

## □ 연구 개요

### ○ 배경

- 독성 예측(Toxicity Prediction)은 신약 개발, 화학물질 안전성 평가, 환경 독성 분석 등 다양한 분야에서 필수적인 기술로 발전
- 기존의 독성 분석은 실험 기반의 생물학적 평가나 QSAR(Qualitative Structure - Activity Relationship) 모델링에 의존
- 실험비용과 시간 소모가 크고, 데이터 희소성 및 인간 해석 의존도가 높다는 한계 존재.
- 이에 따라 생성형 AI와 대규모 언어모델(LLM)을 활용하여 화학 구조(SMILES)·세포 정보·생물학적 assay 문맥을 동시에 이해하고, 자연어 기반 독성 추론(reasoning)을 수행할 수 있는 지능형 독성 예측 시스템에 대한 수요가 급증
- 특히, 최근 초거대 언어모델이 논리적 Chain of Thought(COT) 추론 능력을 보임에 따라, 단순한 분류(classification)를 넘어 추론 과정(reasoning path)을 해석 가능하게 제시하는 Toxicity AI 모델 개발의 가능성 확대

### ○ 관련 연구

- Park J. et al (2025)는 LLM을 활용해 화학 구조, 생물경로(biological pathways), 유전자 온톨로지 정보 등을 통합해 독성 예측 및 설명 가능성을 시도한 모델을 제시 결과적으로 구조 정보나 독성 메커니즘이 복잡한 경우, LLM 기반 접근은 여전히 한계를 가지는 것을 시사
- Kattamreddy A.R. et al (2025)는 LLM을 독성학 위험평가에 적용 가능성을 제안 하였으나 매우 긴 문서 처리 및 도메인 깊이를 요구하는 규제독성 평가 보고서 등에서 LLM의 성능 저하 한계가 있음을 제시. LLM이 독성 메커니즘을 이해하거나 “왜 이 화합물이 독성을 나타내는가”에 대한 설명을 제공하는 데 한계점을 제시
- Caldas Ramos M., (2025) 화학 분야에서 LLM 및 자율 에이전트의 응용 가능성을 분석하고 그 결과 해석 가능성이 아직 성숙하지 않다는 점을 “key challenges”로 지적

### ○ 기존 연구의 한계점 및 개선 사항

- 기존 LLM은 화학 구조 세포 맥락 독성 메커니즘을 단순 결합만 가능하며 왜 독성을

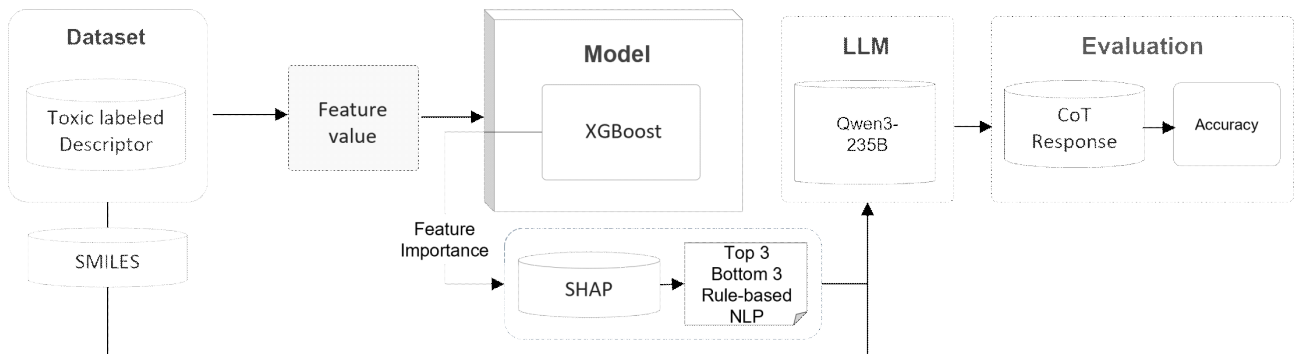
보이는가에 대한 논리적 추론 경로를 제시하지 못함 따라서 단순 독성 분류가 아닌  
인과적 reasoning 및 자연어 근거 생성이 가능한 모델을 구현

