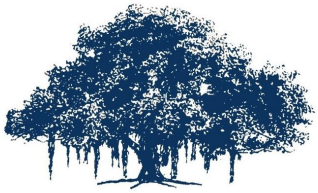
Precog Task - Graphs
Pranav Swarup Kumar

Lateral Entry CSD

Pranav Swarup Kumar 2025121011

Link to GitHub Repo Link to Resume



INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Contents

1 | Quick Overall Project Overview

"It matters not what someone is born, but what they grow to be." — Albus Dumbledore

Table 1.1: Dataset summary.

Entities	1,316 people
Relations	28 types (parent, sibling, grandparent, cousin, aunt/uncle)
Training triplets	13,821
Test triplets	590 (motherOf, fatherOf, sonOf, daughterOf only)
Family components	50 (completely disjoint, avg. size ∼26)

Task 1: Exploratory Data Analysis

Built a custom **interactive Streamlit dashboard** with PyVis for visual exploration of the family graph. Extracted per-entity features such as gender (inferred from relation types), generational depth, degree statistics, anomaly detection.
Key finding: The graph decomposes into 50 disjoint family components with no cross-family edges. Gender is reliably inferrable from relation types alone and does not need a weighted metric to decide, as there are no anomalies. (Both for gender and generation) (example: heads of `motherOf` are always female).

Task 2: Community Detection

Applied BFS, Louvain, Girvan-Newman, and Label Propagation to both the full graph and nuclear family sub-graphs.
Key finding: Standard algorithms fail on the full kinship graph — families are too dense internally, resembling cliques rather than the sparse-with-bottlenecks structure these algorithms assume. Restricting to nuclear family edges (parent-child + sibling) recovers meaningful communities. **This same density problem resurfaced in Task 4 as over-smoothing in R-GCN** — both are manifestations of the same graph property.

Task 3: Rule Mining

Implemented AMIE-style rule mining [18] for inverse and two-hop compositional rules. Evaluated with both standard confidence and PCA confidence [19].
Key finding: Standard confidence underestimates rule quality in open-world KGs. PCA confidence accounts for data incompleteness and gives more accurate assessments. Mined 750 two-hop rules out of $28^2 = 784$ possible; the 34 missing ones correspond to biologically impossible relation chains. **The mined inverse rules directly explained Task 4's 100% test set coverage**, closing the loop between symbolic and neural approaches.

Task 4: Link Prediction

Established baselines (random, frequency, inverse rules), trained DistMult and iterated through four versions of R-GCN, analysed embedding quality and failure modes.
Key findings:

- 100% of test triplets derivable from inverse rules. The test measures relational pattern learning, not unknown link prediction.
- DistMult achieves MRR = 0.733 (mean rank 1.9). Simple and effective.
- R-GCN underperformed initially (MRR 0.26) due to over-smoothing on dense components. After four iterations of diagnosis and architectural fixes, single layer, gated residual, matched hyperparameters, v4 reached MRR 0.52.
- The gated residual settled at $\alpha \approx 0.47$: the model found a near-equal blend of GNN structure and raw embeddings, slightly preferring the latter.
- Learned relation embeddings independently discovered kinship taxonomy (generational distance structure).

The Thread Through All Four Tasks

In hindsight, I'm pretty impressed how each task informed the next. Task 1's structural analysis revealed the disjoint components that Task 2 tried (and partially failed) to recover algorithmically. Task 2's failure on dense subgraphs foreshadowed Task 4's R-GCN struggles. Dense kinship structure breaks algorithms designed for sparse graphs, whether community detection or message-passing GNNs. Task 3's mined rules were perhaps the most insightful. They explained Task 4's inflated metrics. Perfect inverse coverage is why DistMult does so well and why structural reasoning is unnecessary for this particular test set.
In each task and subtask, I've *really* tried to go beyond what was asked and try out new implementations, came up with my own metrics and read up on literature to draw the right qualitative insights. Moreover, this Latex document is something I am very proud of. Everything here was typed by me.

I'd like to say - 10/10 task. Absolutely loved tinkering around with the amount of freedom I had. Lost track of how long I spent doing all this :)

2 | Task 1 - Primary Analysis

2.1 | Qualitative Insights from basic graph exploration

"Understanding is the first step to acceptance." - Albus Dumbledore

1. The concentration of high-degree nodes in intermediate generations reflects a natural hierarchical family structure in which connectivity peaks among individuals who simultaneously link ancestors and descendants. The distribution looks like a bell curve. Non-trivial number of isolated and sparsely connected individuals suggests intentional inclusion of peripheral or terminal family members.
2. The analysis reveals a knowledge graph that is highly structured yet, not mechanically generated by simple closure rules. The partial derivability of higher-order kinship relations indicates that the dataset is not just a transitive expansion of basic parent child relations, but also encodes a decent amount of independently asserted relational information.
3. The complete absence of structural anomalies and symmetry violations suggests that the graph was constructed under strong global consistency constraints, maybe via controlled generation or rigorous curation.

Note - Definition of Nuclear Family: Used a strict definition requiring one identified mother and one identified father per child, 445 nuclear families were detected. The average nuclear family size is 3.65 individuals.

Table 2.1: Dataset Summary Statistics

Metric	Value
Number of people	1,316
Number of triplets	13,821
Distinct relation types	28

Table 2.2: Gender Distribution

Gender	Count
Female (F)	670
Male (M)	646
Unknown	0
Ambiguous	0

Table 2.3: Generation Distribution

Generation	Count
0 (oldest)	164
1	265
2	316
3	316
4	188
5	60
6	7

Table 2.4: Anomaly and Consistency Checks

Category	Count
Clean individuals	1,316
Generation violations	0
Symmetry violations	0

Table 2.5: Family Structure Statistics

Metric	Value
Nuclear families	445
Average family size	3.65
Sibling groups	181

Table 2.6: Degree-Based Node Categories

Category	Count
High-degree nodes (≥ 40)	41
Isolated nodes (≤ 2)	85

Table 2.7: Derivable Relation Coverage

Relation	Derivable	Total	Coverage (%)
grandmotherOf	647	813	79.6
grandfatherOf	647	813	79.6
grandsonOf	459	814	56.4
granddaughterOf	507	812	62.4

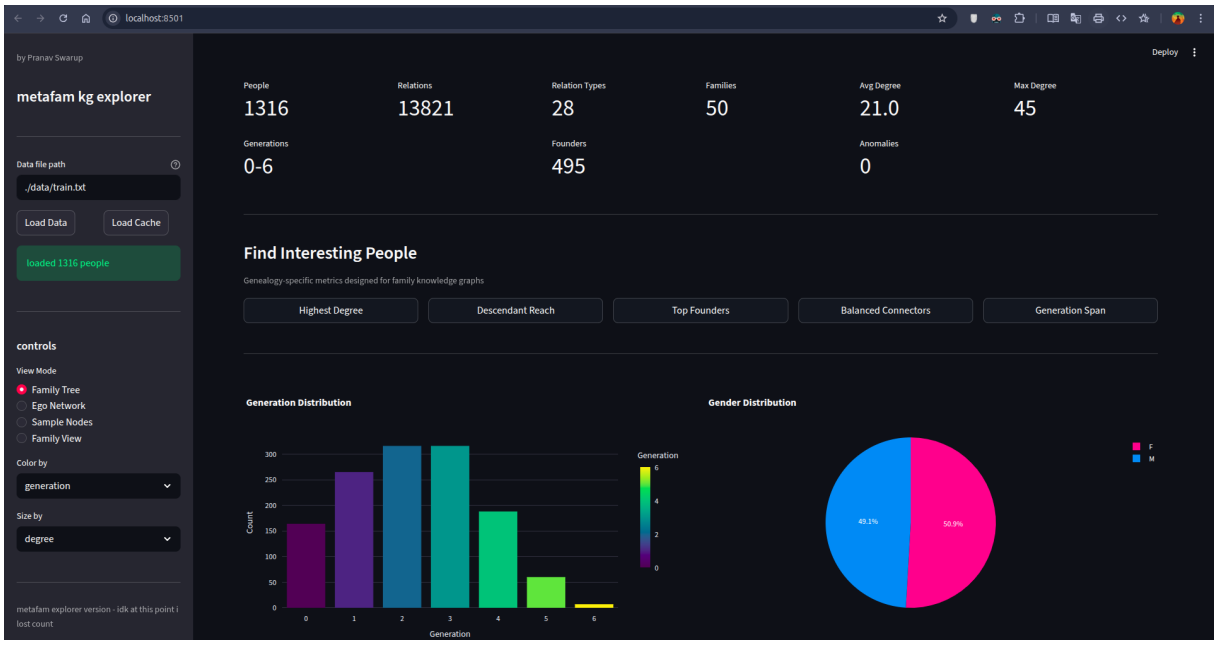


Figure 2.1: Overall Stats

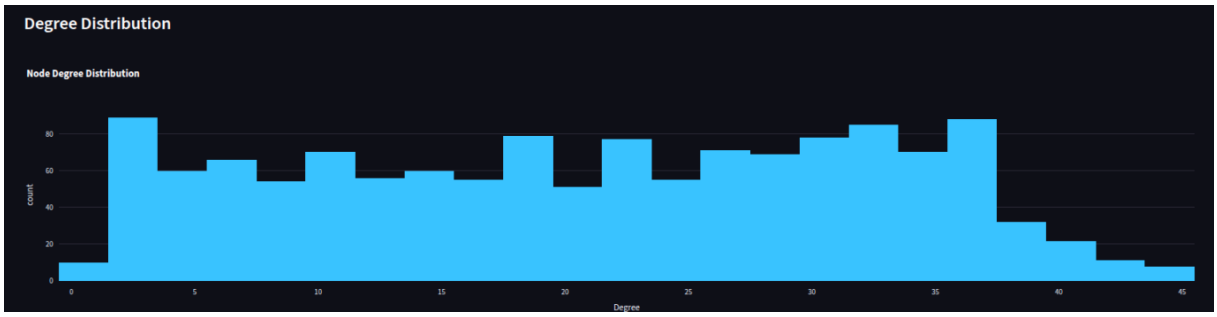


Figure 2.2: Node Degree Distribution

3 | Task 1 - Secondary Analysis Inspired by Graph Theory

This section focuses on global structure, intra-family connectivity, generational organization, and node roles. All metrics are computed on a graph $G = (V, E)$ where vertices represent individuals and edges represent kinship relations. Family membership and structural proximity are symmetric, and so I have used this ontological fact to perform analyses on an undirected projection (unless stated otherwise) of the graph to capture family connectivity independent of relation direction.

3.1 | Graph Construction and Assumptions

The original dataset consists of directed, typed triplets (h, r, t) encoding kinship relations. For graph-theoretic analysis, two representations are used:

- A directed graph G_d, preserving parent–child directionality where required.
- An undirected graph G_u, obtained by ignoring edge direction and relation type, used for connectivity and centrality analyses.

This projection assumes that family membership and structural proximity are symmetric notions, while acknowledging that genealogical semantics (e.g., ancestry) are directional.

