analysis_cleaned_golden

July 24, 2025

[41]:
```
import json
import re
import requests
import itertools
from collections import defaultdict, Counter
import pandas as pd
from itertools import combinations
import matplotlib.pyplot as plt
```

[42]:
```
with open("data/all_data.json", "r", encoding="utf-8") as f:
    all_data = json.load(f)

with open("data/cleaned_data.json", "r", encoding="utf-8") as f:
    cleaned_data = json.load(f)

with open("data/gold_standard.json", "r", encoding="utf-8") as f:
    golden_standard = json.load(f)
```

[ ]:
```

```

[43]:
```
has_pwc_cat = 0
has_orkg_cat = 0
has_openalex_cat = 0
has_openaire_cat = 0
for paper in all_data:
    if paper['openaire_categories_flat']:
        has_openaire_cat += 1
    if paper['openalex_categories_flat']:
        has_openalex_cat += 1
    if paper['papers_with_code_categories_flat']:
        has_pwc_cat += 1
    if paper['orkg_categories_flat']:
        has_orkg_cat += 1

all_papercount = len(all_data)
print(f"#papers with OpenAlex cat: {has_openalex_cat}")
print(f"#papers with OpenAIRE cat: {has_openaire_cat}")
```

```
print(f"#papers with PwC cat: {has_pwc_cat}")
print(f"#papers with ORKG cat: {has_orkg_cat}")
print(f"#papers with missing OpenAlex cat: {round((1-has_openalex_cat/
 ↪all_papercount)*100, 2)}")
print(f"#papers with missing OpenAIRE cat: {round((1-has_openaire_cat/
 ↪all_papercount)*100, 2)}")
print(f"#papers with missing PwC cat: {round((1-has_pwc_cat/
 ↪all_papercount)*100, 2)}")
print(f"#papers with missing ORKG cat: {round((1-has_orkg_cat/
 ↪all_papercount)*100, 2)}")
```

```
#papers with OpenAlex cat: 120
#papers with OpenAIRE cat: 98
#papers with PwC cat: 92
#papers with ORKG cat: 93
#papers with missing OpenAlex cat: 4.76
#papers with missing OpenAIRE cat: 22.22
#papers with missing PwC cat: 26.98
#papers with missing ORKG cat: 26.19
```

[44]:

```
pwc_cat_cnt = []
orkg_cat_cnt = []
openalex_cat_cnt = []
openaire_cat_cnt = []
for paper in cleaned_data:
    if paper['openaire_categories_flat']:
        openaire_cat_cnt.append(len(paper['openaire_categories_flat']))
    if paper['openalex_categories_flat']:
        openalex_cat_cnt.append(len(paper['openalex_categories_flat']))
    if paper['papers_with_code_categories_flat']:
        pwc_cat_cnt.append(len(paper['papers_with_code_categories_flat']))
    if paper['orkg_categories_flat']:
        orkg_cat_cnt.append(len(paper['orkg_categories_flat']))

print(f"avg OpenAlex cats cleaned: {round(sum(openalex_cat_cnt)/
 ↪len(openalex_cat_cnt), 2)}")
print(f"avg OpenAIRE cats cleaned: {round(sum(openaire_cat_cnt)/
 ↪len(openaire_cat_cnt), 2)}")
print(f"avg with PwC cats cleaned: {round(sum(pwc_cat_cnt)/len(pwc_cat_cnt),␣
 ↪2)}")
print(f"avg ORKG cat cleaned: {round(sum(orkg_cat_cnt)/len(orkg_cat_cnt), 2)}")
```

```
avg OpenAlex cats cleaned: 12.39
avg OpenAIRE cats cleaned: 7.43
avg with PwC cats cleaned: 16.73
avg ORKG cat cleaned: 2.83
```

