิเคราะห์ข้อมูล วิธีการแบบดั้งเดิมต้องอาศัยการอธิบายด้วยตนเองจากผู้เชี่ยวชาญหรือใช้ระบบกฎที่ต้องเขียนโปรแกรมที่ซับซ้อน งานวิจัยนี้นำเสนอแนวทางแบบผสมผสาน (Hybrid Approach) ที่:

1. รับคำอธิบายกฎในรูปแบบภาษาธรรมชาติจากผู้เชี่ยวชาญ
2. แปลงเป็นตรรกะเชิงรูปแบบ (Formal Logic) โดยอัตโนมัติด้วย Prolog
3. ประยุกต์ใช้กฎเหล่านี้กับข้อมูลขนาดใหญ่อย่างสม่ำเสมอ
4. รองรับการให้เหตุผลที่ซับซ้อนผ่าน Chain Rules
5. ให้ข้อมูลป้อนกลับและการตรวจสอบผ่านภาพ (Visual Feedback)

1.1 Auto-Labeling Engine ที่พัฒนาขึ้นใหม่

เราได้พัฒนา Auto-Labeling Engine ที่ออกแบบมาโดยเฉพาะสำหรับข้อมูลตารางทั่วไป (Generic Tabular Data) โดย Engine นี้มีคุณสมบัติหลักดังนี้:

- **Configuration-Driven Architecture**: สามารถปรับใช้กับข้อมูลชุดใหม่ได้โดยแก้ไขแค่ไฟล์ JSON โดยไม่ต้องเปลี่ยนโค้ด

- **Point-based Inference**: ประมวลผลการให้เหตุผล (Inference) แบบ Row-by-Row โดยพิจารณาค่าของแต่ละจุดข้อมูลในขณะนั้น
- **Dynamic Prolog Integration**: โหลดและประมวลผลกฎ Prolog แบบไดนามิก รองรับ Chain Rules และ Helper Predicates
- **Multi-domain Support**: รองรับหลากหลายสาขา เช่น การตรวจสอบคุณภาพอากาศ, การติดตามสุขภาพ, การควบคุมคุณภาพในอุตสาหกรรม
- **Automatic Visualization**: สร้างกราฟแสดงผลอัตโนมัติสำหรับการวิเคราะห์ข้อมูลเชิงเวลา (Time-series Analysis)

ความแตกต่างจากระบบอื่นๆ คือ Engine ของเราเน้นความง่ายในการใช้งานสำหรับผู้ที่ไม่มีความรู้ด้านการเขียนโปรแกรม โดยใช้ภาษาธรรมชาติเป็นอินเทอร์เฟซหลัก และรักษาความสามารถในการอธิบายได้ (Explainability) ผ่าน Prolog Rules ที่มนุษย์อ่านเข้าใจได้

2 สถาปัตยกรรมระบบ

2.1 องค์ประกอบหลัก

ระบบประกอบด้วย 4 ส่วนหลัก:

1. **ส่วนติดต่อผู้ใช้แบบภาษาธรรมชาติ (Natural Language Interface)**: ใช้ Tkinter GUI รับข้อมูลจากผู้ใช้
2. **ตัวแปลภาษาด้วย LLM (LLM Translation Engine)**: ใช้ Gemini API แปลงภาษาธรรมชาติเป็นกฎ Prolog
3. **ตัวประมวลผลตรรกะ Prolog (Prolog Inference Engine)**: ใช้ PySwip นำกฎไปใช้กับข้อมูล
4. **ส่วนแสดงผลกราฟ (Visualization Module)**: ใช้ Matplotlib แสดงผลการวิเคราะห์ข้อมูลเชิงเวลา

2.2 ขั้นตอนการทำงาน

1. ผู้ใช้ป้อนกฎเป็นภาษาธรรมชาติ (ไทยหรืออังกฤษ)
2. ระบบโหลดการตั้งค่าสำหรับ Use Case ที่เลือก
3. Gemini API แปลงเป็นประพจน์ Prolog
4. ตรวจสอบและบันทึกกฎพร้อม Timestamp
5. PySwip โหลดกฎและ Query แต่ละแถวข้อมูล
6. ติดป้ายกำกับและบันทึกลง CSV พร้อม Metadata
7. แสดงผลรูปแบบกราฟและการกระจายตัวของป้ายกำกับ

