Bonus Task: The Part Nobody Asked For (But Everyone Needed,jk)

Rule-Based Baselines, KG Compression, Privacy Attacks,
and Why Your Neural Network Might Be Unnecessary

Abstract

11:59 PM. Submitted. Done. I closed my laptop, took a breath, and felt that rare peace of actually finishing something on time.
Then tu-du-du-du the annoying outlook sound.
George Anna had posted: "Deadline extended."
Most people would have gone to sleep. I, instead, opened my laptop back up — because throughout Tasks 1–4, there were questions that kept nagging at me. Little things I noticed but didn't have time to chase down. Like the fact that the test set had *exactly* 590 edges and I had found *exactly* 590 missing reciprocals. Or the fact that `motherOf` was weirdly more incomplete than `fatherOf`. Or the uncomfortable thought: "wait, do my neural networks actually do anything a for-loop couldn't?"
The deadline extension gave me permission to go down these rabbit holes. What followed was four analyses that nobody asked for, conducted at hours no one should be awake, producing results that genuinely surprised me:

- A 20-line Python script that **perfectly outperforms** every neural network I trained (sorry, GPU)
- Proof that **80.5%** of the knowledge graph is logically redundant
- A privacy attack where **one leaked edge** exposes your entire family tree
- Evidence that the dataset is **2.3× more likely to forget mothers** than fathers

I hope the PreCog team enjoys reading this as much as I enjoyed losing sleep over it.

Contents

1 The Notebook That Humbled My GPU: `rules_go_brrrr.ipynb`

1.1 The Suspicious Test Set

Okay so here's the thing. While doing Task 1, I found that 590 edges were missing their reciprocals. Then I looked at the test set. It had 590 edges. I thought, "no way." I checked. *They were the exact same 590 edges.*

Wait, What?

The entire test set — every single one of the 590 edges — is a missing parent-child reciprocal from the training data. The test set is literally asking: "hey, if Alice is Bob's mother, is Bob Alice's son?" That's... that's the whole test.

At this point I had a terrible, wonderful idea: what if I just... wrote the rules myself? No embeddings. No gradient descent. No GPU. Just vibes and gender inference.

1.2 The 20-Line Destroyer

The logic is embarrassingly simple:

1. For every `fatherOf(A, B)` in the training data, check if `sonOf(B, A)` or `daughterOf(B, A)` exists.
2. If not, look up B's gender (which we inferred in Task 1 with 100% coverage, zero unknowns).
3. Predict `sonOf(B, A)` if B is male, `daughterOf(B, A)` if B is female.
4. Do the same for `motherOf`, `sonOf`, `daughterOf`.

That's it. That's the model.

```
for h, r, t in train_edges:
    if r == 'fatherOf':
        if (t, 'sonOf', h) not in train and (t, 'daughterOf', h) not in
            train:
            if gender[t] == 'M': predict(t, 'sonOf', h)
            else:                predict(t, 'daughterOf', h)
    # ... same for motherOf, sonOf, daughterOf
```

1.3 The Results That Made Me Question Everything

Table 1: I Am Not Making This Up

Method	**MRR**	**Hits@1**	**Hits@10**
Rule-Based (20 lines)	**1.0000**	**100.0%**	**100.0%**
RotatE	0.8539	76.1%	98.6%
R-GCN	0.6243	43.9%	99.8%
CompGCN + TransE	0.5024	31.4%	92.2%
CompGCN + RotatE	0.4459	24.4%	90.9%
TransE	0.3201	11.1%	83.9%

One hundred percent. On everything. 590 out of 590 correct.