[45]:

```
pwc_cat_cnt = []
orkg_cat_cnt = []
openalex_cat_cnt = []
openaire_cat_cnt = []
for paper in golden_standard:
    if paper['openaire_categories_flat']:
        openaire_cat_cnt.append(len(paper['openaire_categories_flat']))
    if paper['openalex_categories_flat']:
        openalex_cat_cnt.append(len(paper['openalex_categories_flat']))
    if paper['papers_with_code_categories_flat']:
        pwc_cat_cnt.append(len(paper['papers_with_code_categories_flat']))
    if paper['orkg_categories_flat']:
        orkg_cat_cnt.append(len(paper['orkg_categories_flat']))

print(f"avg OpenAlex cats golden: {round(sum(openalex_cat_cnt)/
 ↪len(openalex_cat_cnt), 2)}")
print(f"avg OpenAIRE cats golden: {round(sum(openaire_cat_cnt)/
 ↪len(openaire_cat_cnt), 2)}")
print(f"avg with PwC cats golden: {round(sum(pwc_cat_cnt)/len(pwc_cat_cnt),␣
 ↪2)}")
print(f"avg ORKG cat golden: {round(sum(orkg_cat_cnt)/len(orkg_cat_cnt), 2)}")
```

```
avg OpenAlex cats golden: 4.84
avg OpenAIRE cats golden: 3.67
avg with PwC cats golden: 4.66
avg ORKG cat golden: 1.93
```

[46]:

```
categories = set([])
orkg_categories = set([])
pwc_categories = set([])
openalex_categories = set([])
openaire_categories = set([])

for paper in cleaned_data:
    for cat in paper['openaire_categories_flat']:
        categories.add(cat)
        openaire_categories.add(cat)
    for cat in paper['openalex_categories_flat']:
        categories.add(cat)
        openalex_categories.add(cat)
    for cat in paper['papers_with_code_categories_flat']:
        categories.add(cat)
        pwc_categories.add(cat)
    for cat in paper['orkg_categories_flat']:
        categories.add(cat)
        orkg_categories.add(cat)
```

```python
print(f"#all categories cleaned: {len(list(categories))}")
print(f"#ORKG categories cleaned: {len(list(orkg_categories))}")
print(f"#PwC categories cleaned: {len(list(pwc_categories))}")
print(f"#OpenAlex categories cleaned: {len(list(openalex_categories))}")
print(f"#OpenAIRE categories cleaned: {len(list(openaire_categories))}")
```

```
#all categories cleaned: 728
#ORKG categories cleaned: 133
#PwC categories cleaned: 198
#OpenAlex categories cleaned: 277
#OpenAIRE categories cleaned: 157
```

[47]:
```python
categories = set([])
orkg_categories = set([])
pwc_categories = set([])
openalex_categories = set([])
openaire_categories = set([])

for paper in golden_standard:
    for cat in paper['openaire_categories_flat']:
        categories.add(cat)
        openaire_categories.add(cat)
    for cat in paper['openalex_categories_flat']:
        categories.add(cat)
        openalex_categories.add(cat)
    for cat in paper['papers_with_code_categories_flat']:
        categories.add(cat)
        pwc_categories.add(cat)
    for cat in paper['orkg_categories_flat']:
        categories.add(cat)
        orkg_categories.add(cat)

print(f"#all categories cleaned: {len(list(categories))}")
print(f"#ORKG categories cleaned: {len(list(orkg_categories))}")
print(f"#PwC categories cleaned: {len(list(pwc_categories))}")
print(f"#OpenAlex categories cleaned: {len(list(openalex_categories))}")
print(f"#OpenAIRE categories cleaned: {len(list(openaire_categories))}")
```

```
#all categories cleaned: 300
#ORKG categories cleaned: 75
#PwC categories cleaned: 119
#OpenAlex categories cleaned: 96
#OpenAIRE categories cleaned: 38
```