2.3 Auto-Labeling Engine Architecture

Engine ที่เราพัฒนาขึ้นมีสถาปัตยกรรมแบบ Modular ที่แยกส่วนการทำงานออกเป็นโมดูลต่างๆ:

- **Configuration Manager** (`query_engine_config.py`): จัดการการตั้งค่าทั้งหมด รวมทั้ง path, ชื่อไฟล์, และการแมปตัวแปร
- **Rule Parser** (`extract_predicates_from_rules`): แยกวิเคราะห์กฎ Prolog เพื่อหาชื่อ Predicate, จำนวนและชื่อของ Arguments
- **Data Mapper**: แปลงข้อมูลจาก CSV เป็นรูปแบบที่ Prolog เข้าใจได้ (เช่น Time "HH:MM" $\rightarrow$ Integer Hour)
- **Inference Engine** (`apply_rule_to_csv`): ประมวลผลแต่ละแถวด้วย Point-based Inference
- **Visualization Generator** (`plot_labeled_results`): สร้างกราฟแสดงผลอัตโนมัติ

3 วิธีการ (Methodology)

3.1 การแปลงภาษาธรรมชาติเป็น Prolog

ระบบใช้ Prompt Template ที่ออกแบบมาอย่างละเอียดซึ่งระบุ:

- ไวยากรณ์ Prolog และตัวดำเนินการที่ถูกต้อง ($\leq$, $\geq$, $>$, $<$)
- รูปแบบ Predicate แบบหลายอาร์กิวเมนต์: `predicate(Var1, Var2, ..., 'label') :- conditions`
- กฎการตั้งชื่อตัวแปร (ตัวแปรขึ้นต้นด้วยตัวพิมพ์ใหญ่, predicates เป็นตัวพิมพ์เล็ก)
- รูปแบบ Chain Rules สำหรับตรรกะที่ซับซ้อน

ตัวอย่างการแปลง:

Input (ภาษาธรรมชาติ):

> "จำแนกคุณภาพอากาศเป็น 3 ระดับ: ดี (PM2.5 < 25), ปานกลาง (PM2.5 25-50), แย่ (PM2.5 > 50)"

Output (Prolog):

```
good_pm(PM2_5) :- PM2_5 < 25.
moderate_pm(PM2_5) :- PM2_5 >= 25, PM2_5 =< 50.
poor_pm(PM2_5) :- PM2_5 > 50.

label_air_quality(PM2_5, ' ') :- good_pm(PM2_5).
label_air_quality(PM2_5, '    ') :- moderate_pm(PM2_5).
label_air_quality(PM2_5, ' ') :- poor_pm(PM2_5).
```

3.2 การรองรับ Chain Rules

ระบบรองรับการให้เหตุผลแบบลำดับชั้นผ่าน Helper Predicates:

```
% Helper predicates  ( label argument)
high_pm(PM2_5) :- PM2_5 > 50.
high_temp(Temperature) :- Temperature > 25.

%           ( helpers)
label_condition(PM2_5, Temperature, '     ') :-
    high_pm(PM2_5), high_temp(Temperature).
```

3.3 กระบวนการใช้กฎ (Point-based Inference)

สำหรับแต่ละแถวในชุดข้อมูล:

1. แยกค่าและแมปกับตัวแปร Prolog
2. แปลงชนิดข้อมูล (เช่น “HH:MM” → ชั่วโมงเป็นจำนวนเต็ม)
3. Assert facts เข้าไปใน Prolog Knowledge Base
4. Query ทุก Label Predicates ด้วยค่าของแถวปัจจุบัน
5. รวบรวมป้ายกำกับที่ตรงเงื่อนไข (single-label หรือ multi-label)
6. Retract facts เพื่อเตรียมสำหรับแถวถัดไป

หมายเหตุสำคัญ: กระบวนการนี้เป็นแบบ *stateless* คือแต่ละแถวถูกประมวลผลอิสระจากกัน ไม่มีการเก็บข้อมูลจากแถวก่อนหน้า ดังนั้น **ไม่สามารถใช้กฎที่ต้องการเปรียบเทียบกับข้อมูลย้อนหลังได้** เช่น:

- “ถ้าค่าลดลง 5 หน่วยจากแถวก่อนหน้า”
- “ถ้าค่าเฉลี่ยของ 3 แถวล่าสุดมากกว่า 50”

4 Implementation Details

4.1 Configuration Management

The system uses JSON configuration files for each use case:

```
{
  "use_case": "PM_Temperature",
  "prolog_variables": [
    {"csv_column": "Temp", "prolog_name": "Temperature"},
    {"csv_column": "PM2.5", "prolog_name": "PM2_5"},
    {"csv_column": "Time", "prolog_name": "TimeHour"}
```

```
  ],
  "labeling": {
    "label_column": "auto_label",
    "multi_label": true
  }
}
```

4.2 Argument Parsing

Critical feature: The system parses Prolog rules to extract argument names and order:

```
def extract_predicates_from_rules(rule_file):
    # Parse: predicate(Arg1, Arg2, ..., 'Label') :- condition
    match = re.search(r"(\w+)\((.+),\s*'([^']+)'\)\s*:-", line)
    arg_list = [a.strip() for a in args_str.split(',')]
    # Handles underscore variables: _, _Var
```

4.3 Visualization

The system automatically generates two-panel visualizations:

1. **Time Series Plot**: Shows PM2.5 and Temperature trends with labeled regions highlighted as colored spans
2. **Distribution Chart**: Bar chart showing frequency of each label

Consecutive data points with the same label are grouped into shaded regions, reducing visual clutter.

5 กรณีศึกษา: ระบบตรวจสอบคุณภาพอากาศ PM2.5

5.1 ชุดข้อมูล

เราใช้ข้อมูลคุณภาพอากาศเป็นตัวอย่างการใช้งานระบบ (ไม่ใช่จุดเน้นหลักของ Engine):

- **แหล่งข้อมูล**: การวัด 72 ชั่วโมงต่อเนื่อง (3 วัน)
- **Features**: อุณหภูมิ (°C), PM2.5 (□g/m^3), เวลา (HH:MM)
- **วัตถุประสงค์**: จำแนกสภาพคุณภาพอากาศ

หมายเหตุ: กรณีศึกษานี้เป็นเพียงตัวอย่างหนึ่งของการประยุกต์ใช้ Engine ที่เราพัฒนาขึ้น ระบบสามารถใช้กับข้อมูลตารางทั่วไปในสาขาอื่นๆ ได้เช่นกัน เช่น:

- การติดตามสัญญาณชีพของผู้ป่วย (อุณหภูมิร่างกาย, ชีพจร, ความดันโลหิต)
- การตรวจสอบคุณภาพผลิตภัณฑ์ในโรงงาน (ขนาด, น้ำหนัก, อุณหภูมิ)
- การวิเคราะห์ข้อมูลการเงิน (ราคา, ปริมาณการซื้อขาย, ตัวชี้วัดทางเทคนิค)

5.2 ตัวอย่างกฎ

กฎที่ 1 - การจำแนกแบบง่าย:

```
air_quality(PM2_5, ' ') :- PM2_5 < 25.
air_quality(PM2_5, '    ') :- PM2_5 >= 25, PM2_5 =< 50.
air_quality(PM2_5, ' ') :- PM2_5 > 50.
```

กฎที่ 2 - Multi-Variable พร้อม Chain Rules:

```
poor_pm(PM2_5) :- PM2_5 > 50.
high_temp_for_monitoring(Temperature) :- Temperature > 25.

label_overall_condition(PM2_5, Temperature, '     ') :-
    poor_pm(PM2_5),
    high_temp_for_monitoring(Temperature).
```

5.3 ผลลัพธ์

ป้ายกำกับ	จำนวน	เปอร์เซ็นต์
ปานกลาง	37	51.4%
แย่	35	48.6%
ดี	0	0%

ตารางที่ 1: การกระจายตัวของป้ายกำกับคุณภาพอากาศ PM2.5

ระบบระบุช่วงเวลาที่มีคุณภาพอากาศแย่ (PM2.5 > 50) และปานกลางได้สำเร็จ โดยไม่มีการอ่านค่า “ดี” ในช่วงมลพิษนี้