## ○ 연구 목표

- Chain of Thought 결과를 지원하는 모델로써, 한국어 Reasoning 가능할 뿐만 아니라, 가중치 최적화를 통한 Toxicity 추론에 최적화 모델 구현
- 객관적인 성능평가 지표를 통한 모델의 유효성 평가를 통해 Toxicity AI 프로토타입 모델 개발
- MMLU Accuracy (chain-of-thought, 0-shot) 수치 90%이상

## □ 주요 설계

### ○ Architecture 구성도



## ○ 수도 코드 (Pesudo code)

---

Input:

- D: Toxicity Source Data (SMILES, descriptors, assay context, label)
- $N_{\text{iter}}$ : Number of Prediction Iterations
- K: Learning Steps per Iteration

Output:

- TC: ADMET Toxicity Converter (self-evolving toxicity prediction system)
- 

```
function ToxicityConverter(D, N_iter, K):
```

```
    # 1. Initialization
```

```
    Initialize Predictive_Model_Set M = {QSAR, GNN, Transformer, Ensemble}
```

```
    Initialize Feature_Representation_Bank F = {}
```

```
    Initialize Evaluation_Metrics E = {AUC, F1, RMSE}
```

```
    # 2. Iterative Learning Loop
```

```
    for iter in range(1, N_iter + 1):
```

```
        # 2-1. Multi-model learning
```

```
        for model in M:
```

```
            model.train(D)
```

```
            predictions = model.predict(D)
```

```
            E[model] = Evaluate(predictions, D.labels)
```

```
        # 2-2. Feature interpretation and importance analysis
```

```
        for model in M:
```

```
            importance_scores = ComputeFeatureImportance(model)
```

```
            Update(F, importance_scores)
```

```
        # 2-3. Self-evolution via adaptive feature representation
```

```
        if iter > 1:
```

---

---

```

    ΔF = Compare(F[iter], F[iter-1])
    if SignificantChange(ΔF):
        FineTune(M, guidance=F)
        AdjustLearningRate(M, based_on=ΔF)

# 2-4. Meta-policy optimization
meta_reward = Aggregate(E, weights=FeatureWeights(F))
PolicyUpdate(M, reward=meta_reward)

# 2-5. Convergence check
if Converged(E, threshold=ε):
    break

# 3. Synthesis and conversion output
TC = SynthesizeConverter(M, F)
return TC

```

---

## □ 기술 개발 결과

### ○ 주요 성능 지표(Performance Metrics)

- 개발 목표: MMLU(Massive Multitask Language Understanding) 기준 정답 정확도 90% 이상 달성
- 개발 결과: 최종 정답 정확도 95% 달성, 목표 대비 +5%p 초과 달성
- 검증 방식:
  - MMLU 평가 체계에 준하여 도메인 지식 기반 문항을 구성
  - 모델이 각 문항에 대해 선택지 중 정확한 답을 고르는 방식으로 테스트 수행
  - 독성 여부 분류 등 실제 도메인 문제를 반영한 문항 활용
- MMLU 형식 기반 테스트 예시

You are a cheminformatics/toxicology expert specializing in molecular toxicity prediction. Please answer the user's question.  
The user will provide you four SMILES with question.

Your answer should be only one of A, B, C, or D.

Which of the following SMILES is mostly toxic(non-toxic)?

Choices)

A) COC(=O)c1cccnc1

B) Oc1c(Cl)c(Cl)cc(Cl)c1Cl

C) [O-]P(=O)(CCCCCCCCCCCCCCCC)OC1CC[N+](C)(C)CC1

D) N#CCCOCCOCCC#N

- 종합 결과:

- 본 모델은 MMLU 기반 테스트에서 목표치를 상회하는 95%의 정확도를 달성하였으며, 독성 판단과 같은 전문 도메인 문제에서도 안정적인 선택 능력을 확보함