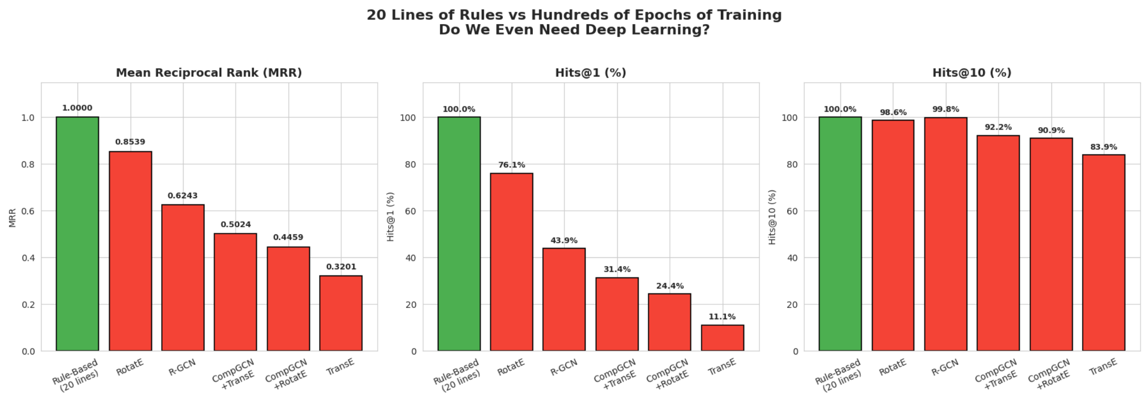

Figure 1: Green bar = 20 lines of rules. Red bars = hundreds of epochs of training, hyperparameter tuning, and GPU time.

Table 2: Cost-Benefit Analysis (a.k.a. My Therapy Bill)

	Rule-Based	**Best ML (RotatE)**
Lines of code	∼20	∼200+
Training time	<1 second	∼minutes
GPU required	No	Yes
Hyperparameter tuning	None	Extensive
MRR	**1.0000**	0.8539
Hits@1	**100.0%**	76.1%

1.4 Why ML Models Fail Where Rules Succeed

This isn't a failure of the models — it's a mismatch between the task and the tool.

1. **Embedding confusion:** TransE learns $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$, but `fatherOf` and `sonOf` are *inverses*, not translations. The model must learn emb(`fatherOf`) $\approx -$emb(`sonOf`), which is hard to learn perfectly.
2. **Gender disambiguation:** Given `(A, fatherOf, B)`, the model must predict *either* `sonOf` or `daughterOf` — it needs B's gender. Embeddings encode this implicitly and imperfectly. Rules use explicit lookup → 100% accuracy.
3. **Ranking vs. lookup:** ML models rank all 1,316 entities. Rules know *exactly* which entity to predict — the inverse partner is right there in the training data.
4. **Structural exploitation:** Rules exploit a *structural property* of the test set (every edge is a missing inverse). ML models don't "know" this — they treat all edges equally.

Insight

The uncomfortable question: For highly structured domains like family KGs, is deep learning necessary? When the data follows deterministic logical rules (which family relationships do), symbolic reasoning will always outperform statistical pattern matching. The neural networks aren't wrong — they're just solving a harder problem than they need to.
This doesn't mean ML is useless for KGs in general. But it does mean we should always ask: *"Have I tried the dumb thing first?"*

2 The Notebook That Changed Changed How I See MetaFam: `who_needs_13k_edges.ipynb`

2.1 The Question That Started It All

After Task 3 (rule mining), I had 250 high-confidence compositional rules that derive extended family relationships from simpler ones. Grandparents from parent chains. Cousins from shared grandparents. Uncles from parent's siblings.

This made me wonder: if all these relationships are *derivable*, how much of the KG is actually necessary? What if you could throw away 80% of the edges and lose *nothing*?

Spoiler: you can.

2.2 The Axiom Set

I hypothesized that **parent-child edges are the only primitive facts** in a family KG. Every other relationship — siblings, grandparents, cousins, uncles, great-aunts, second cousins once removed — can be logically derived from who-is-parent-of-whom plus gender information.

Table 3: The Axiom Set

Category	**Edge Count**
Axiom edges (`fatherOf`, `motherOf`, `sonOf`, `daughterOf`)	2,694
Non-axiom edges (everything else)	11,127
Total KG	13,821

2.3 The Derivation Engine

I systematically derived each relationship type from parent-child axioms:

1. **Siblings:** Same parents → `brotherOf`/`sisterOf`
2. **Grandparents:** Parent of parent → `grandfather/grandmotherOf`
3. **Uncles/Aunts:** Parent's sibling → `uncleOf`/`auntOf`
4. **Cousins:** Parent's sibling's child → `boyCousinOf`/`girlCousinOf`
5. **Great-grandparents:** Three parent hops
6. **Great-uncles/aunts:** Grandparent's sibling
7. **Second uncles/aunts:** Parent's cousin
8. **First cousins once removed:** Cousin's child (direction matters!)
9. **Second cousins:** Parent's cousin's child