3.2 | Basic Graph Statistics

Let $|V| = 1316$ and $|E_u| = 7480$ denote the number of nodes and undirected edges, respectively. Graph density is defined as

$$\rho = \frac{2|E_u|}{|V|(|V|-1)}.$$

Table 3.1: Basic Graph Statistics

Metric	Value
Nodes ($\|V\|$)	1316
Undirected edges ($\|E_u\|$)	7480
Directed triplets	13821
Density (ρ)	0.0086
Average degree	11.37
Degree range	1 – 23

Qualitative Interpretation The low density is very obvious considering how most node pairs are forbidden from connecting by ontology. Two parents will never have a sibling edge between them and so on. And so, the natural rules that family trees are based on constrain the relations between nodes greatly. Average degree and bounded degree range are more informative though. They confirm that connectivity is biologically and semantically constrained rather than an unbounded phenomenon.

3.3 | Connected Components and Family Decomposition

Connected components are computed on G_u. Each component corresponds to a maximal set of mutually related individuals. THIS HAS BEEN ELABORATED IN TASK 2's RESULTS.

Table 3.2: Connected Component Statistics

Metric	Value
Number of components	50
Largest component size	27
Smallest component size	26
Singleton components	0

Qualitative Interpretation Each connected component corresponds exactly to one extended family, with remarkably uniform size. This shows that the dataset seems to be partitioned into independent family units, and (much to my dismay) that no spurious cross-family relations exist. **This result is one of the strongest structural validations of the dataset.**

3.4 | Diameter and Path Lengths

For each connected component C, the diameter is defined as

$$\text{diam}(C) = \max_{u,v \in C} d(u,v),$$

where $d(u,v)$ is the shortest-path distance in G_u. Average shortest path length is computed as the mean of all pairwise distances.

Table 3.3: Path Length Statistics (Across Families)

Metric	Value
Diameter range	3 – 5
Most common diameter	3
Average path length (mean)	1.69

Qualitative Interpretation Small diameters and short path lengths are structurally expected in family graphs due to hierarchical parent-child chains combined with sibling and cousin shortcuts. These metrics primarily confirm the absence of weirdly long chains. Not much use in revealing emergent social properties.

3.5 | Clustering Coefficient

The local clustering coefficient of a node v is defined as

$$C(v) = \frac{2e_v}{k_v(k_v - 1)},$$

where k_v is the degree of v and e_v is the number of edges among its neighbors. Global clustering is the average of $C(v)$ over all nodes.

Table 3.4: Clustering Coefficient Statistics

Metric	Value
Global average clustering	0.7908
Clustering range (families)	0.69 – 0.90

Qualitative Interpretation It's high because of Sibling cliques and shared parents! That is a main feature of this graph and makes the results an obvious consequence of kinship ontology.
However, clustering stratified by generation is informative. Older generations exhibit higher clustering due to tightly connected sibling groups, while middle generations show lower clustering. This is an important insight into determining exactly why middle generations have a higher degree. They seem to play a bigger role as vertical connectors between ancestors and descendants, than exhibiting high sibling clustering.

3.6 | Centrality Measures

Standard centrality measures are computed within each connected component.

- Degree centrality: $C_D(v) = \frac{k_v}{|V_C|-1}$
- Betweenness centrality:

  $$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}},$$

  where σ_{st} is the number of shortest paths from s to t.
- Closeness centrality:

  $$C_C(v) = \frac{1}{\sum_{u \in V_C} d(v,u)}.$$

- Eigenvector centrality is measuring importance via important neighbours.

Table 3.5: Top Nodes by Standard Centrality Measures (In the Largest Family Component)

Person	Gen	Degree	Betweenness	Closeness
elias6	3	0.8462	0.0414	0.8667
lisa5	3	0.8462	0.0414	0.8667
nico4	3	0.8462	0.0414	0.8667
selina10	2	0.8077	0.0178	0.8387
olivia0	2	0.8077	0.0178	0.8387

Table 3.6: Top Nodes by Eigenvector Centrality

Person	Gen	Eigenvector Score
nico4	3	0.2366
elias6	3	0.2366
lisa5	3	0.2366
oskar24	2	0.2347
selina10	2	0.2347

Qualitative Interpretation Middle-generation individuals tend to score highly across degree, closeness, and eigenvector centrality because they connect both upward and downward. This is expected for a family graph. **Betweenness** centrality is inflated for parents in tree like structures and **should not be interpreted as social influence.** These metrics prove to fall short in describing importance since they merely reinforce semantic sense.

3.7 | Articulation Points

A node is an articulation point if its removal increases the number of connected components. Articulation points are identified on G_u.

Table 3.7: Articulation Point Statistics

Metric	Value
Total articulation points	95
Dominant generations	3–5

Qualitative Interpretation Initially I assumed this to be a strong result and tried verifying the seemingly "bridge persons" on my dashboard. What I found is that for tree-like family structures, nearly every non-leaf node is an articulation point. Thus, while the computation was right, articulation pointed to primarily identify parents rather than actual uniquely critical individuals.

3.8 | Generation-Specific Analysis

Individuals are grouped by inferred generation. Founders are defined as nodes with no parents; leaves as nodes with no children.

Table 3.8: Generation-Level Statistics

Gen	Count	Avg. Degree	Founders	Leaves
0	164	16.09	164	0
1	265	16.48	121	41
2	316	20.94	92	92
3	316	25.11	81	128
4	188	24.68	32	108
5	60	22.10	5	50
6	7	17.00	0	7

Qualitative Interpretation Founders concentrate in generation 0 and leaves in the highest generations, strongly validating the generation inference procedure. Average degree peaks in middle generations, consistent with their dual role as descendants and ancestors. (Bell Curve Shape as in Fig 1.1)

3.8.1 | Subgraph Validations

The parent-child subgraph is constructed from `motherOf` and `fatherOf` relations and analysed as a directed graph. The parent-child subgraph is a directed acyclic graph (DAG), forming a forest structure. The sibling sub-graph decomposes into fully connected cliques with no missing edges.

4 | Task 1 - Refined Centrality Metrics based on Genealogy

Table 4.1: Family-Specific Centrality Metrics

Metric	Formula	What it means
Descendant Count	$\lvert\text{Descendants}(v)\rvert$	Downstream genealogical impact - "How many people exist because of this guy"
Weighted Descendant Count	$\sum_{d \in \text{Desc}(v)} \frac{1}{\text{gen}(d) - \text{gen}(v)}$	Influence weakens as generation number of descendants increase. I tried capturing this by making sure closer descendants matter more.
Upward Diversity	$\lvert\text{Ancestors}(v) \cap \text{Founders}\rvert$	How many distinct founders (earliest ancestors) feed into a person. Degree of lineage mixing. Number of independent founder lines converging at v.
Role Defining Index	$\frac{\lvert\text{Desc}\rvert - \lvert\text{Anc}\rvert}{\lvert\text{Desc}\rvert + \lvert\text{Anc}\rvert + 1}$	Structural role: founder-like (closer to +1), leaf-like (closer to -1), or generational bridge (≈ 0).
Vertical Importance	$\lvert\text{Ancestors}(v)\rvert \times \lvert\text{Descendants}(v)\rvert$	Number of Ancestor-descendant pairs passing through v.
Generation Span (GS)	$\max(\text{gen}_{\text{desc}}) - \min(\text{gen}_{\text{anc}})$	Temporal reach of v across generations in the family tree.

- Metrics are computed on the **parent-child DAG only**, excluding lateral relations like siblings unless explicitly stated.
- Founders are defined as nodes with zero in-degree in the DAG.

4.1 | Descendant Count

Table 4.2: Descendant Count by Generation

Generation	Avg. Descendants
0	8.6
1	5.2
2	2.7
3	1.3
4	0.6
5	0.2
6	0.0

Table 4.3: Highest Descendant Count

Person	Generation	Descendants
christian1226	0	17
christian1276	0	17
larissa1275	0	17
luca763	0	16
florian1019	0	16
daniel1309	0	16
elena1041	0	16
amelie1211	1	16

Descendant Count decreases monotonically by generation, confirming strict hierarchical inheritance from founders to leaves.

4.2 | Upward Diversity

Table 4.4: Diversity Score by Ancestry (Distinct Founders)

Generation	Avg. Founder Lineages
0	1.0
1	0.5
2	0.3
3	0.3
4	0.2
5	0.1
6	0.0

This is an actually cool find! **The metric increases with generation depth**, indicating progressive lineage convergence rather than just pure descent. Distinct founders are becoming related due to marriages happening between their respective descendants.

4.3 | Role Defining Index

This result was surprising to me as all the nodes fit in the Balanced category. However the reason is in way the data is structured.

1. Small, symmetric, template-like families
2. Almost equal ancestor/descendant cones. Leading to an index of (≈ 0).
3. No long tails

Such a metric does not have dynamic range for graphs like these.

4.4 | Vertical Importance

Table 4.5: Centrality score based on how much the person allows vertical movement by Generation

Generation	Avg. Mediated Pairs
0	97.4
1	45.4
2	15.4
3	4.0
4	0.9
5	0.3
6	0.0

Another cool insight! In this dataset, founders mediate the majority of ancestor–descendant paths because they possess large descendant cones, while later generations rapidly lose out on this index as descendant counts diminish.

4.5 | Key Takeaway from Genealogy Inspired Metrics

These metrics demonstrate that centrality in family graphs is inherently *genealogical rather than social.* By explicitly modeling ancestry, descent, and generational mediation, they uncover structural roles and lineage dynamics that standard graph-theoretic centrality measures fail to capture.

5 | The Custom Built MetaFam Exploring Dashboard

"The truth is a beautiful and terrible thing." - Albus Dumbledore

5.1 | A brief overview

Since conventional pre made tools for visualising graphs did not help much when viewing a hierarchical family tree KG, I decided to build a full blown dashboard specifically for the given data. (spent a lot of time on this so... Do load it up and play around!)

1. It is built using **python, pyvis and streamlit** deployed locally. The homepage shows our derived metrics such as generations, founders, leaves, descendant reach, and generational balance, in a graphical form.
2. My favourite part is the hands-on exploration. You can graph **ego networks, family trees, or entire connected family groups**, with flexible controls for coloring and sizing nodes based on generation, gender, or connectivity, by taking person ID as input.
3. There are tools to trace **relationship paths** between two people and to drill down into detailed profiles for any individual, including their relatives and anomaly flags.