[48]:
```python
# Mapping of category to SKGs it's found in
category_to_skgs = defaultdict(set)
```

```
for paper in cleaned_data:
    skg_cats = {
        "orkg": set(paper["orkg_categories_flat"]),
        "pwc": set(paper["papers_with_code_categories_flat"]),
        "openalex": set(paper["openalex_categories_flat"]),
        "openaire": set(paper["openaire_categories_flat"]),
    }

    for skg, cats in skg_cats.items():
        for cat in cats:
            category_to_skgs[cat].add(skg)

# Count how many categories appear in 1, 2, 3, or 4 SKGs
agreement_counter = Counter()
for cat, skgs in category_to_skgs.items():
    agreement_counter[len(skgs)] += 1

# Total unique categories
total_unique_cats = len(category_to_skgs)

print(f"\nTotal unique categories: {total_unique_cats}")
print("\nAgreement levels:")
for k in range(1, 5):
    print(f"Categories appearing in {k} SKGs: {agreement_counter[k]}")
```

```
Total unique categories: 728

Agreement levels:
Categories appearing in 1 SKGs: 695
Categories appearing in 2 SKGs: 29
Categories appearing in 3 SKGs: 4
Categories appearing in 4 SKGs: 0
```

[49]:
```
# Mapping of category to SKGs it's found in
category_to_skgs = defaultdict(set)

for paper in golden_standard:
    skg_cats = {
        "orkg": set(paper["orkg_categories_flat"]),
        "pwc": set(paper["papers_with_code_categories_flat"]),
        "openalex": set(paper["openalex_categories_flat"]),
        "openaire": set(paper["openaire_categories_flat"]),
    }

    for skg, cats in skg_cats.items():
        for cat in cats:
```

```
            category_to_skgs[cat].add(skg)

# Count how many categories appear in 1, 2, 3, or 4 SKGs
agreement_counter = Counter()
for cat, skgs in category_to_skgs.items():
    agreement_counter[len(skgs)] += 1

# Total unique categories
total_unique_cats = len(category_to_skgs)

print(f"\nTotal unique categories: {total_unique_cats}")
print("\nAgreement levels:")
for k in range(1, 5):
    print(f"Categories appearing in {k} SKGs: {agreement_counter[k]}")
```

```
Total unique categories: 300

Agreement levels:
Categories appearing in 1 SKGs: 277
Categories appearing in 2 SKGs: 18
Categories appearing in 3 SKGs: 5
Categories appearing in 4 SKGs: 0
```

[50]:
```
paper_overlap_counter = Counter()

for paper in cleaned_data:
    skg_cats = {
        "orkg": set(paper["orkg_categories_flat"]),
        "pwc": set(paper["papers_with_code_categories_flat"]),
        "openalex": set(paper["openalex_categories_flat"]),
        "openaire": set(paper["openaire_categories_flat"]),
    }

    # Build reverse map: category → list of SKGs
    category_skg_map = {}
    for skg, cats in skg_cats.items():
        for cat in cats:
            category_skg_map.setdefault(cat, set()).add(skg)

    # Count how many categories appear in how many SKGs
    overlap_levels = Counter()
    for skgs in category_skg_map.values():
        overlap_levels[len(skgs)] += 1

    # Add to overall paper-level count
    for level in [2, 3, 4]:
```

```
        if overlap_levels[level] > 0:
            paper_overlap_counter[level] += 1

# Print results
print("Number of papers with at least one overlapping category in:")
print(f"- 2 SKGs: {paper_overlap_counter[2]}")
print(f"- 3 SKGs: {paper_overlap_counter[3]}")
print(f"- 4 SKGs: {paper_overlap_counter[4]}")
```

```
Number of papers with at least one overlapping category in:
- 2 SKGs: 43
- 3 SKGs: 1
- 4 SKGs: 0
```