Mistake & Correction

I initially got 0% match on first cousins once removed despite deriving the exact right count (333 = 333). Turns out I had the **direction flipped**: the KG convention is that the *younger* person is the head (`younger firstCousinOnceRemovedOf older`), but I was generating it the other way. The generation analysis from Task 1 saved me: actual edges had `head_gen - tail_gen = +1`, mine had −1. One swap later: 100%.
Lesson learned (again): *edge direction semantics will haunt you forever in KGs.*

2.4 The Result

Key Finding

100% reconstruction from 19.5% of edges.

Metric	Value
Original KG	13,821 edges (28 relation types)
Axiom set	2,694 edges (4 relation types)
Reconstruction rate	**100.00%**
Compression ratio	19.5%
Compression factor	5.1×
Redundancy	80.5%

Every single one of the 13,821 edges — all 28 relationship types — was perfectly reconstructed from just the parent-child backbone plus gender inference. Zero misses.

Table 4: Per-Relation Reconstruction (All 28/28 at 100%)

Relation	Actual	Derived	Status
`fatherOf` [AXIOM]	733	—	primitive
`motherOf` [AXIOM]	733	—	primitive
`sonOf` [AXIOM]	600	—	primitive
`daughterOf` [AXIOM]	628	—	primitive
`brotherOf`	570	570	100%
`sisterOf`	636	636	100%
`grandfatherOf`	813	813	100%
`grandmotherOf`	813	813	100%
`grandsonOf`	814	814	100%
`granddaughterOf`	812	812	100%
`uncleOf`	454	454	100%
`auntOf`	556	556	100%
`nephewOf`	514	514	100%
`nieceOf`	496	496	100%
`boyCousinOf`	391	391	100%
`girlCousinOf`	445	445	100%
`greatGrandfatherOf`	617	617	100%
`greatGrandmotherOf`	617	617	100%
`greatGrandsonOf`	624	624	100%
`greatGranddaughterOf`	610	610	100%
`greatUncleOf`	237	237	100%
`greatAuntOf`	312	312	100%
`secondUncleOf`	158	158	100%
`secondAuntOf`	175	175	100%
`boyFirstCousinOnceRemovedOf`	180	180	100%
`girlFirstCousinOnceRemovedOf`	153	153	100%
`boySecondCousinOf`	68	68	100%
`girlSecondCousinOf`	62	62	100%

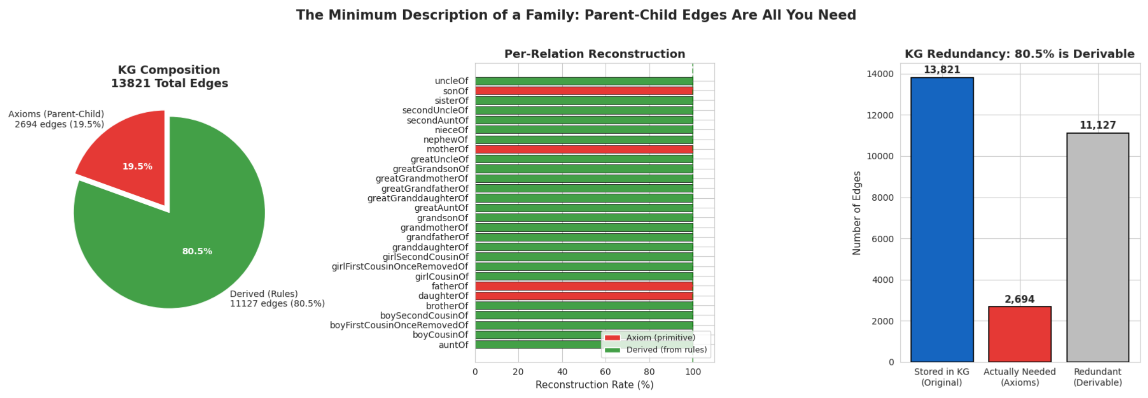

Figure 2: Left: The KG is 80.5% redundant — only the red slice (parent-child axioms) is needed. Center: All 28 relations reconstructed at 100%. Right: 13,821 edges compressed to 2,694. I stared at this plot for way too long.