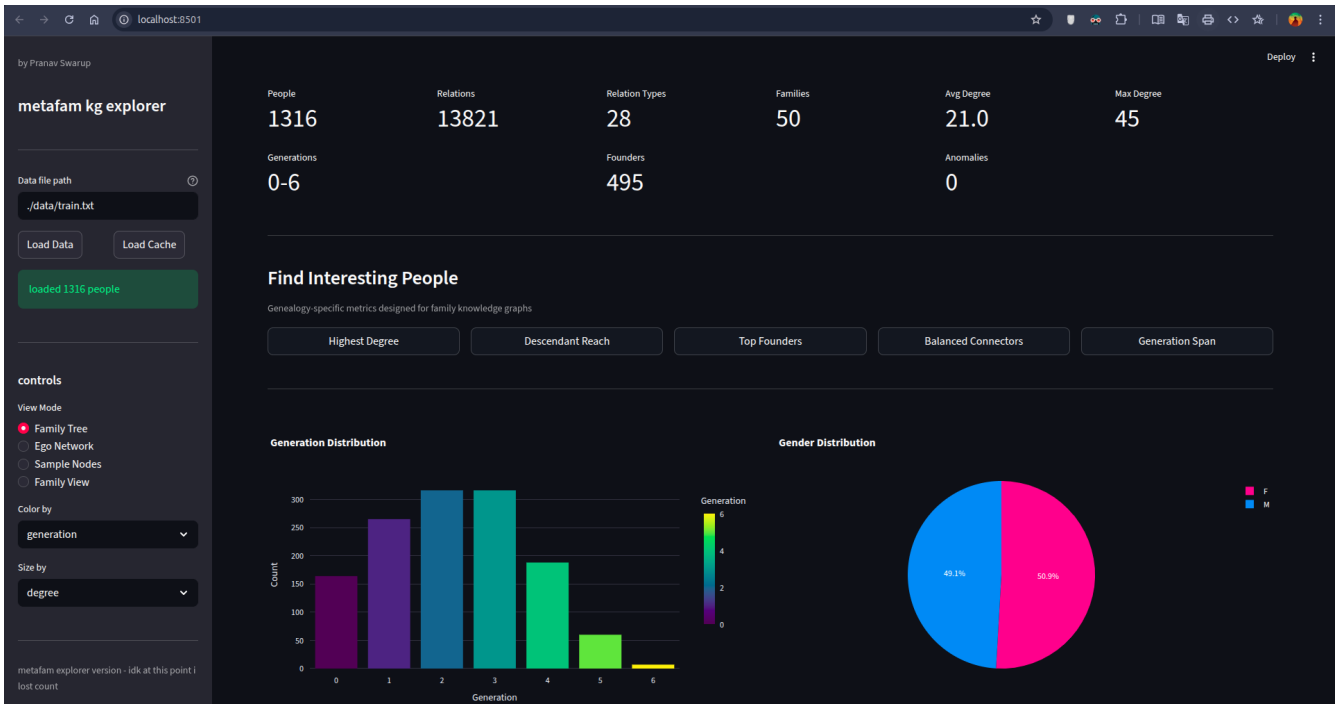


Figure 5.1: Overall Statistics

5.2 | Genealogical Results Plotted

1. Gender distribution is split evenly.
2. Decreasing trend of Descendant reach confirms hierarchical tree structure and strict hierarchical inheritance from founders to leaves.
3. Vertical Importance measures how much an individual mediates ancestor–descendant paths. Founders dominate this index because they sit above large descendant cones and therefore lie on many vertical paths. As generations progress, descendant sets shrink rapidly, causing a steep decline in mediated pairs, reaching near-zero for the youngest generations.
4. Most families are around the same size around 26-27 people. This indicates that the dataset was synthetically prepared or pruned to mimic this homogeneity
5. Middle generations have higher connectivity and this can be confirmed from the ontology of family trees.