## ○ 소스코드 및 주석

```
from __future__ import annotations

import argparse
import asyncio
import json
import logging
import os
import random
import re
import sys

from typing import Any, Iterable, List, Mapping, Tuple, Optional

from tqdm import tqdm
from openai import AsyncOpenAI

ANSWER_TAG_RE = re.compile(r"<answer>(.*?)</answer>", flags=re.IGNORECASE | re.DOTALL)

def read_json(path: str) -> List[Mapping[str, Any]]:
    with open(path, "r", encoding="utf-8") as f:
        return json.load(f)

def write_json(obj: Any, path: str) -> None:
    os.makedirs(os.path.dirname(path) or ".", exist_ok=True)
```

```
withopen(path,"w",encoding="utf-8")asf:
    json.dump(obj,f,ensure_ascii=False,indent=2)
```

```
classModelConfig:
```

```
    def__init__(self,api_key:str,base_url:Optional[str],model:str,
                temperature:float=0.6,top_p:float=0.95)->None:
        self.api_key=api_key
        self.base_url=base_url
        self.model=model
        self.temperature=temperature
        self.top_p=top_p
```

```
@classmethod
```

```
deffrom_env(cls)->"ModelConfig":
```

```
    api_key=os.getenv("OPENAI_API_KEY")
```

```
    ifnotapi_key:
```

```
        raiseEnvironmentError("환경변수 OPENAI_API_KEY 가 필요합니다.")
```

```
    base_url=os.getenv("OPENAI_BASE_URL")orNone # 선택
```

```
    model=os.getenv("MODEL_NAME","gpt-4o-mini") # 기본 공개 모델명 예시
```

```
    try:
```

```
        temperature=float(os.getenv("OPENAI_TEMPERATURE","0.6"))
```

```
        top_p=float(os.getenv("OPENAI_TOP_P","0.95"))
```

```
    exceptValueError:
```

```
        temperature,top_p=0.6,0.95
```

```
    returncls(api_key=api_key,base_url=base_url,model=model,
```

```
              temperature=temperature,top_p=top_p)
```

```
defbuild_message(item:Mapping[str,Any])->List[Mapping[str,str]]:
```

```
    system=(
```

```
        "당신은 분자 독성 예측에 특화된 화학정보학/독성학 전문가입니다.\n"
```

```
        "사용자는 독성/비독성에 영향을 많이 끼치는 Feature 3개씩을 제공합니다.\n\n"
```

```
        "입력(사용자가 제공):\n"
```

```
        "- SMILES\n- Cell Type\n- Cell Line\n- Bio Assay Name\n"
```

```
        "- 독성에 끼치는 영향이 큰 상위 3개 RDKit Feature\n"
```

```
        "- 비독성에 끼치는 영향이 큰 상위 3개 RDKit Feature\n"
```

```
        "수행 과업(Tasks):\n"
```

```
        "SMILES 구조 분석\n"
```

```

"- 고리(방향족/지방족), 헤테로원자, 전하 중심, 반응성 모티프, H-결합 공여/수용기 등을\n"
" SMILES에서 직접 관찰 가능한 범위로만 기술.\n\n"
"Cell Type, Cell Line, Assay Name 특징 분석 및 SMILES와 연결\n\n"
"RDKit feature 분석\n"
"- 각 feature가 의미하는 바와 일반적 독성 리스크에 주는 영향 요약.\n"
"- 가능한 경우 Assay 맥락(예: ARE 산화스트레스)과 연결.\n\n"
"종합 판단(최종 결론)\n"
"- (1) SMILES 모티프, (2) Cell line/Cell type + Assay 맥락, (3) RDKit feature를 통합해\n"
" 독성 여부를 이진으로 판단.\n\n"
"출력 규칙:\n"
"- 본문은 영어로 작성.\n"
"- 마지막 줄에 아래 중 하나만 단독 표기:\n"
"<answer>toxic</answer>\n"
"<answer>nontoxic</answer>\n\n"
"tool call을 사용하지 마세요."