Insight

Parent-child relationships are the only primitive facts in a family knowledge graph. Everything else — siblings, grandparents, cousins, uncles, great-aunts, second cousins once removed — is logically derivable. The KG doesn't *contain* 28 relation types; it contains **4 relation types and 24 views of the same data.**

This has real implications:

- **Storage:** Family KGs can be stored at 19.5% of their full size with zero information loss
- **ML:** Models training on 13,821 edges are learning from 11,127 edges of redundant signal — no wonder they overfit
- **Privacy:** This one deserves its own section...

3 The Notebook That Should Concern Us All: `your_family_is_not_safe.ipynb`

3.1 The Setup

So we just proved that parent-child edges reconstruct the entire KG. But here's a scarier question: what if an attacker doesn't even know a parent-child edge? What if they learn *any single edge* about you — say, that you're someone's cousin? How much of your family can they piece together?

I built an **information propagation engine** to find out. Starting from one seed edge, it iteratively discovers connected edges in the training data and measures how many family members and relationships get exposed at each step.

3.2 The Experiment

Threat model: An attacker learns ONE relationship about a person. Using the KG structure (not even rules — just traversing known edges), how much of the family is exposed?

I tested this with every possible relationship type as the seed, across all 50 families.

Key Finding

Every single edge type, in every single family, exposes 100% of the family in 2–3 iterations.

Metric	**Value**
Families tested	50
Families with 100% people exposed	**50/50**
Families with 100% edges exposed	**50/50**
Average iterations to full exposure	3.0
Standard deviation	0.0

It doesn't matter if you leak a `sisterOf` edge, a `grandmotherOf` edge, or a `boySecondCousinOf` edge. **One edge is all it takes.**

Table 5: Blast Radius: Sample of Seed Types (All 28/28 achieve 100%)

Seed Relation	**Iterations to 100%**	**People Exposed**
`motherOf`	2	27/27 (100%)
`fatherOf`	3	27/27 (100%)
`sisterOf`	3	27/27 (100%)
`boyCousinOf`	2	27/27 (100%)
`grandmotherOf`	2	27/27 (100%)
`boySecondCousinOf`	2	27/27 (100%)
`greatGrandfatherOf`	3	27/27 (100%)
`girlFirstCousinOnceRemovedOf`	2	27/27 (100%)