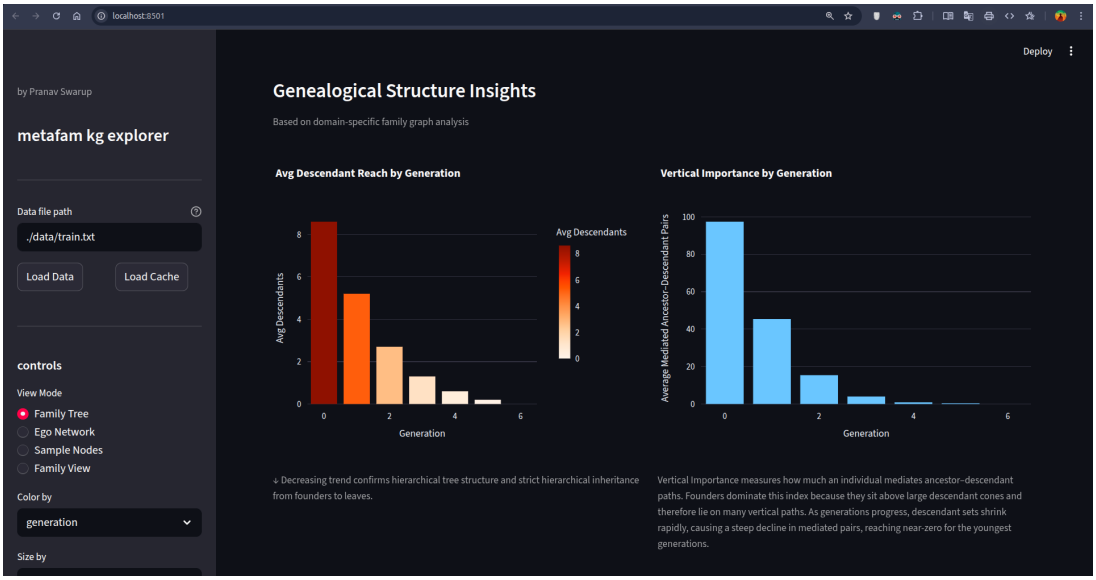


Figure 5.2: Genealogical Insights 1

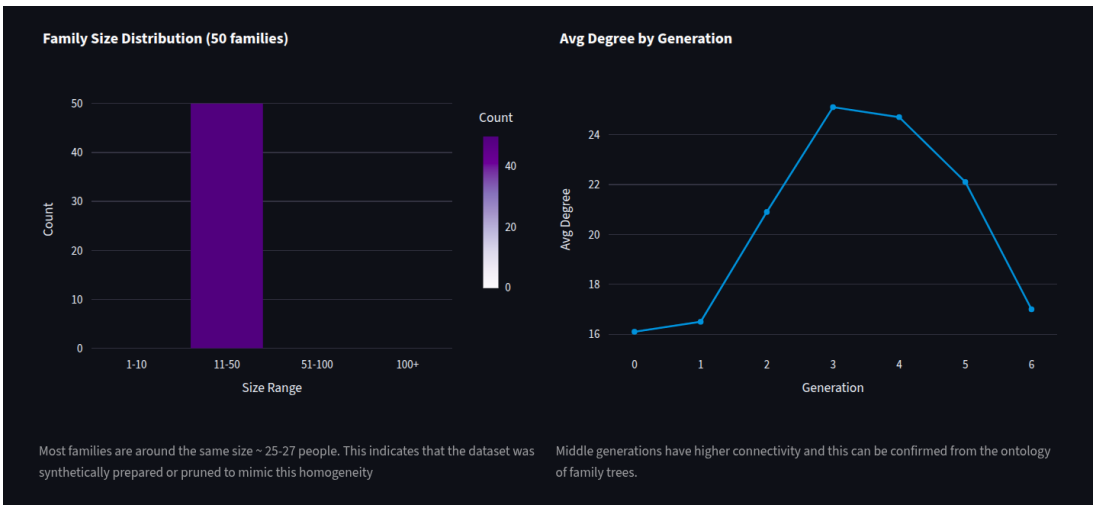


Figure 5.3: Genealogical Insights 2

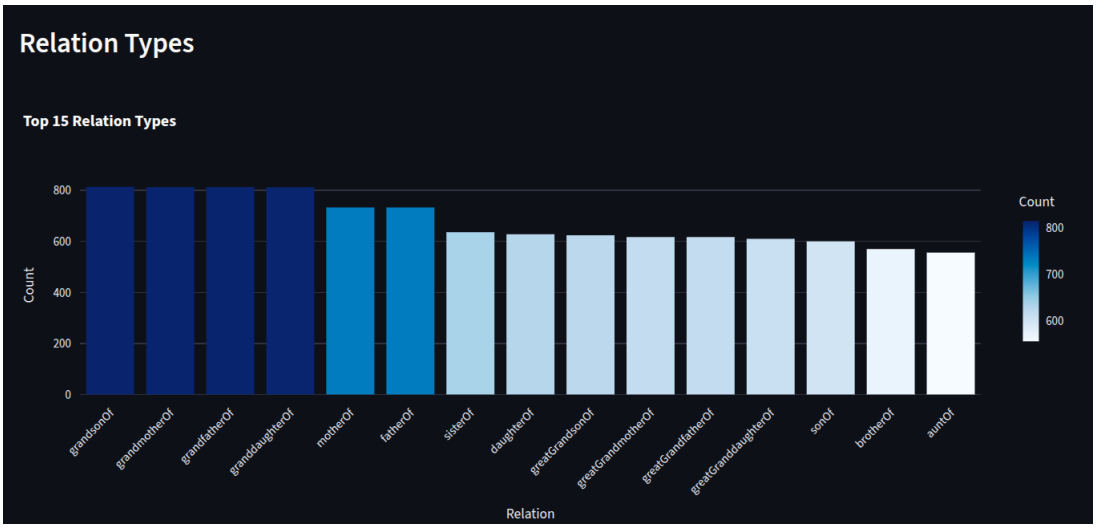


Figure 5.4: Top Relations

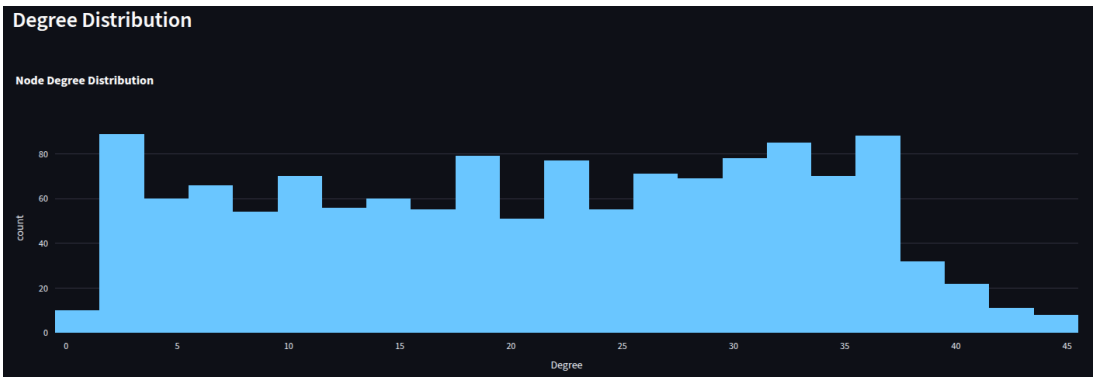


Figure 5.5: Degree Distribution

Figure 5.6: Plot of all Edges to an example node

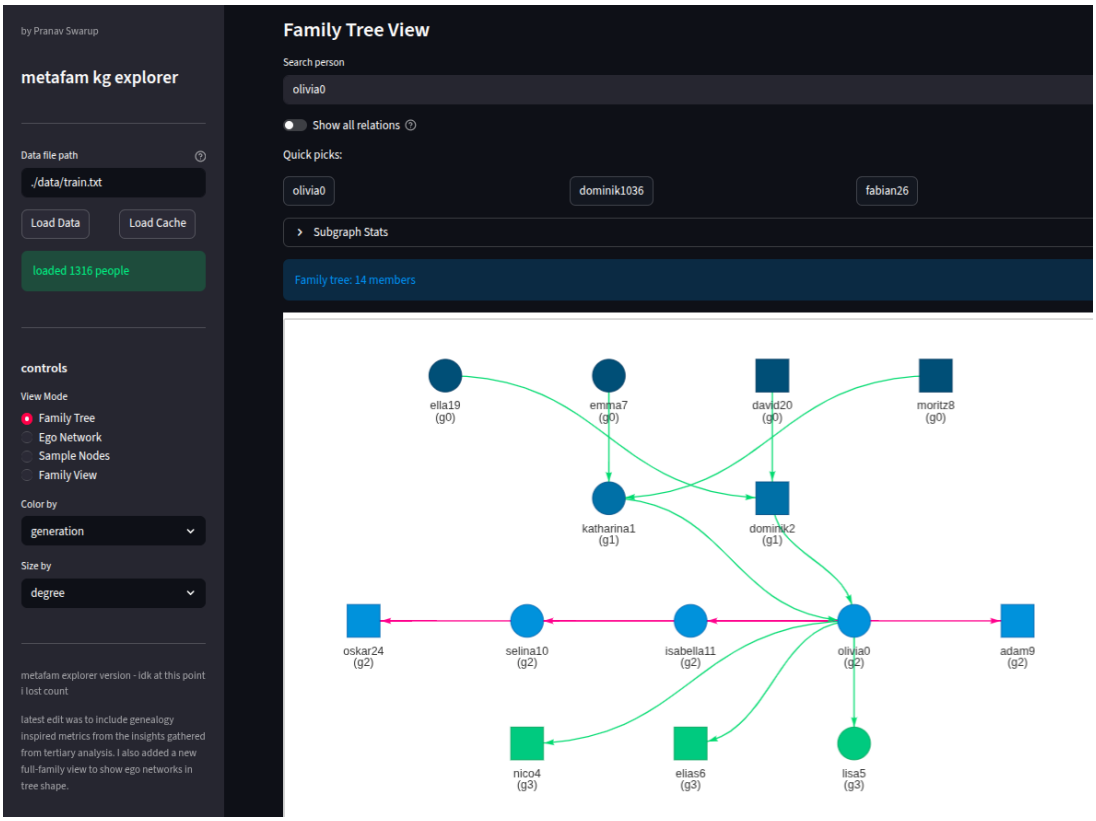


Figure 5.7: Limited Family View for a particular node

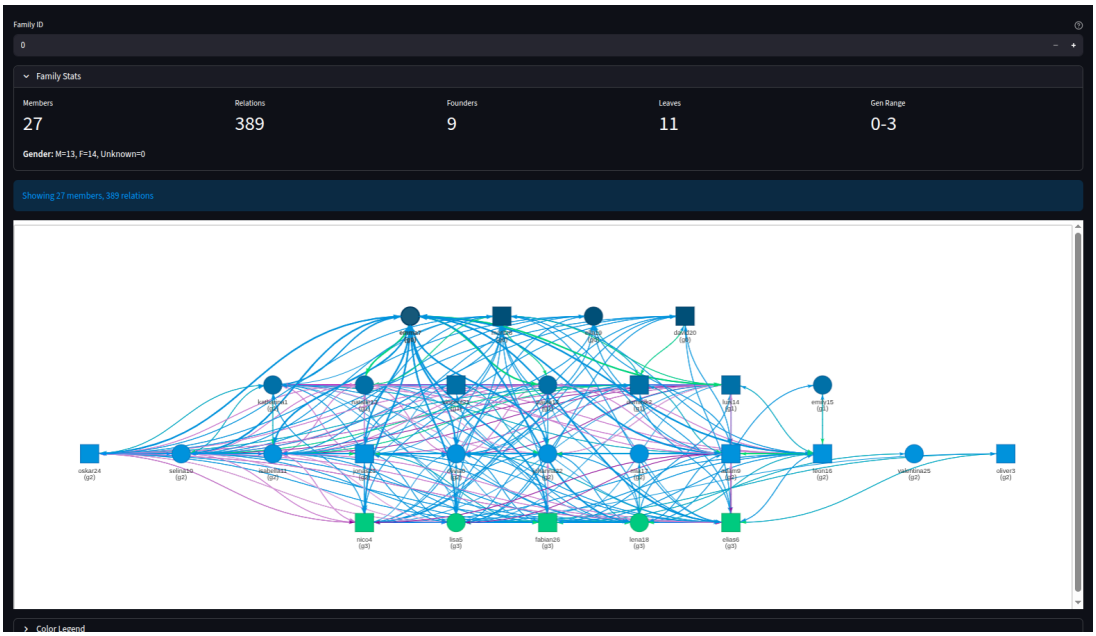


Figure 5.8: Full Family View of FamilyIndex = 0

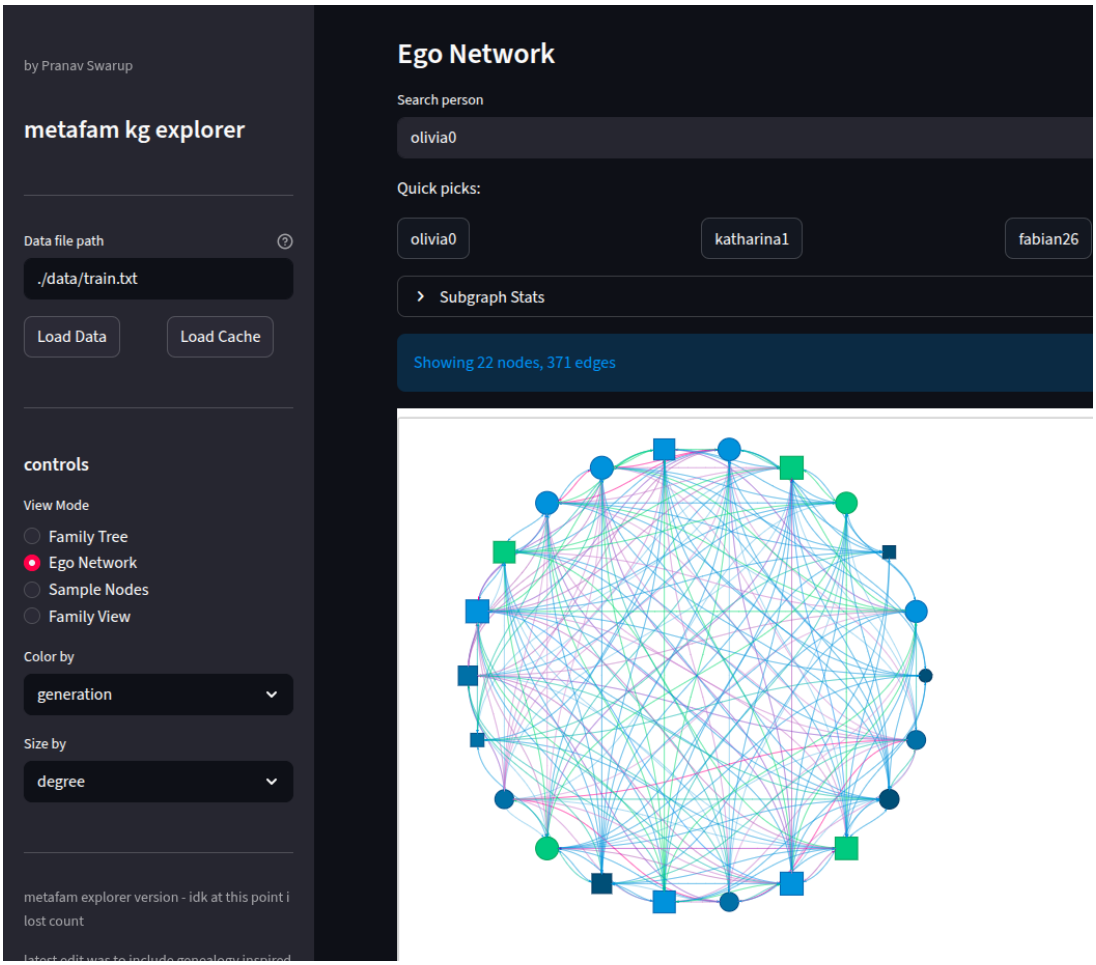


Figure 5.9: Ego Network

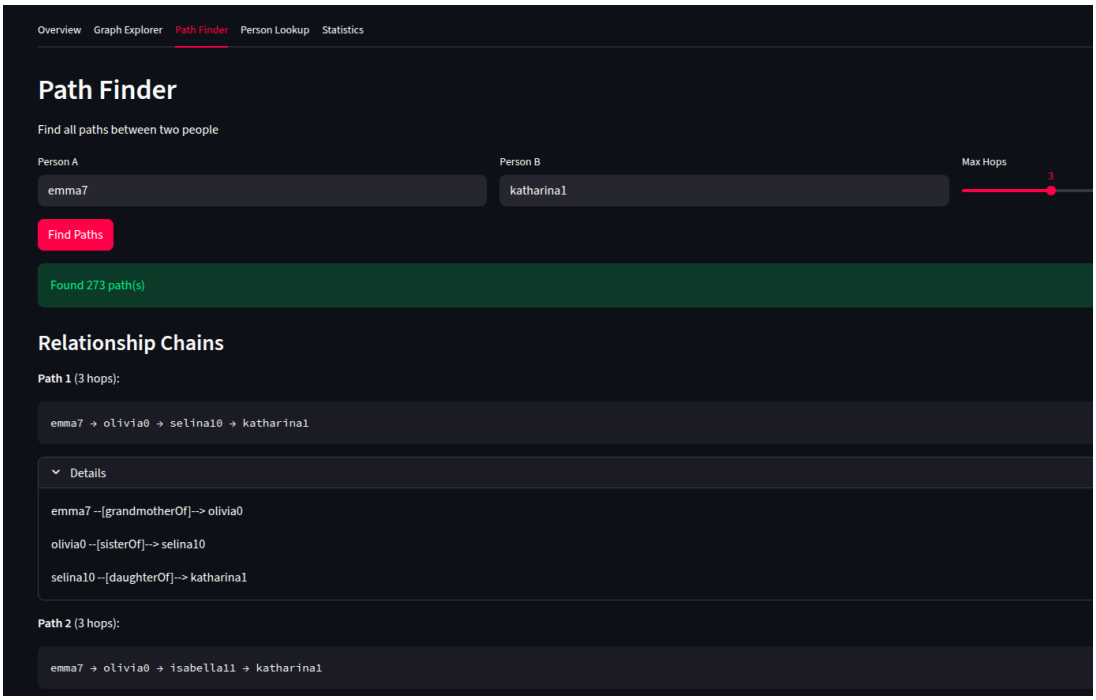


Figure 5.10: Path Finding with Tunable Parameters

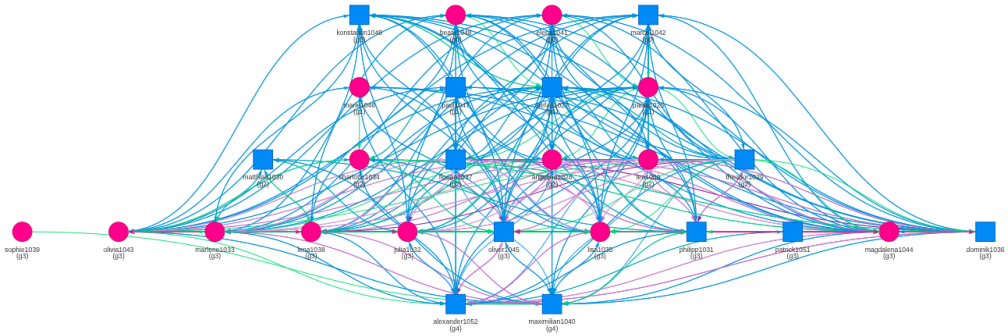


Figure 5.11: Another Family Tree coloured by Gender

6 | Task 2 Results

The uniform, synthetic structure of the dataset with 50 disjoint families of nearly identical size, renders much of the standard community detection narrative uninformative at face value. At the graph-wide level, the first hypothesis was to check if each connected component corresponds exactly to one family. If true, notions such as "bridge individuals connecting different family clusters" are structurally impossible, as the graph contains no cross-family edges. So the focus in task 2 pivots to applying a variety of tools to predict communities within connected groups. Another key focus is to propose metrics/methods to rank how related two people are using a variety of techniques.