[51]:

```
paper_overlap_counter = Counter()

for paper in golden_standard:
    skg_cats = {
        "orkg": set(paper["orkg_categories_flat"]),
        "pwc": set(paper["papers_with_code_categories_flat"]),
        "openalex": set(paper["openalex_categories_flat"]),
        "openaire": set(paper["openaire_categories_flat"]),
    }

    # Build reverse map: category → list of SKGs
    category_skg_map = {}
    for skg, cats in skg_cats.items():
        for cat in cats:
            category_skg_map.setdefault(cat, set()).add(skg)

    # Count how many categories appear in how many SKGs
    overlap_levels = Counter()
    for skgs in category_skg_map.values():
        overlap_levels[len(skgs)] += 1

    # Add to overall paper-level count
    for level in [2, 3, 4]:
        if overlap_levels[level] > 0:
            paper_overlap_counter[level] += 1

# Print results
print("Number of papers with at least one overlapping category in:")
print(f"- 2 SKGs: {paper_overlap_counter[2]}")
print(f"- 3 SKGs: {paper_overlap_counter[3]}")
print(f"- 4 SKGs: {paper_overlap_counter[4]}")
```

```
Number of papers with at least one overlapping category in:
- 2 SKGs: 65
```

- 3 SKGs: 2
- 4 SKGs: 0

[52]:
```python
# Dictionary to hold paper titles per overlap level
papers_with_overlap = {
    2: [],
    3: [],
    4: []
}

for paper in cleaned_data:
    skg_cats = {
        "orkg": set(paper["orkg_categories_flat"]),
        "pwc": set(paper["papers_with_code_categories_flat"]),
        "openalex": set(paper["openalex_categories_flat"]),
        "openaire": set(paper["openaire_categories_flat"]),
    }

    # Build reverse map: category → set of SKGs it appears in
    category_skg_map = {}
    for skg, cats in skg_cats.items():
        for cat in cats:
            category_skg_map.setdefault(cat, set()).add(skg)

    # Count categories by their SKG overlap level
    overlap_levels = {k: 0 for k in [2, 3, 4]}
    for skgs in category_skg_map.values():
        if 2 <= len(skgs) <= 4:
            overlap_levels[len(skgs)] += 1

    # Save paper title if there's at least one category for that level
    for level in [2, 3, 4]:
        if overlap_levels[level] > 0:
            papers_with_overlap[level].append(paper["title"])

# Print results
for level in [2, 3, 4]:
    print(f"\nPapers with at least one category in {level} SKGs␣
↪({len(papers_with_overlap[level])} papers):")
    for title in papers_with_overlap[level]:
        print(f"- {title}")
```

```
Papers with at least one category in 2 SKGs (43 papers):
- MiniCPM: Unveiling the Potential of Small Language Models with Scalable
  Training Strategies
- Enhancing text-based knowledge graph completion with zero-shot large language
models: A focus on semantic enhancement
```

- COCONut: Modernizing COCO Segmentation
- Annotation Errors and NER: A Study with OntoNotes 5.0
- Understanding and Tackling Label Errors in Individual-Level Nature
  Language Understanding
- Human Evaluation of Procedural Knowledge Graph Extraction from Text with Large Language Models
- TinyLlama: An Open-Source Small Language Model
- Self-Contrast: Better Reflection Through Inconsistent Solving Perspectives
- Search-in-the-Chain: Interactively Enhancing Large Language Models with Search for Knowledge-intensive Tasks
- The Power of Noise: Redefining Retrieval for RAG Systems
- Retrieval meets Long Context Large Language Models
- Corrective Retrieval Augmented Generation
- UniMS-RAG: A Unified Multi-source Retrieval-Augmented Generation for Personalized Dialogue Systems
- FABULA: Intelligence Report Generation Using Retrieval-Augmented Narrative Construction
- Reasoning on Graphs: Faithful and Interpretable Large Language Model Reasoning
- G-Retriever: Retrieval-Augmented Generation for Textual Graph
  Understanding and Question Answering
- Generating Benchmarks for Factuality Evaluation of Language Models
- PURPLE: Making a Large Language Model a Better SQL Writer
- ARES: An Automated Evaluation Framework for Retrieval-Augmented Generation Systems
- Automating psychological hypothesis generation with AI: when large language models meet causal graph
- Compact Language Models via Pruning and Knowledge Distillation
- CYCLE: Learning to Self-Refine the Code Generation
- Verification and Refinement of Natural Language Explanations through LLM-Symbolic Theorem Proving
- GWQ: Gradient-Aware Weight Quantization for Large Language Models
- SPIQA: A Dataset for Multimodal Question Answering on Scientific Papers
- Artificial intelligence for literature reviews: opportunities and challenges
- SciDQA: A Deep Reading Comprehension Dataset over Scientific Papers
- Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling
- UL2: Unifying Language Learning Paradigms
- Have LLMs Advanced Enough? A Challenging Problem Solving Benchmark For Large Language Models
- ChemCrow: Augmenting large-language models with chemistry tools
- StarCoder: may the source be with you!
- WizardLM: Empowering Large Language Models to Follow Complex Instructions
- WizardMath: Empowering Mathematical Reasoning for Large Language Models via Reinforced Evol-Instruct
- Jais and Jais-chat: Arabic-Centric Foundation and Instruction-Tuned Open Generative Large Language Models
- Orca: Progressive Learning from Complex Explanation Traces of GPT-4
- MEGA: Multilingual Evaluation of Generative AI

```
- ChatGPT Beyond English: Towards a Comprehensive Evaluation of Large Language
Models in Multilingual Learning
- M3Exam: A Multilingual, Multimodal, Multilevel Benchmark for Examining Large
Language Models
- Measuring Massive Multitask Chinese Understanding
- Evaluating language models for mathematics through interactions
- How well do Large Language Models perform in Arithmetic tasks?
- MME: A Comprehensive Evaluation Benchmark for Multimodal Large Language Models

Papers with at least one category in 3 SKGs (1 papers):
- Annotation Errors and NER: A Study with OntoNotes 5.0

Papers with at least one category in 4 SKGs (0 papers):
```