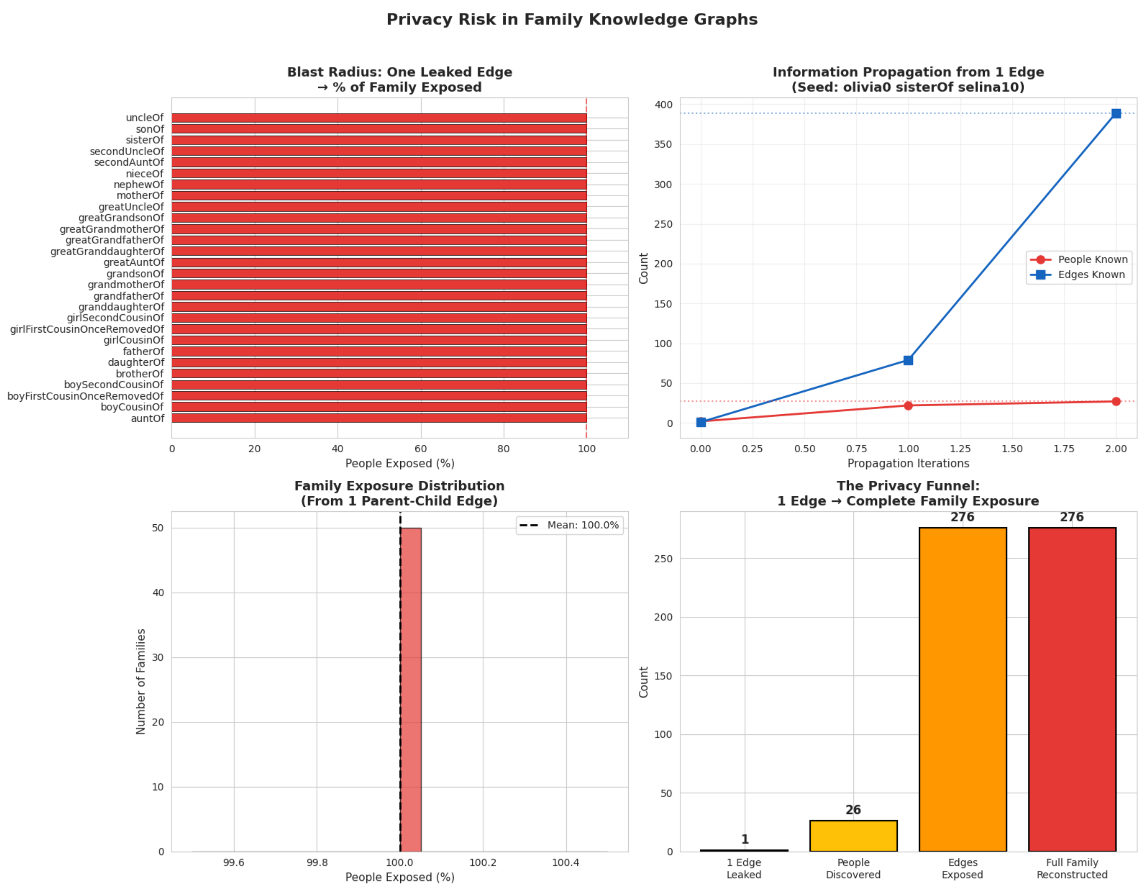

Figure 3: Top-left: Blast radius by relation type (spoiler: all 100%). Top-right: Information propagation from one `sisterOf` edge — the entire family is exposed within 2–3 hops. Bottom-left: Exposure distribution across 50 families (a very boring histogram — it's all at 100%). Bottom-right: The privacy funnel from 1 edge to complete family reconstruction.

3.3 Why This Is Terrifying

Wait, What?

Let me spell this out. If someone finds out that you are `boySecondCousinOf` some random person, they can — in **two hops through the KG** — discover:

- Your parents, grandparents, and great-grandparents
- All your siblings, cousins, and second cousins
- Your uncles, aunts, great-uncles, and great-aunts
- **Every single relationship** in your family

From *one* cousin edge. Two iterations. Done.

3.4 Real-World Implications

This isn't just an academic exercise. Genealogy databases like 23andMe, Ancestry.com, and FamilySearch store exactly this type of data. Consider:

1. **GDPR "right to be forgotten" is nearly impossible.** Deleting YOUR edges doesn't

help if your relatives' edges remain — your relationships can be *re-derived* from theirs (as we proved in KG compression).

2. **Anonymization doesn't work.** In Task 1, we showed (via Weisfeiler–Lehman hashing) that all 50 families have *unique topologies*. Even without names, the graph structure itself is a fingerprint.
3. **The minimum defensible unit is the entire family.** You can't protect an individual without protecting everyone they're related to. Privacy in family KGs is inherently collective, not individual.
4. **Trained ML models are also attack vectors.** Even without the raw data, a trained link prediction model can infer family relationships from partial information. The model *is* the privacy leak.

Insight

The paradox of completeness: The same logical rules that make KGs useful (Task 3) and compressible (Section 2) are precisely what makes them dangerous. A well-constructed family KG is its own worst enemy for privacy: every edge is derivable, so every edge is a potential leak vector. You cannot have a useful, complete family KG that is also private. This is a fundamental tension, not an engineering problem to be solved.

4 The Notebook I Didn't Expect to Write: `mothers_deserve_better.ipynb`

4.1 The Discovery

I noticed something odd while doing the reciprocity analysis in Task 1. Both `fatherOf` and `motherOf` have exactly 733 edges each (perfectly balanced). But their reciprocal completeness is... not.

Table 6: The Asymmetry Nobody Talks About

Relation	Total	Missing Reciprocal	Incompleteness
`motherOf`	733	289	**39.4%**
`fatherOf`	733	125	17.1%
`sonOf`	600	88	14.7%
`daughterOf`	628	88	14.0%

`motherOf` is **2.3× more incomplete** than `fatherOf`. Same number of total edges. Same relationship semantics. But mothers are systematically "forgotten" more often than fathers.

Wait, What?