6.1 | Graph-Theoretic Algorithms for Full Graph Community Detection

6.1.1 | The Humble BFS

A sanity script based on BFS traversal was used to verify graph connectivity. This confirmed the existence of exactly 50 disjoint connected components, each corresponding to a single extended family. The result matches `networkx`'s `connected_components()` output exactly.

```
…/cog  main ✘?  17:24
❯ python -m src.task2.family_disjoint_verification

Dataset: 1316 nodes, 7480 undirected edges
Manual components: 50
NetworkX components: 50
Manual BFS verifies what NetworkX gave us
family sizes: [27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 26, 26,
26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26,
 26, 26, 26, 26, 26, 26, 26, 26, 26, 26]
```

Figure 6.1: BFS-based verification of family-level disjoint connected components.

6.1.2 | Louvain

The Louvain algorithm was applied to the full undirected mapping of the given KG. It recovered exactly 50 communities with a modularity score of 0.979, with community sizes tightly concentrated between 26 and 27 nodes. Since there are no inter-family edges, modularity maximization cannot benefit from splitting or merging components. **This is a forced optimum and not a new discovery.**

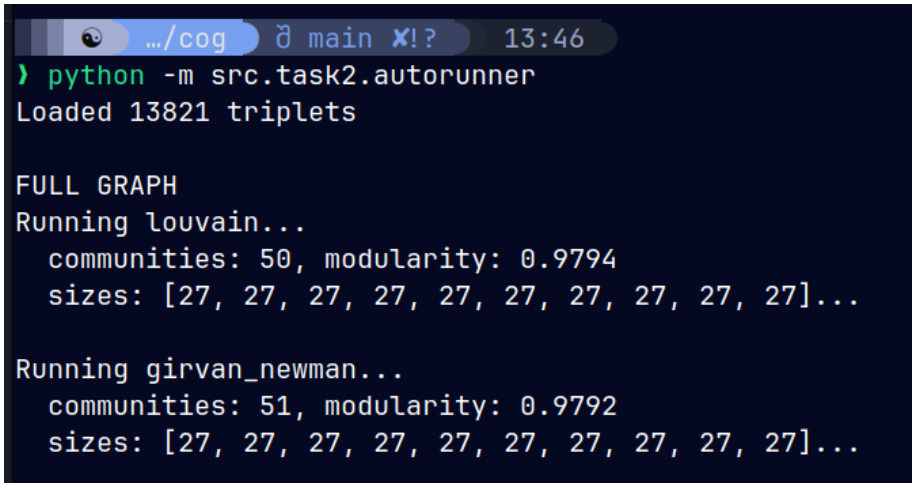


Figure 6.2: Louvain and Girvan-Newman Results

6.1.3 | Girvan-Newman

The Girvan-Newman algorithm was applied to the full undirected graph. By iteratively removing edges with highest betweenness, the method produced 51 communities with a modularity score of 0.979. The resulting sizes again lie tightly in the range of 26-27 individuals, aligning almost perfectly with the family-level connected components.
The slight over-segmentation (51 instead of 50 communities) arises from the algorithm's tendency to split highly tree-like structures once central parent-child edges are removed.
But mainly, Girvan-Newman does not merge distinct families. Inter-family separation is structurally absolute in this graph. **The deviation from the 50-family partition is due to algorithmic granularity rather than data ambiguity.**

6.1.4 | Label Propagation

Label propagation was run as a purely local, non-parametric baseline. The algorithm produced 64 communities with a high modularity score of 0.965. The method seems to occasionally over-fragment families. Despite this, label propagation never merges distinct families, and if anything, may merely over cut within a family group.

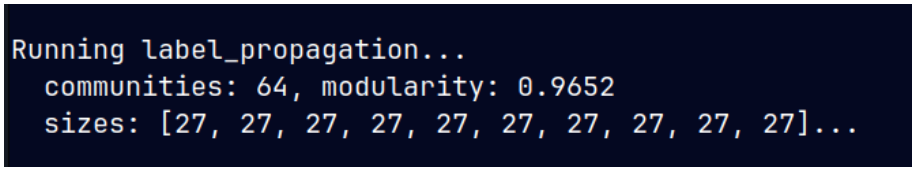


Figure 6.3: Label Prop

6.2 | ML Algorithms for Full Graph Community Detection

6.2.1 | Node2vec + k-means

An embedding-based machine learning approach was applied using node2vec followed by k-means clustering. This pipeline produced only **9 clusters** with modularity 0.868, each containing multiple families. Here I hypothesise that this behaviour arises because node2vec optimizes for role-based structural similarity rather than preserving connected components. Causing families with similar internal topology to collapse into shared embedding regions. This highlights an interesting difference between connectivity-preserving methods and representation-learning approaches.

```
Running node2vec_kmeans...
  communities: 9, modularity: 0.8681
  sizes: [224, 198, 195, 157, 131, 121, 105, 105, 80]
```

Figure 6.4: Node2Vec + K means

6.2.2 | Spectral Clustering

Spectral clustering was applied to the full graph using the precomputed adjacency matrix. This method assumes a globally connected graph and seeks low-dimensional embeddings. Since the genealogical graph consists of 50 disconnected components, this assumption is violated, leading to unstable embeddings.

```
/home/pranav/psk/cog/venv/lib/python3.12/site-packages/sklearn/man
ifold/_spectral_embedding.py:324: UserWarning: Graph is not fully
connected, spectral embedding may not work as expected.
  warnings.warn(
  communities: 7, modularity: 0.4640
  sizes: [895, 213, 132, 26, 26, 13, 11]
```

Figure 6.5: Spectral Clustering Result (with Error from sklearn)

As a result, spectral clustering **produced only 7 communities** with a substantially lower modularity of 0.464, merging multiple families into large clusters while leaving a few small families intact. Spectral techniques optimize for global variance in connectivity rather than preserving hard connectivity constraints, making them unsuitable for graph-wide family detection in this setting.

6.3 | All Algorithms tested on (Intra-Family) Sub-Graphs

Intra-family community detection within undirected family sub-graphs fails to reveal stable or meaningful substructure. **Modularity values remain uniformly low, with high variance across families**, indicating weak and unstable partitions that are sensitive to minor structural differences rather than reflecting genuine communities.

Girvan–Newman exhibits extreme over-fragmentation, often producing many small communities (up to 15 per family) with near-zero modularity, a consequence of recursively cutting high-betweenness parent–child edges in tree-like genealogical structures.
In contrast, label propagation collapses each family into a single community with zero modularity, reflecting agreement in dense sibling and cousin-dominated graphs.

These results demonstrate that dense lateral kinship relations erase the sparsity gradients required for community detection, rendering standard algorithms either unstable, overly aggressive, or entirely degenerate when applied to full genealogical family graphs.

```
INTRA-FAMILY | full_undirected graph

louvain (across 10 families):
  modularity: mean=0.1925, std=0.0859, range=[0.0726, 0.3876]
  communities: mean=2.6, range=[2, 4]

girvan_newman (across 10 families):
  modularity: mean=0.0562, std=0.0573, range=[-0.0000, 0.1605]
  communities: mean=10.5, range=[2, 15]

spectral (across 10 families):
  modularity: mean=0.1572, std=0.1037, range=[0.0071, 0.3876]
  communities: mean=3.4, range=[2, 5]

label_propagation (across 10 families):
  modularity: mean=0.0000, std=0.0000, range=[0.0000, 0.0000]
  communities: mean=1.0, range=[1, 1]

node2vec_kmeans (across 10 families):
  modularity: mean=0.1782, std=0.0948, range=[0.0195, 0.3876]
  communities: mean=2.4, range=[2, 4]
```

Figure 6.6: Intra-Family metrics (Averaged over 10 families)

6.4 | Alternative Graph Representations

Graph wide metrics resulted in no new insights about the data. Moreover, the consistently weak and unstable results obtained from even intra-family community detection, led me to question whether the algorithms would give better results if I played around with the graph representation itself!
I hypothesised that the dense lateral relations present in genealogical data such as sibling, cousin, and extended relationship edges were overwhelming the community detection algorithms. To test this, I first experimented with retaining only parent–child relations while discarding same-generation and cousin edges. The resulting graphs exhibited substantially clearer and more stable clustering. This observation motivated a systematic comparison of multiple graph representations on a single family.

```
…/cog  main ✘!?  14:29
❯ python -m src.task2.autorunner
Loaded 13821 triplets

GRAPH REPRESENTATION COMPARISON (family 0)
Family size: 27 nodes

--- full_undirected (206 edges) ---
  louvain: 2 comms, mod=0.0726, sizes=[15, 12]
  girvan_newman: 2 comms, mod=-0.0000, sizes=[26, 1]
  spectral: 4 comms, mod=0.0071, sizes=[22, 2, 2, 1]
  label_propagation: 1 comms, mod=0.0000, sizes=[27]
  node2vec_kmeans: 3 comms, mod=0.0195, sizes=[12, 9, 6]

--- nuclear_family (56 edges) ---
  louvain: 5 comms, mod=0.5263, sizes=[8, 6, 5, 4, 4]
  girvan_newman: 7 comms, mod=0.5035, sizes=[6, 6, 5, 4, 4, 1, 1]
  spectral: 5 comms, mod=0.5263, sizes=[8, 6, 5, 4, 4]
  label_propagation: 4 comms, mod=0.4794, sizes=[12, 6, 5, 4]
  node2vec_kmeans: 5 comms, mod=0.5263, sizes=[8, 6, 5, 4, 4]

--- generational_affinity (206 edges) ---
  louvain: 3 comms, mod=0.1369, sizes=[12, 9, 6]
  girvan_newman: 9 comms, mod=0.0036, sizes=[19, 1, 1, 1, 1, 1, 1, 1, 1]
  spectral: 2 comms, mod=0.0574, sizes=[18, 9]
  label_propagation: 1 comms, mod=0.0000, sizes=[27]
  node2vec_kmeans: 2 comms, mod=0.0924, sizes=[19, 8]

--- relation_weighted (206 edges) ---
  louvain: 5 comms, mod=0.2165, sizes=[8, 6, 5, 4, 4]
  girvan_newman: 15 comms, mod=0.0235, sizes=[13, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
  spectral: 5 comms, mod=0.2013, sizes=[9, 8, 4, 3, 3]
  label_propagation: 1 comms, mod=0.0000, sizes=[27]
  node2vec_kmeans: 4 comms, mod=0.2010, sizes=[12, 6, 5, 4]
```

Figure 6.7: Results for Varying Representations of a Family Sub-Graph

6.4.1 | Full Undirected Graph - For Baseline

The full undirected representation treats every kinship relation as an unweighted, symmetric edge. This is the same as what I ran the previous metrics on. Most algorithms either collapse the family into a single community or produce weak, unstable splits with near-zero modularity. I am mentioning it here again just to provide a baseline for the other representations.