```

```
)
```

```

user=(
    f"SMILES: {item.get('SMILES', '')}\n"
    f"Cell Type: {item.get('CELL_LINE', '')}\n"
    f"Cell Line: {item.get('CELL_TYPE', '')}\n"
    f"Bio Assay Name: {item.get('ARE', '')}\n"
    f"TOP3 FEATURE: {item.get('rule_top', [])}\n"
    f"BOTTOM3 FEATURE: {item.get('rule_bottom', [])}\n\n"
    "제공된 정보를 바탕으로 SMILES형태로 된 위 분자의 독성/비독성 여부를 판단하시오."
)

```

```
)
```

```

return[
    {"role":"system","content":system},
    {"role":"user","content":user},
]

```

```

# =====
# 정답 판정
# =====

```

```

defsplit_correct_wrong(
    data:Iterable[Mapping[str,Any]],
    print_counts:bool=True,

```

```
)->Tuple[List[Mapping[str,Any]],List[Mapping[str,Any]],List[Mapping[str,Any]]]:
```

```
    """응답의 <answer> 태그를 파싱하여 정/오/미싱 분류.
```

```
    LABEL 규칙: 0=비독성, 1=독성
```

```
    """
```

```
    correct,wrong,missing=[],[],[]
```

```
    for item in data:
```

```
        label=item.get("LABEL")
```

```
        ans_text=str(item.get("answer",""))
```

```
        matches=[m.strip() for m in ANSWER_TAG_RE.findall(ans_text)]
```

```
        if not matches:
```

```
            missing.append(item)
```

```
            continue
```

```
        pred_nontoxic=any(m=="비독성" or m.lower()=="nontoxic" for m in matches)
```

```
        pred_toxic=any(m=="독성" or m.lower()=="toxic" for m in matches)
```

```
        if label==0 and pred_nontoxic:
```

```
            correct.append(item)
```

```
        elif label==1 and pred_toxic:
```

```
            correct.append(item)
```

```
        else:
```

```
            wrong.append(item)
```

```
    if print_counts:
```

```
        logging.info("correct_cnt: %d",len(correct))
```

```
        logging.info("wrong_cnt: %d",len(wrong))
```

```
        logging.info("missing_cnt: %d",len(missing))
```

```
    return correct,wrong,missing
```

```
# =====
```

```
# LLM 호출 (비동기)
```

```
# =====
```

```
async def chat_once(client: AsyncOpenAI, cfg: ModelConfig, messages: List[Mapping[str, str]],
```



```

        retries:int=3,initial_delay:float=1.0)->str:
    delay=initial_delay
    forattemptinrange(retries):
        try:
            resp=awaitclient.chat.completions.create(
                model=cfg.model,
                messages=messages,
                temperature=cfg.temperature,
                top_p=cfg.top_p,
            )
            returnresp.choices[0].message.contentor""
        exceptException:
            ifattempt==retries-1:
                raise
            awaitasyncio.sleep(delay)
            delay*=2 # 지수 백오프
    return""

asyncdefrun_concurrent(items:List[Mapping[str,Any]],cfg:ModelConfig,concurrency:int=16)->List[str]:
    sem=asyncio.Semaphore(concurrency)
    client=AsyncOpenAI(api_key=cfg.api_key,base_url=cfg.base_url)
    results:List[Optional[str]]= [None]*len(items)

    asyncdefworker(i:int)->None:
        messages=build_message(items[i])
        asyncwithsem:
            out=awaitchat_once(client,cfg,messages)
            results[i]=out

    tasks=[asyncio.create_task(worker(i))foriinrange(len(items))]

    forintqdm(asyncio.as_completed(tasks),total=len(tasks),desc="inference"):
        awaitf

    return[ror""forinresults]

# =====
# 메인

```

```

# =====

def main(argv: Optional[List[str]] = None) -> int:
    parser = argparse.ArgumentParser(description="Public toxicity benchmark runner")

    parser.add_argument("--input", required=True,
                        help="입력 JSON 경로 (벤치마크 데이터)")
    parser.add_argument("--output", required=True,
                        help="추론 및 메트릭을 저장할 JSON 경로")
    parser.add_argument("--rounds", type=int, default=3,
                        help="동일 벤치마크를 반복 실행하여 평균 점수 계산 (기본 3)")
    parser.add_argument("--concurrency", type=int, default=16,
                        help="동시 추론 개수")
    parser.add_argument("--seed", type=int, default=42)
    parser.add_argument("--log_level", default="INFO",
                        choices=["DEBUG", "INFO", "WARNING", "ERROR"])

    args = parser.parse_args(argv)

    logging.basicConfig(
        level=getattr(logging, args.log_level),
        format="[%asctime)s] %(levelname)s: %(message)s",
    )

    random.seed(args.seed)

    cfg = ModelConfig.from_env()

    # 입력 데이터 적재 (단일 JSON 파일 가정)
    dataset = read_json(args.input)
    if not dataset:
        logging.error("입력 데이터가 비어 있습니다.")
        return 2

    N = len(dataset)
    logging.info("Benchmark size: %d", N)

    total_acc_sum = 0.0