[53]:

```
# Dictionary to hold paper titles per overlap level
papers_with_overlap = {
    2: [],
    3: [],
    4: []
}

for paper in golden_standard:
    skg_cats = {
        "orkg": set(paper["orkg_categories_flat"]),
        "pwc": set(paper["papers_with_code_categories_flat"]),
        "openalex": set(paper["openalex_categories_flat"]),
        "openaire": set(paper["openaire_categories_flat"]),
    }

    # Build reverse map: category → set of SKGs it appears in
    category_skg_map = {}
    for skg, cats in skg_cats.items():
        for cat in cats:
            category_skg_map.setdefault(cat, set()).add(skg)

    # Count categories by their SKG overlap level
    overlap_levels = {k: 0 for k in [2, 3, 4]}
    for skgs in category_skg_map.values():
        if 2 <= len(skgs) <= 4:
            overlap_levels[len(skgs)] += 1

    # Save paper title if there's at least one category for that level
    for level in [2, 3, 4]:
        if overlap_levels[level] > 0:
            papers_with_overlap[level].append(paper["title"])

# Print results
```

```python
for level in [2, 3, 4]:
    print(f"\nPapers with at least one category in {level} SKGs␣
↪({len(papers_with_overlap[level])} papers):")
    for title in papers_with_overlap[level]:
        print(f"- {title}")
```

```
Papers with at least one category in 2 SKGs (65 papers):
- MiniCPM: Unveiling the Potential of Small Language Models with Scalable
Training Strategies
- OLMo: Accelerating the Science of Language Models
- Enhancing text-based knowledge graph completion with zero-shot large language
models: A focus on semantic enhancement
- COCONut: Modernizing COCO Segmentation
- Annotation Errors and NER: A Study with OntoNotes 5.0
- Understanding and Tackling Label Errors in Individual-Level Nature Language
Understanding
- Human Evaluation of Procedural Knowledge Graph Extraction from Text with Large
Language Models
- Structure Guided Large Language Model for SQL Generation
- TinyLlama: An Open-Source Small Language Model
- Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone
- Gemini 1.5: Unlocking multimodal understanding across millions of tokens of
context
- Self-Contrast: Better Reflection Through Inconsistent Solving Perspectives
- Self-Refine Instruction-Tuning for Aligning Reasoning in Language Models
- Pride and Prejudice: LLM Amplifies Self-Bias in Self-Refinement
- Search-in-the-Chain: Interactively Enhancing Large Language Models with Search
for Knowledge-intensive Tasks
- The Power of Noise: Redefining Retrieval for RAG Systems
- Retrieval meets Long Context Large Language Models
- RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval
- Corrective Retrieval Augmented Generation
- UniMS-RAG: A Unified Multi-source Retrieval-Augmented Generation for
Personalized Dialogue Systems
- FABULA: Intelligence Report Generation Using Retrieval-Augmented Narrative
Construction
- Reasoning on Graphs: Faithful and Interpretable Large Language Model Reasoning
- G-Retriever: Retrieval-Augmented Generation for Textual Graph Understanding
and Question Answering
- Generating Benchmarks for Factuality Evaluation of Language Models
- BAMBOO: A Comprehensive Benchmark for Evaluating Long Text Modeling Capacities
of Large Language Models
- PURPLE: Making a Large Language Model a Better SQL Writer
- Middleware for LLMs: Tools Are Instrumental for Language Agents in Complex
Environments
- Knowledge-to-SQL: Enhancing SQL Generation with Data Expert LLM
```