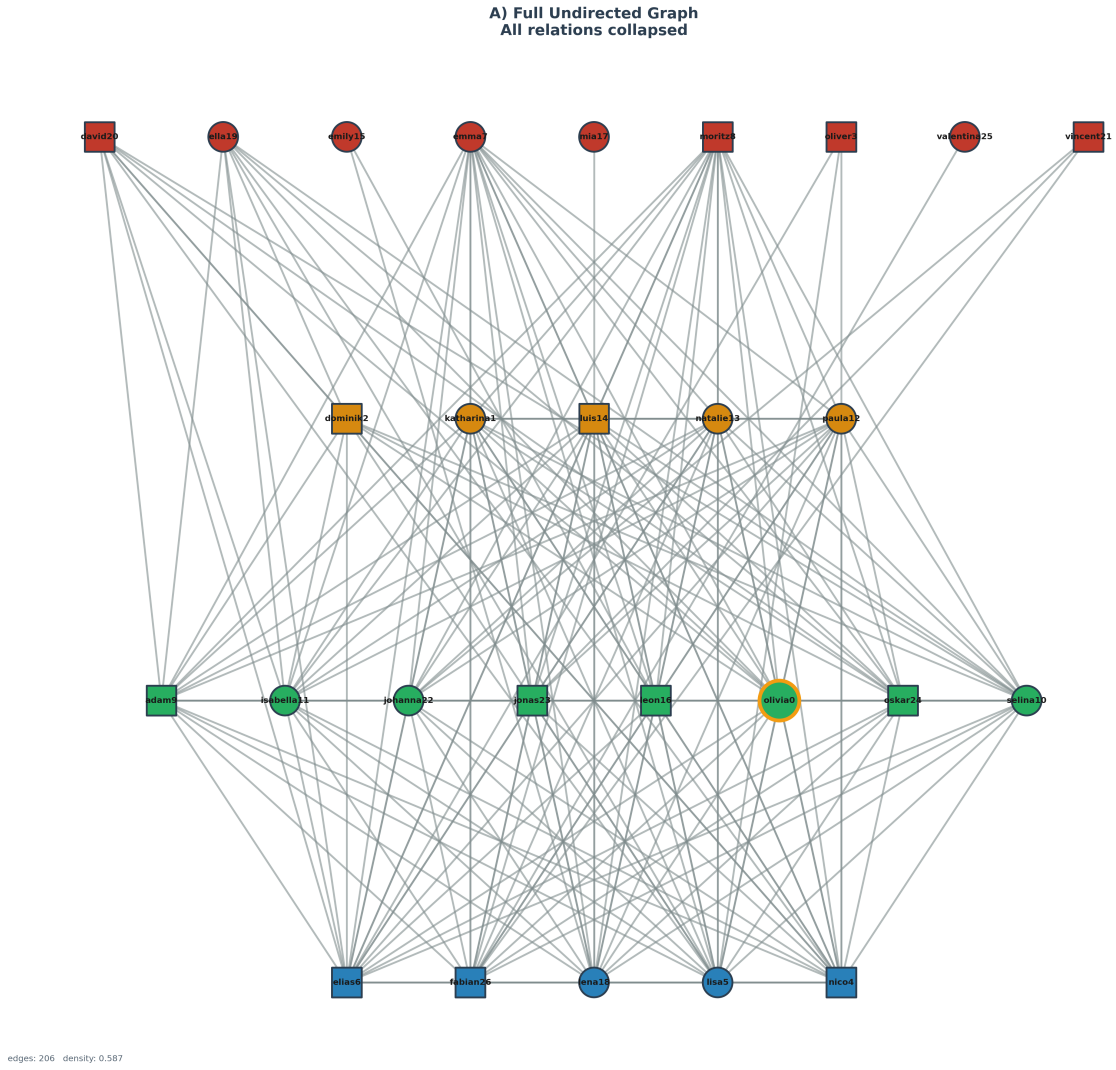


Figure 6.8: Full Undirected Representation of Family0

6.4.2 | Nuclear Family Graph

The nuclear-family representation retains only parent–child and sibling relations, explicitly removing cousins and extended kinship edges. This drastically reduces edge density and restores a clear hierarchical backbone to the graph. Across all algorithms, this representation yields consistent multi-community partitions with high modularity of approx 0.5, closely corresponding to nuclear family units. This confirms the hypothesis that lateral relations were masking meaningful structure.

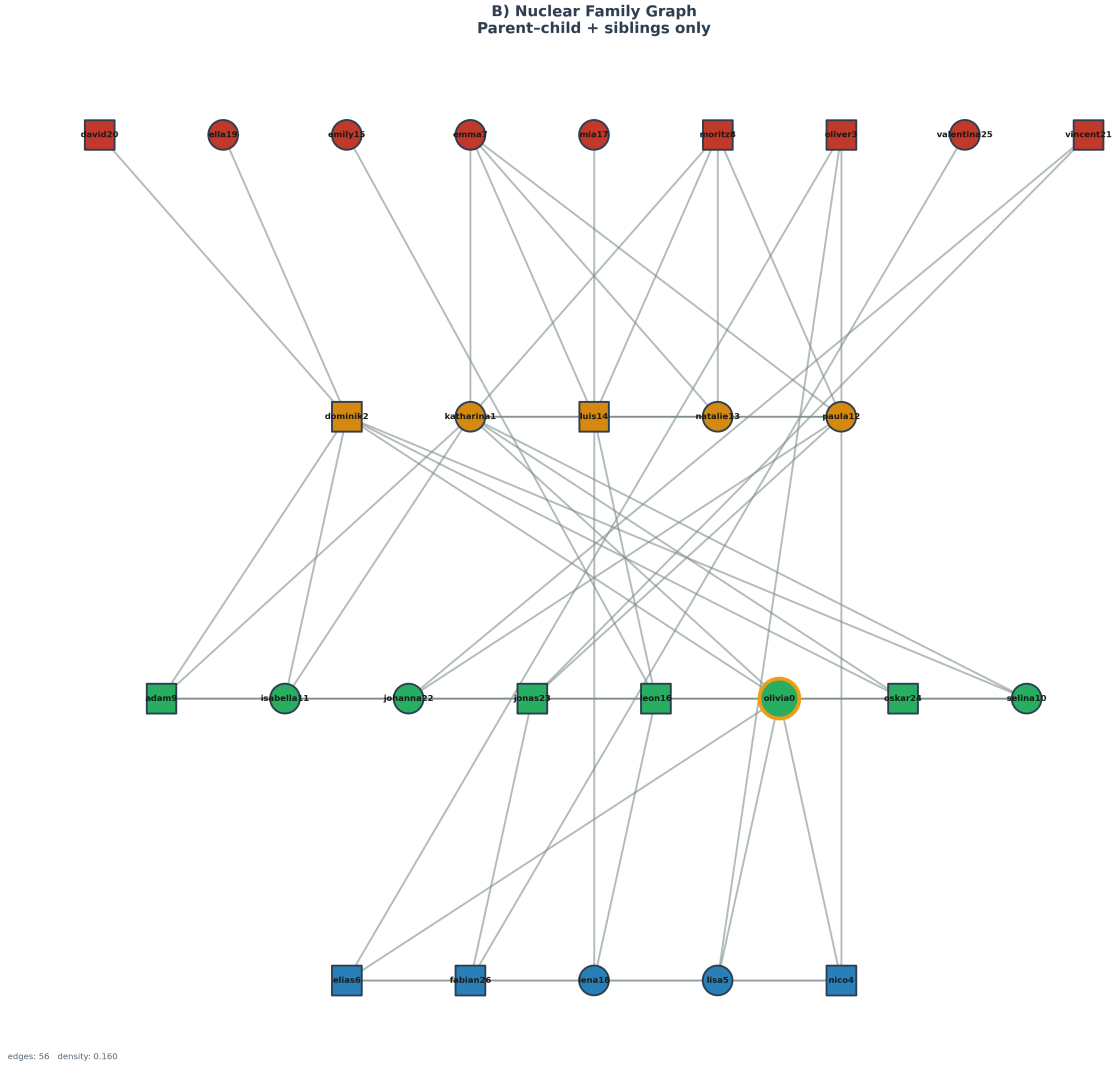


Figure 6.9: Nuclear Family Representation of Family0

6.4.3 | Generational Affinity Graph

The generational-affinity graph preserves all relations but weights edges inversely by generational distance, giving higher importance to same- or adjacent-generation connections. While this softens the influence of distant kinship, it does not eliminate dense lateral connectivity. As a result, community structure improves marginally but remains weak, indicating that weighting alone is insufficient to counteract the structural dominance of extended relations.

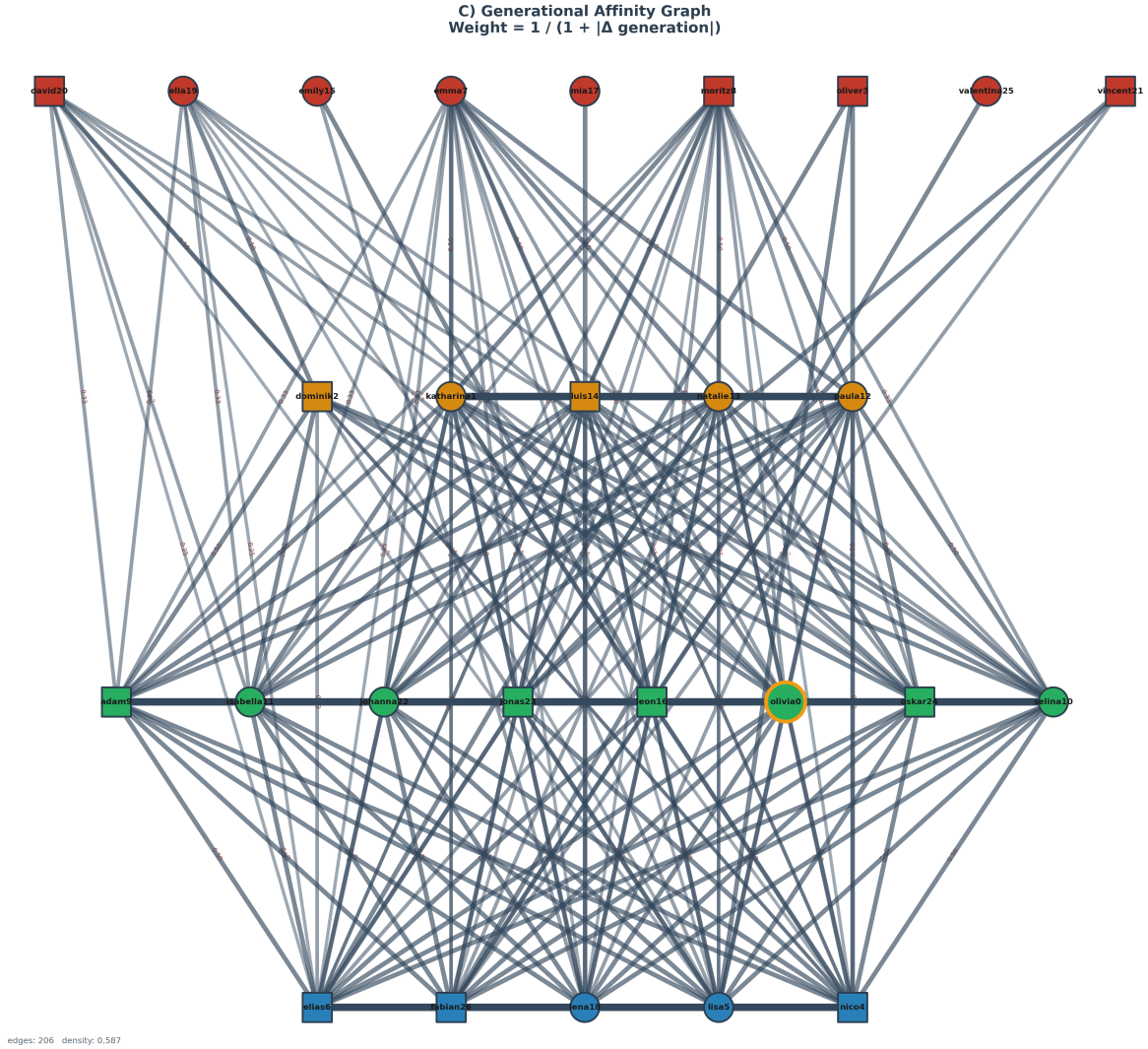


Figure 6.10: Generation Affinity Representation of Family0

6.4.4 | Relation-Weighted Graph

In the relation-weighted graph, edges are weighted based on semantic closeness, with sibling and parent–child relations assigned higher weights than distant relatives. This representation partially recovers community structure, particularly for modularity-based and spectral methods, but still fails to produce stable partitions across all algorithms. The persistence of weak clustering suggests that **explicit pruning of relations is more effective than continuous weighting in genealogical graphs.**