```

```

for i in range(1, args.rounds + 1):
    logging.info("Round %d/%d started", i, args.rounds)
    answers = asyncio.run(run_concurrent(dataset, cfg, concurrency=args.concurrency))

    # 결과 병합 및 저장
    for row, ans in zip(dataset, answers):
        row["answer"] = ans

    write_json(dataset, args.output)

    correct, wrong, missing = split_correct_wrong(dataset, print_counts=True)
    acc = len(correct) / max(1, N)
    total_acc_sum += acc
    logging.info("Round %d accuracy: %.4f", i, acc)

avg_acc = total_acc_sum / args.rounds
logging.info("Average accuracy over %d rounds: %.4f", args.rounds, avg_acc)

# 간단히 stdout에도 출력
print(json.dumps({
    "records": N,
    "rounds": args.rounds,
    "avg_accuracy": avg_acc,
}, ensure_ascii=False, indent=2))


return 0

if __name__ == "__main__": # pragma: no cover
    try:
        raise SystemExit(main())
    except KeyboardInterrupt:
        print("Interrupted.", file=sys.stderr)
        raise

```

## □ 소스코드 공개

○ Git-Hub 공개: <https://github.com/KwangSun-Ryu/ADMET-AGI-Toxicity-AI-Prototype-and-Baseline--.git>


**ADMET-AGI-Toxicity-AI-Prototype-and-Baseline--** Public







Watch 0

main 1 Branch 0 Tags



+

<> Code

 **OREJH** [UNIVA] \_\_pycache\_\_ 제거 39eb7d6 · 2 minutes ago 205 Commits

 Generalized ADMET Inference Baseline	[UNIVA] 폴더 구조 변경	7 minutes ago
 Toxicity AI Prototype	[UNIVA] 폴더 구조 변경	5 minutes ago
 .gitignore	[UNIVA] 폴더 구조 변경	5 minutes ago
 README.md	[UNIVA] 폴더 구조 변경	5 minutes ago
 run_vllm.sh	[UNIVA] Toxicity AI 모듈 업데이트 및 README 추가	yesterday
 utils.py	[UNIVA] Toxicity AI 모듈 업데이트 및 README 추가	yesterday

README

## UNIVA 프로젝트 설명서

### 프로젝트 개요

- 목적: 신약 개발 초기 단계(기초 연구-비임상 실험)에서 요구되는 ADMET(흡수·분포·대사·배설·독성) 특성 분석을 자동화하기 위해, 인지-추론 기반 AGI 에이전트 플랫폼을 구축한다. 본 플랫폼은 대규모 독성·약물동태 데이터와 온톨로지 기반 지식을 통합하여, 분자 수준의 ADMET 프로파일을 자율적으로 추론·해석할 수 있는 차세대 AI 시스템을 지향한다.
- 구현 내용: 1차년도 1단계 목표로 **Generalized ADMET Inference** 베이스라인과 **Toxicity AI** 프로토타입 모델을 구축한다. 이를 통해 기존 신약개발 과정에서 반복되는 수작업 기반 독성 예측 및 ADMET 전주기 분석의 단절 문제를 해소하고, 능동적 의사결정이 가능한 self-evolving ADMET AI 에이전트 개발의 초석을 마련한다.

## □ 참고문헌

1. CoTox: Chain of Thought Based Molecular Toxicity Reasoning and Prediction (Park J. et al., 2025, arXiv)
2. The future of large language models in toxicological risk assessment: Opportunities and challenges (Kattamreddy A.R., Chinnam H., 2025)
3. A review of large language models and autonomous agents in chemistry (Caldas Ramos M., Collison C.J., White A.D., 2025)