5.4 ข้อสังเกตสำคัญ

กฎเหล่านี้เป็น **Point-based Rules** คือพิจารณาเฉพาะค่า ณ เวลานั้นๆ หากต้องการกฎที่ซับซ้อนขึ้น เช่น:

- “ถ้า PM2.5 เพิ่มขึ้น 20 หน่วยภายใน 2 ชั่วโมง ให้เตือนว่ามลพิษเพิ่มขึ้นอย่างรวดเร็ว”

Engine ปัจจุบัน**ยังไม่รองรับ**เพราะต้องการข้อมูลย้อนหลัง ซึ่งเป็นส่วนหนึ่งของงานพัฒนาในอนาคต

6 คุณสมบัติหลัก

6.1 การจัดเก็บกฎแบบ Timestamped

กฎที่สร้างขึ้นทั้งหมดจะถูกบันทึกพร้อม Timestamp:

`generated_rules_20251205_011531.pl`

วิธีนี้ช่วยเก็บประวัติและเปรียบเทียบชุดกฎต่างๆ ได้

6.2 การรองรับ Multi-Label

ระบบสามารถติดหลายป้ายกำกับพร้อมกันได้:

```
34: " ;        "
```

6.3 การติดตาม Metadata

CSV ที่ผลลัพธ์จะมี:

- คอลัมน์ข้อมูลเดิม
- `auto_label`: ป้ายกำกับที่ถูกติด
- `rules_file`: ชื่อไฟล์กฎที่ถูกใช้

6.4 Configuration-Driven Design

ผู้ใช้สามารถปรับใช้กับข้อมูลชุดใหม่ได้โดยแก้ไขเฉพาะไฟล์ JSON:

- กำหนด CSV columns และ Prolog variable mapping
- ระบุ path สำหรับไฟล์ input/output
- เลือก single-label หรือ multi-label mode
- กำหนด prompt template เป็นภาษาไทยหรืออังกฤษ

ไม่ต้องเปลี่ยนโค้ด Python เลย - เพียงแค่สร้าง use case ใหม่ในโฟลเดอร์ `KB/`

7 ข้อดี

1. **ใช้งานง่าย**: ผู้เชี่ยวชาญในสาขาต่างๆ สามารถเขียนกฎด้วยภาษาธรรมชาติได้
2. **อธิบายได้ (Explainable)**: กฎ Prolog อ่านและตรวจสอบได้โดยมนุษย์
3. **สม่ำเสมอ (Consistent)**: กฎเดียวกันถูกใช้กับข้อมูลทั้งหมดอย่างเท่าเทียม
4. **ยืดหยุ่น (Flexible)**: รองรับตรรกะที่ซับซ้อนผ่าน Chain Rules
5. **ตรวจสอบย้อนกลับได้ (Traceable)**: กฎถูกบันทึกพร้อม Timestamp และ Metadata
6. **แสดงผลกราฟ (Visual)**: สร้างกราฟอัตโนมัติสำหรับการตรวจสอบผลลัพธ์
7. **Generic และใช้ซ้ำได้**: Engine รองรับข้อมูลตารางทั่วไป ปรับใช้กับ Domain ใหม่ได้ง่าย

8 ข้อจำกัดและงานในอนาคต

8.1 ข้อจำกัดปัจจุบัน

1. **ขึ้นอยู่กับคุณภาพของ LLM**: การแปลภาษาธรรมชาติเป็น Prolog ขึ้นอยู่กับความสามารถของ Gemini API ซึ่งอาจสร้างกฎที่ไม่ถูกต้องในบางกรณี

2. **Singleton Variable Warnings**: หากกฎไม่ใช้ตัวแปรบางตัว Prolog จะแสดงคำเตือน ซึ่งต้องแก้ไขด้วยการใช้ underscore (_)

3. **จำกัดรูปแบบข้อมูล**: ปัจจุบันรองรับเฉพาะข้อมูลตาราง (Tabular Data) ในรูปแบบ CSV เท่านั้น

4. **Point-based Inference เท่านั้น - ข้อจำกัดสำคัญของ Engine**:

   Engine ที่เราพัฒนาขึ้นทำงานแบบ *point-based inference* คือพิจารณาเฉพาะค่าของแต่ละแถวข้อมูล ณ จุดนั้นๆ เท่านั้น **ไม่สามารถคำนวณย้อนหลังหรือเปรียบเทียบกับจุดข้อมูลก่อนหน้าได้**

   ตัวอย่างกฎที่ยังไม่รองรับ:

   - “ถ้าอุณหภูมิลดลงมากกว่า 3 องศาภายใน 3 ชั่วโมง ให้ติดป้ายว่า ‘ลดลงอย่างรวดเร็ว’”
   - “ถ้า PM2.5 เพิ่มขึ้น 20% จากค่าเฉลี่ยของ 5 ชั่วโมงที่ผ่านมา”
   - “ถ้าแนวโน้มเป็นขาขึ้นต่อเนื่อง 4 ช่วงเวลาติดต่อกัน”

   กฎเหล่านี้ต้องการ *temporal context* หรือการเข้าถึงข้อมูลก่อนหน้า ซึ่ง Engine ปัจจุบันไม่รองรับเพราะออกแบบมาให้ประมวลผลแต่ละจุดอิสระจากกัน (Stateless row-by-row processing)

   เหตุผลทางเทคนิค:

   - Prolog facts ถูก assert และ retract ในแต่ละ iteration ไม่มีการเก็บประวัติ
   - Query string สร้างจากค่าของแถวปัจจุบันเท่านั้น
   - ไม่มี sliding window หรือ temporal aggregation mechanism

5. **ไม่มีการตรวจจับความขัดแย้งของกฎ**: หากมีกฎที่ขัดแย้งกัน ระบบจะรายงานทุกป้ายกำกับที่ตรงเงื่อนไข (ใน multi-label mode) หรือใช้กฎแรกที่ตรงเงื่อนไข (ใน single-label mode)

8.2 การพัฒนาในอนาคต

1. **รองรับ Temporal Rules และ Historical Context**:

   - เพิ่ม Sliding Window Mechanism สำหรับเข้าถึงข้อมูลก่อนหน้า
   - รองรับการคำนวณ Trend Analysis (ขาขึ้น, ขาลง, อัตราการเปลี่ยนแปลง)
   - เพิ่ม Temporal Predicates เช่น `decreased_by(Variable, Amount, TimeWindow)`
   - เก็บ Historical Facts ใน Prolog Knowledge Base แทนการ retract ทุกครั้ง

2. **รองรับรูปแบบข้อมูลเพิ่มเติม**: JSON, ฐานข้อมูล (SQL/NoSQL), Time-series databases

3. **การปรับแต่งกฎแบบโต้ตอบ (Interactive Rule Refinement)**:
    - แสดงตัวอย่างผลลัพธ์แบบ Real-time
    - เสนอแนะการแก้ไขกฎที่อาจมีปัญหา
    - Validation feedback loop
4. **การผสานกับ Active Learning**: ใช้ผลลัพธ์จาก Auto-labeling เป็น Initial labels สำหรับโมเดล ML
5. **การตรวจจับและแก้ไขความขัดแย้งของกฎ**:
    - วิเคราะห์กฎทั้งหมดเพื่อหาความขัดแย้ง
    - เสนอแนะลำดับความสำคัญของกฎ (Rule Priority)
6. **Export กฎไปรูปแบบอื่น**: Decision Trees, Rule-based Neural Networks, PMML
7. **การเพิ่มประสิทธิภาพ**:
    - Parallel processing สำหรับข้อมูลขนาดใหญ่
    - Caching mechanism สำหรับ Helper Predicates
    - Incremental labeling (label เฉพาะข้อมูลใหม่)

9 สรุป

ระบบนี้แสดงให้เห็นถึงประสิทธิผลของการผสมผสาน Symbolic AI (Prolog) กับ LLMs สมัยใหม่สำหรับการติดป้ายกำกับข้อมูลอัตโนมัติ โดยการให้ผู้เชี่ยวชาญแสดงกฎด้วยภาษาธรรมชาติ ในขณะที่รักษาความเข้มงวดทางตรรกะผ่าน Prolog ระบบนี้เชื่อมช่องว่างระหว่างความสะดวกในการใช้งานกับการให้เหตุผลเชิงรูปแบบ