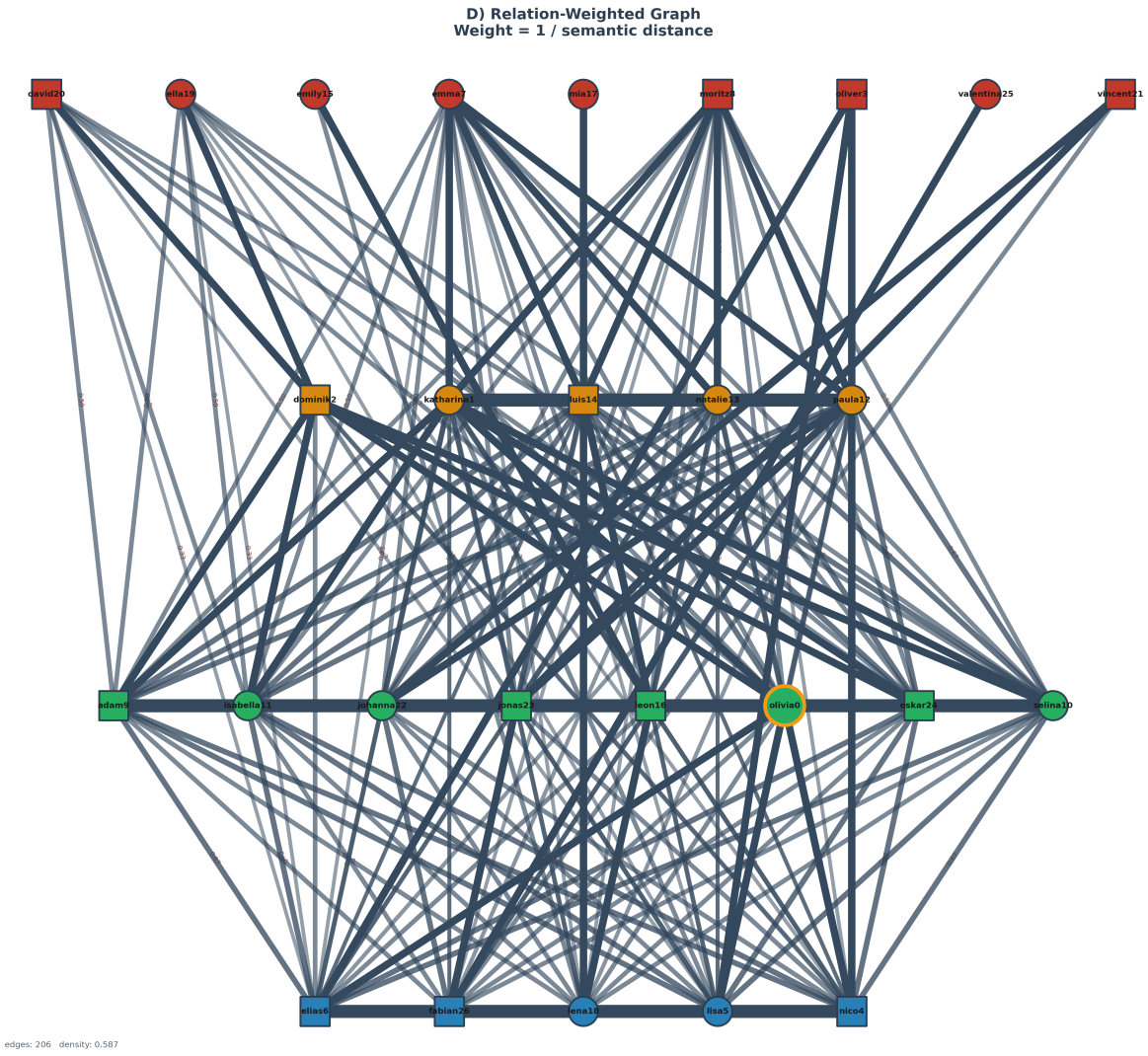


Figure 6.11: Enter Caption

6.5 | Exploring Nuclear Families Deeper

Having established that restricting the graph to nuclear relations improves structural clarity, I examined whether this representation yields stable and method-agnostic community structure across 10 random families.

```
>>> Nuclear family graph representation <<<

INTRA-FAMILY | nuclear_family graph

louvain (across 10 families):
  modularity: mean=0.5596, std=0.0424, range=[0.4877, 0.6235]
  communities: mean=5.2, range=[5, 6]

girvan_newman (across 10 families):
  modularity: mean=0.5554, std=0.0471, range=[0.4688, 0.6191]
  communities: mean=5.6, range=[5, 7]

spectral (across 10 families):
  modularity: mean=0.5472, std=0.0396, range=[0.4688, 0.6100]
  communities: mean=4.8, range=[4, 6]

label_propagation (across 10 families):
  modularity: mean=0.5098, std=0.0953, range=[0.2604, 0.6100]
  communities: mean=6.1, range=[4, 7]

node2vec_kmeans (across 10 families):
  modularity: mean=0.5583, std=0.0413, range=[0.4877, 0.6235]
  communities: mean=5.2, range=[4, 6]


 …/cog  main ✘!?  15:20
❯
```

Figure 6.12: All metrics on 10 random families represented as Nuclear Families

1. **High and stable modularity across methods.** All algorithms except label propagation achieve mean modularity values clustered tightly in the range $[0.54, 0.56]$, with low variance. This indicates that the nuclear graph induces a partitioning that is not algorithm-specific but instead reflects an intrinsic structural signal.

2. **Low dispersion in community counts.** Louvain, Girvan-Newman, spectral clustering, and Node2Vec+KMeans all converge to approximately five to six communities per family. This consistency suggests that the nuclear representation imposes a natural scale on community formation, rather than allowing arbitrary fragmentation.

3. **Agreement between fundamentally different methods.** Modularity optimisation (Louvain), divisive approaches (Girvan-Newman), spectral methods, and embedding-based clustering all recover comparable partitions. Such agreement which was not there with the full undirected graph points to a well-conditioned adjacency structure.

4. **Embedding-based methods mirror modularity optimisation.** Node2Vec followed by KMeans closely matches Louvain in both modularity and community count. This alignment implies that random-walk embeddings over the nuclear graph successfully encode the same structural information captured by direct modularity maximisation.

6.6 | Fragility vs. Redundancy Analysis

Though I had a genealogy metrics such as Vertical Importance and Upward Diversity from Task 1, I wanted to try removing certain nodes and analysing if they have a direct impact in the orphaning of downstream generation nodes.
For five families, I constructed an (i) undirected kinship graph containing all relations and (ii) a directed parent child DAG capturing ancestry. Each individual was removed in turn, and the resulting structural impact was quantified using two measures. Increase in the number of connected components and the number of orphaned descendants in the DAG.

Table 6.1: Summary of node-removal effects by generation (representative families).

Generation	Count	Avg. ΔComponents	Avg. Reach Loss (%)	Avg. Orphaned	Splits?
0	2–6	0.00	≈7.7	10–17	No
1	3–7	0.00	≈7.7	3–10	No
2	4–11	0.00	≈7.7	1–4	No
3	4–7	0.14–0.40	8.7–10.3	≤1	Yes
4–5	1–6	0.33–1.00	≈15.1	0	Yes

The analysis revealed another expected fact. Full kinship graphs are highly redundant. Removal of most individuals does not increase the number of connected components, and reachability loss remains modest and uniform across generations. Structural fragility is confined to a small number of deep-generation nodes, whose removal detaches a single terminal branch from the main component. Conversely, removing ancestral individuals produces large numbers of orphaned descendants in the DAG without fragmenting the undirected graph.

6.7 | Different Approaches to Closest Relation Metrics

Standard graph-theoretic distance on the full undirected graph is effectively uninformative, as the **dense lateral connectivity collapses most intra-family shortest paths to one or two hops.**

This uniformity renders hop count unsuitable for measuring relative relatedness. While a purely genealogical path distance based on lowest common ancestors is well-defined on the parent-child DAG, it ultimately recovers known kinship tiers in a discrete and deterministic manner, offering limited additional insight. Instead, I focus on two complementary approaches that extend beyond hop count while preserving semantic meaning.