- Improving Demonstration Diversity by Human-Free Fusing for Text-to-SQL
- Decomposition for Enhancing Attention: Improving LLM-based Text-to-SQL through Workflow Paradigm
- ARES: An Automated Evaluation Framework for Retrieval-Augmented Generation Systems
- CompassJudger-1: All-in-one Judge Model Helps Model Evaluation and Evolution
- Gemma 2: Improving Open Language Models at a Practical Size
- MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases
- Compact Language Models via Pruning and Knowledge Distillation
- Self-Refinement of Language Models from External Proxy Metrics Feedback
- CYCLE: Learning to Self-Refine the Code Generation
- Verification and Refinement of Natural Language Explanations through LLM-Symbolic Theorem Proving
- MobileQuant: Mobile-friendly Quantization for On-device Language Models
- GWQ: Gradient-Aware Weight Quantization for Large Language Models
- SPIQA: A Dataset for Multimodal Question Answering on Scientific Papers
- Artificial intelligence for literature reviews: opportunities and challenges
- SciDQA: A Deep Reading Comprehension Dataset over Scientific Papers
- Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling
- UL2: Unifying Language Learning Paradigms
- Llama 2: Open Foundation and Fine-Tuned Chat Models
- Have LLMs Advanced Enough? A Challenging Problem Solving Benchmark For Large Language Models
- WizardCoder: Empowering Code Large Language Models with Evol-Instruct
- StarCoder: may the source be with you!
- WizardLM: Empowering Large Language Models to Follow Complex Instructions
- WizardMath: Empowering Mathematical Reasoning for Large Language Models via Reinforced Evol-Instruct
- Jais and Jais-chat: Arabic-Centric Foundation and Instruction-Tuned Open Generative Large Language Models
- Orca: Progressive Learning from Complex Explanation Traces of GPT-4
- CMATH: Can Your Language Model Pass Chinese Elementary School Math Test?
- MEGA: Multilingual Evaluation of Generative AI
- ChatGPT Beyond English: Towards a Comprehensive Evaluation of Large Language Models in Multilingual Learning
- M3Exam: A Multilingual, Multimodal, Multilevel Benchmark for Examining Large Language Models
- Measuring Massive Multitask Chinese Understanding
- Evaluating language models for mathematics through interactions
- How well do Large Language Models perform in Arithmetic tasks?
- StructGPT: A General Framework for Large Language Model to Reason over Structured Data
- MMBench: Is Your Multi-modal Model an All-Around Player?
- MME: A Comprehensive Evaluation Benchmark for Multimodal Large Language Models
- Xiezhi: An Ever-Updating Benchmark for Holistic Domain Knowledge Evaluation
- C-Eval: A Multi-Level Multi-Discipline Chinese Evaluation Suite for Foundation

Models

Papers with at least one category in 3 SKGs (2 papers):
- Annotation Errors and NER: A Study with OntoNotes 5.0
- SPIQA: A Dataset for Multimodal Question Answering on Scientific Papers

Papers with at least one category in 4 SKGs (0 papers):