9.1 Auto-Labeling Engine ที่พัฒนาขึ้น

เราได้พัฒนา Auto-Labeling Engine แบบใหม่ที่มีคุณสมบัติสำคัญดังนี้:

- **Generic และ Reusable**: ออกแบบมาสำหรับข้อมูลตารางทั่วไป ไม่ผูกติดกับ domain ใดโดยเฉพาะ
- **Configuration-Driven**: ปรับใช้กับ use case ใหม่ได้โดยแค่แก้ไขไฟล์ JSON
- **Explainable AI**: กฎ Prolog สามารถอ่านและตรวจสอบได้โดยมนุษย์
- **Point-based Inference**: ประมวลผลแต่ละแถวอิสระจากกัน เหมาะสำหรับกฎที่พิจารณาค่า ณ ขณะนั้น
- **Chain Rules Support**: รองรับการให้เหตุผลแบบลำดับชั้นผ่าน Helper Predicates

สถาปัตยกรรมแบบเปิด (Open Architecture) ที่ใช้การตั้งค่า JSON ทำให้ระบบสามารถปรับใช้กับหลากหลายสาขาได้ เช่น:

- การตรวจสอบคุณภาพอากาศและสิ่งแวดล้อม

- การติดตามสุขภาพและอาการทางการแพทย์
- การตรวจสอบการปฏิบัติตามกฎระเบียบทางการเงิน
- การควบคุมคุณภาพในอุตสาหกรรม
- การวิเคราะห์ข้อมูลทางวิทยาศาสตร์

9.2 ข้อจำกัดที่ต้องคำนึงถึง

ผู้ใช้ควรทราบว่า Engine ปัจจุบัน **ไม่รองรับกฎที่ต้องการข้อมูลย้อนหลังหรือการเปรียบเทียบกับจุดข้อมูลก่อนหน้า** เช่น กฎที่เกี่ยวกับ:

- การเปลี่ยนแปลงในช่วงเวลา (“ลดลง X หน่วยภายใน Y ชั่วโมง”)
- การคำนวณค่าเฉลี่ยเคลื่อนที่ (Moving Average)
- การวิเคราะห์แนวโน้ม (Trend Analysis)

ข้อจำกัดนี้เกิดจากการออกแบบ Point-based Inference ที่ประมวลผลแต่ละแถวอิสระจากกัน การพัฒนาในอนาคตจะเพิ่ม Temporal Context Support เพื่อรองรับกฎประเภทนี้

9.3 ผลงานหลัก

งานวิจัยนี้มีส่วนสนับสนุนหลักดังนี้:

1. **Auto-Labeling Engine ใหม่**: ระบบ Generic สำหรับข้อมูลตารางที่ใช้งานง่ายและอธิบายได้
2. **การผสานระหว่าง LLM และ Symbolic AI**: เชื่อม Natural Language Interface เข้ากับ Logical Reasoning
3. **Configuration-Driven Architecture**: ลดความซับซ้อนในการปรับใช้กับ domain ใหม่
4. **Point-based Inference Methodology**: วิธีการที่ชัดเจนและมีประสิทธิภาพสำหรับการติดป้ายกำกับแบบ row-by-row

กิตติกรรมประกาศ

งานวิจัยนี้ใช้ Google’s Gemini API สำหรับการประมวลผลภาษาธรรมชาติ และ PySwip สำหรับการเชื่อมต่อกับ Prolog

References

[1] PySwip: Python-SWI-Prolog bridge. `https://github.com/yuce/pyswip`

[2] Google Gemini API Documentation. `https://ai.google.dev/`

[3] Bratko, I. (2011). *Prolog Programming for Artificial Intelligence*. Addison-Wesley.

[4] Ratner, A., et al. (2017). Snorkel: Rapid training data creation with weak supervision. *VLDB Endowment*, 11(3), 269-282.

[5] Gunning, D., et al. (2019). XAI—Explainable artificial intelligence. *Science Robotics*, 4(37).