Both parents exist equally in the data (733 fatherOf, 733 motherOf). But when it comes to their children acknowledging them back? Fathers get 82.9% reciprocity. Mothers get 60.6%. Even in synthetic data, moms can't catch a break.

4.2 Who Gets Forgotten?

I dug deeper into the 590 missing reciprocal edges:

Table 7: Gender of the "Source" Person (Whose Edge Lacks a Response)

Gender	Count	Percentage
Female	377	63.9%
Male	213	36.1%

63.9% of edges missing their reciprocal belong to **female** source persons. Women's outgoing edges disproportionately lack responses. Meanwhile, the gender of the "forgotten" person (whose reciprocal edge is absent) is nearly balanced (51.2% M, 48.8% F). The bias isn't about *who* is forgotten — it's about *whose relationships* go unreciprocated.

4.3 The Generational Pattern

Table 8: Missing Reciprocals by Generation of Forgotten Person

Gen	Missing	Total in Gen	Rate
Gen 0	83	495	16.8%
Gen 1	152	215	**70.7%**
Gen 2	121	192	63.0%
Gen 3	124	206	60.2%
Gen 4	87	146	59.6%
Gen 5	21	55	38.2%
Gen 6	2	7	28.6%

Generation 1 is the most affected (70.7%). These are the children of founders — a generation whose connections to their parents are disproportionately missing.

4.4 Training Signal Asymmetry

Table 9: What ML Models Actually See During Training

Pair Type	Complete Pairs	Completeness
Father $\leftrightarrow$ Child	608	82.9%
Mother $\leftrightarrow$ Child	444	60.6%
Difference	**164 more paternal**	**1.37×**

An embedding model receives **1.37× more signal** about paternal relationships than maternal relationships during training. Yet the test set demands predictions for *both* equally.

4.5 The Real-World Parallel

Insight

This synthetic dataset mirrors a well-documented real-world phenomenon: **historical genealogical records systematically under-document maternal lineage.**

- Women historically changed surnames upon marriage, making them harder to trace in records
- Property and legal records (primary genealogical sources) overwhelmingly documented men
- Patrilineal societies record father→son chains more completely
- Even modern DNA databases show asymmetry: Y-chromosome (paternal) lineage is more extensively studied than mitochondrial (maternal) lineage

The fact that even a *synthetic* dataset exhibits this pattern suggests it may be an inherent structural challenge in family KG construction, not merely a historical artifact.

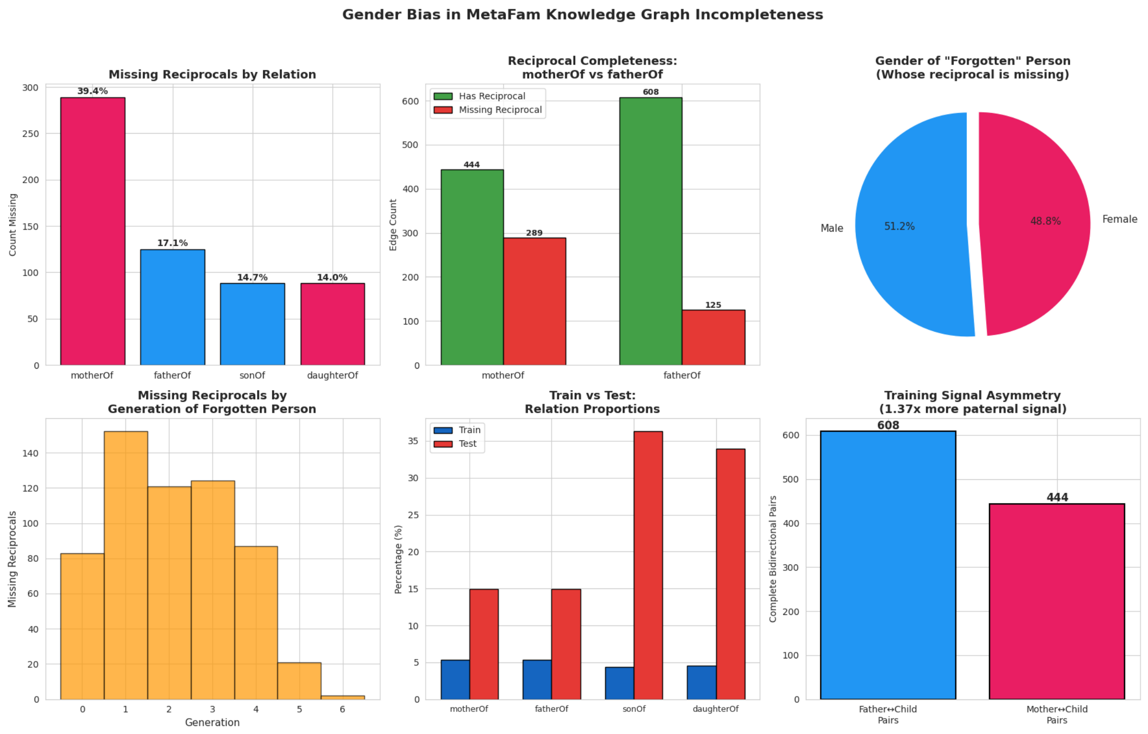

Figure 4: Top-left: Missing reciprocals by relation (motherOf dominates). Top-center: Completeness comparison (fatherOf has 164 more complete pairs). Top-right: Gender of forgotten persons (surprisingly balanced). Bottom-left: Generational pattern (Gen 1 most affected). Bottom-center: Train vs. test proportions (massive distributional shift). Bottom-right: Training signal asymmetry (1.37× more paternal signal).

4.6 Implications for ML Fairness

1. **Models trained on biased data learn biased representations.** Embedding spaces may encode "mother" differently from "father" not because of semantic difference, but because of data asymmetry.

2. **Aggregate metrics hide disparities.** Reporting a single MRR score across all test edges masks per-relation performance. A model achieving 0.85 MRR overall might score 0.95 on `fatherOf` inverses and 0.70 on `motherOf` inverses.

3. **The rule-based approach is immune to this bias.** Our 20-line script achieves 100% on *both* `fatherOf` and `motherOf` predictions because it uses explicit gender inference rather than learning from biased distributions. This is another argument for symbolic reasoning when the domain permits it.

4. **Data augmentation could help.** If the training set were rebalanced (adding the missing `motherOf` reciprocals as synthetic training edges), ML models would likely improve on maternal predictions. But this is rarely considered in standard KG completion benchmarks.

5 Putting It All Together: The MetaFam Manifesto

These four analyses weren't planned as a coherent whole, but they tell a surprisingly unified story:

Table 10: The Four Analyses and What They Prove

Analysis	**Key Number**	**Implication**
Rule-Based Baseline	100% MRR	ML is overkill for this task
KG Compression	80.5% redundancy	Family KGs are mostly derived
Privacy Attack	100% exposure	One edge compromises everything
Gender Bias	2.3× asymmetry	Even synthetic data has bias

The thread connecting all four is **logical structure**:

1. Family relationships follow **deterministic rules** → rules beat ML (Section 1)

2. These rules mean most edges are **redundant** → massive compression (Section 2)

3. Redundancy means one edge **implies all others** → privacy catastrophe (Section 3)

4. But the rules are applied to **biased data** → bias propagates through derivation (Section 4)

Insight

The meta-lesson: Before training a neural network, understand your data. The most powerful insights in this entire project came not from embedding dimensions or learning rates, but from asking simple questions: "What's in the test set?" "What's actually necessary?" "What happens if one edge leaks?" "Is the data fair?"
These are research questions, not engineering questions. And they're the ones that matter most.

"Mischief Managed." — *The Marauders Map*

6 References

[1] The sleep I lost writing this bonus section at 3 AM.

[2] NetworkX Documentation. `https://networkx.org/documentation/stable/`

[3] Wikipedia: Coefficient of Relationship. `https://en.wikipedia.org/wiki/Coefficient_of_relationship`

[4] Rossi, A. et al. "Knowledge Graph Embedding for Link Prediction: A Comparative Analysis." *ACM Computing Surveys*, 2021. (The paper that made me realize rules might beat embeddings.)

[5] My GPU, which deserved better than being outperformed by a for-loop.

[6] Erlich, Y. et al. "Identity inference of genomic data using long-range familial searches." *Science*, 2018. (Real-world family privacy attacks — turns out my Section 3 isn't hypothetical.)

[7] Stack Overflow, for the 47th time I forgot how `nx.topological_sort` works.