6.7.1 | Relation Aware Distance

First, I employ **relation–aware distances** by computing shortest paths on a relation-weighted graph, where edges are weighted by genealogical and semantic proximity. This allows individuals at the same hop distance to be meaningfully differentiated (like siblings versus cousins).

```
 …/cog  main ✘!?  20:03
❯ python -m src.task2.relation_weighed_distance
relation weighed distance analysis

targetguy: oliver3 (gen 2)

Hops   Count   Weighted Range       Example
1      3       [1.0 - 1.0]      closest=lisa5(1.0), farthest=nico4(1.0)
2      19      [2.0 - 4.0]      closest=olivia0(2.0), farthest=moritz8(4.0)
3      4       [4.0 - 5.0]      closest=emily15(4.0), farthest=valentina25(5.0)

Top 10 closest by weighted distance:
  lisa5           dist=1.00  hops=1  gen=3 [fatherOf]
  elias6          dist=1.00  hops=1  gen=3 [fatherOf]
  nico4           dist=1.00  hops=1  gen=3 [sonOf]
  olivia0         dist=2.00  hops=2  gen=2
  selina10        dist=2.50  hops=2  gen=2
  isabella11      dist=2.50  hops=2  gen=2
  oskar24         dist=2.50  hops=2  gen=2
  adam9           dist=2.50  hops=2  gen=2
  katharina1      dist=3.00  hops=2  gen=1
  leon16          dist=3.00  hops=2  gen=2
  dominik2        dist=3.00  hops=2  gen=1
  johanna22       dist=3.00  hops=2  gen=2
  jonas23         dist=3.00  hops=2  gen=2
  paula12         dist=3.50  hops=2  gen=1
  natalie13       dist=3.50  hops=2  gen=1
  luis14          dist=3.50  hops=2  gen=1
  lena18          dist=4.00  hops=2  gen=3
  ella19          dist=4.00  hops=2  gen=0
  david20         dist=4.00  hops=2  gen=0
  fabian26        dist=4.00  hops=2  gen=3
```

Figure 6.13: Relation Weighted Path Lengths

6.7.2 | Jaccard Similarity

Second, I introduce a **shared-ancestor overlap metric, defined as the Jaccard similarity between the ancestor sets of two individuals in the parent-child DAG**. This metric directly captures genealogical closeness through shared lineage, naturally accommodates differing generational depths, and yields an interpretable score in $[0, 1]$. Together, these methods address the shortcomings of unweighted graph distance and provide structurally grounded, rank-able measures of relatedness that align with genealogical intuition.

$$\text{Overlap}(A, B) = \frac{|\text{Anc}(A) \cap \text{Anc}(B)|}{|\text{Anc}(A) \cup \text{Anc}(B)|},$$

(yielding a score in $[0, 1]$ directly reflects shared lineage. lower score can occur between cousins with common grandparents but not parents etc.)

```
❯ python -m src.task2.ancestor_overlap
ANCESTOR OVERLAP RELATEDNESS

target_guy: valentina25 (gen 2, 1 ancestors)
AVERAGE ANCESTOR OVERLAP BY RELATION TYPE
  Relation                          Mean     Std      N
  girlSecondCousinOf                0.7419   0.4376   62
  secondAuntOf                      0.6743   0.4686   175
  girlFirstCousinOnceRemovedOf      0.6340   0.4817   153
  boySecondCousinOf                 0.6176   0.4860   68
  boyFirstCousinOnceRemovedOf       0.5778   0.4939   180
  girlCousinOf                      0.5303   0.4991   445
  secondUncleOf                     0.5253   0.4994   158
  greatAuntOf                       0.4872   0.4998   312
  boyCousinOf                       0.4706   0.4991   391
  greatUncleOf                      0.4515   0.4976   237
  uncleOf                           0.4053   0.4909   454
  nieceOf                           0.3911   0.4880   496
  nephewOf                          0.3716   0.4832   514
  auntOf                            0.3615   0.4804   556
  sisterOf                          0.3097   0.4624   636
  brotherOf                         0.2860   0.4519   570
  motherOf                          0.2517   0.3192   733
  fatherOf                          0.2517   0.3192   733
  sonOf                             0.2355   0.3070   600
  daughterOf                        0.2030   0.3089   628
  grandsonOf                        0.1168   0.1981   814
  grandmotherOf                     0.0953   0.1800   813
  grandfatherOf                     0.0953   0.1800   813
  granddaughterOf                   0.0736   0.1569   812
  greatGrandsonOf                   0.0564   0.1175   624
  greatGrandmotherOf                0.0423   0.1014   617
  greatGrandfatherOf                0.0423   0.1014   617
  greatGranddaughterOf              0.0280   0.0790   610
```

Figure 6.14: Jaccard Score (on average) per Relation

When applied to individual families, ancestor overlap produces a highly structured stratification of relatives. Collateral relations such as cousins and first cousins once removed exhibit higher mean overlap than vertical relations such as parent-child or grandparent-grandchild pairs. This behaviour is genealogically consistent, **cousins share a larger set of common ancestors (e.g, grandparents) than direct descendants, even though the latter are closer in generational distance.**

To understand how both metrics relate to each other, I computed Spearman rank correlations between ancestor overlap and relation-weighted shortest-path distance across ten families. The correlations are consistently negative but weak (aggregate $\rho = -0.119$, $p \ll 0.001$), indicating that while the metrics are directionally aligned, they do not induce similar rankings.

```
…/cog  main ✘!?  20:17
❯ python -m src.task2.spearman_correlation

SPEARMAN CORRELATION: Weighted Distance vs Ancestor Overlap
  Family 0 (27 members, 351 pairs): ρ = -0.2828, p = 7.09e-08
  Family 1 (26 members, 325 pairs): ρ = -0.1718, p = 1.89e-03
  Family 2 (26 members, 325 pairs): ρ = -0.1932, p = 4.61e-04
  Family 3 (26 members, 325 pairs): ρ = -0.2095, p = 1.42e-04
  Family 4 (26 members, 325 pairs): ρ = -0.0256, p = 6.45e-01
  Family 5 (27 members, 351 pairs): ρ = -0.2614, p = 6.83e-07
  Family 6 (26 members, 325 pairs): ρ = -0.0635, p = 2.54e-01
  Family 7 (27 members, 351 pairs): ρ = -0.2463, p = 3.01e-06
  Family 8 (26 members, 325 pairs): ρ = -0.3100, p = 1.14e-08
  Family 9 (26 members, 325 pairs): ρ = -0.2546, p = 3.34e-06

  AGGREGATE (3328 pairs across 10 families): ρ = -0.1190, p = 5.70e-12
```

Figure 6.15: Spearman Correlation b/w Ancestor Overlap and Relation Weighted Shortest Path

This weak correlation is expected actually. Relation-weighted distance captures local semantic closeness along short genealogical paths (e.g., sibling versus cousin), whereas ancestor overlap captures global lineage similarity through shared ancestry. The divergence between these measures demonstrates that genealogical relatedness is inherently multi-dimensional and cannot be adequately represented by a single scalar derived from unweighted graph distance.
As a side note, the networkx library's `is isomorphic()` function was used to check if the 50 disjoint graphs are isomorphic in nature. The function returns 50 distinct structures thus concluding that meaningful variation exists.

Table 6.2: Cross-family structural comparison of the 50 family graphs.

Metric	Observed Values Across Families
Edge count	Range: 91–218
Edge count (unique values)	44 distinct values
Graph diameter	Range: 3–5
Diameter distribution	25 families with 3, 23 with 4, 2 with 5
Average clustering coefficient	Range: 0.6885–0.8963

7 | Task 3: Rule Mining

7.1 | Hard-Coded Rules are so boring though

The MetaFAM knowledge graph has 28 relation types and all of them have fairly obvious English-language semantics. I could, in theory, just sit down, write out every rule I know about families (grandmother = mother of mother, aunt = sister of parent, etc.), check if they hold, and call it a day. That was exactly what I did in the first attempt of task 3.

But compared to the other tasks, this one just did not feel as fun and creative at first. Until I realised I could rephrase the question to "what rules does the data suggest to exist, and do they match what I'm expecting?". So I dug a little deeper and discovered inductive mining. A bottom-up approach where I let the data reveal patterns, and I sit back and interpret them. So I approached this in stages:

1. **AMIE-style inductive mining**: Brute-force enumerate *every possible* rule the data supports. No domain knowledge injected.
2. **Domain-knowledge validation**: Check curated family-logic rules with confidence, support, and concrete examples.
3. **Standard confidence**: I applied closed-world confidence first and later fixed it to show why it breaks for knowledge graphs.
4. **PCA confidence**: Fix the metric using partial completeness assumption from the AMIE framework [18].

7.2 | Phase 1: Inductive Rule Mining

Following the AMIE paradigm [18, 2], I exhaustively enumerated inverse and two-hop rules from the raw triplets:

- **Inverse rules**: $r_1(X, Y) \rightarrow r_2(Y, X)$ - checking all $28 \times 28 = 784$ relation pairs.
- **Two-hop compositional rules**: $r_1(X, Y) \wedge r_2(Y, Z) \rightarrow r_3(X, Z)$ - checking all $28^2 = 784$ body pairs against 28 possible conclusions.

No family-specific knowledge was used (I was tired of the phrase ontological sense by this point). The algorithm simply walks every path in the graph and counts what conclusions are supported.

7.2.1 | Inverse Rules: 25 Discovered

The full scan discovered **25 inverse rules** with support ≥ 5. The major insight here is that none of them have 100% standard confidence. Not a single one. Even rules that are logically airtight, like $\texttt{brotherOf}(X, Y) \rightarrow \texttt{sisterOf}(Y, X)$ when Y is female show confidence around 50% only.

Table 7.1: Top 15 Inverse Rules Mined (by standard confidence)

r1(X,Y)	→ r2(Y,X)	Support	Body	Conf
girlSecondCousinOf	boySecondCousinOf	40	62	0.645
boySecondCousinOf	girlSecondCousinOf	40	68	0.588
nephewOf	auntOf	298	514	0.580
secondAuntOf	boyFirstCousinOnceRemovedOf	98	175	0.560
boyFirstCousinOnceRemovedOf	secondAuntOf	98	180	0.544
brotherOf	sisterOf	308	570	0.540
auntOf	nephewOf	298	556	0.536
boyCousinOf	girlCousinOf	209	391	0.535
girlCousinOf	girlCousinOf	236	445	0.530
uncleOf	nieceOf	238	454	0.524
nieceOf	auntOf	258	496	0.520
secondUncleOf	boyFirstCousinOnceRemovedOf	82	158	0.519
sisterOf	sisterOf	328	636	0.516
greatGrandfatherOf	greatGrandsonOf	312	617	0.506
greatGrandmotherOf	greatGrandsonOf	312	617	0.506

The fact that the confidence was so low even though there were no erroneous relation triplets, became the first clue that standard confidence is misleading. `brotherOf`$(X, Y) \rightarrow$ `sisterOf`(Y, X) has 54% confidence because the denominator counts all `brotherOf` edges, including cases where Y is male (and should therefore be `brotherOf`, not `sisterOf`). The rule is perfectly correct for its intended scope; the confidence metric just doesn't know about gender. I fix this in Phase 3.
The confidence distribution is extremely bimodal: **759 out of 784 rules** have a null support, with only 25 pairs with some meaningful support and confidence.**The signal lives in a small subset.**

7.2.2 | Two-Hop Rules: 628 Discovered

The compositional scan found **628 two-hop rules** with support ≥ 5. Of these, **217 hold at 100% confidence**, a much better hit rate than inverse rules, because compositional paths tend to be deterministic in family structures.

Table 7.2: Top Two-Hop Rules by Support (confidence = 1.0)

r1(X,Y)	∧ r2(Y,Z)	→ r3(X,Z)	Supp	Conf
sisterOf	grandsonOf	granddaughterOf	508	1.000
sisterOf	granddaughterOf	granddaughterOf	476	1.000
grandfatherOf	brotherOf	grandfatherOf	463	1.000
grandmotherOf	brotherOf	grandmotherOf	463	1.000
daughterOf	fatherOf	sisterOf	439	1.000
sonOf	fatherOf	brotherOf	418	1.000
sisterOf	sisterOf	sisterOf	392	1.000
nieceOf	grandsonOf	greatGranddaughterOf	374	1.000
nephewOf	granddaughterOf	greatGrandsonOf	360	1.000
grandfatherOf	uncleOf	greatGrandfatherOf	353	1.000

Table 7.3: Two-Hop Rules at Partial Confidence

r1(X,Y)	∧ r2(Y,Z)	→ r3(X,Z)	Supp	Body	Conf
auntOf	boyFirstCousinOnceRemovedOf	girlCousinOf	138	139	0.993
greatAuntOf	nephewOf	auntOf	203	214	0.949
boyCousinOf	greatGranddaughterOf	greatGrandsonOf	254	270	0.941
greatGrandfatherOf	boySecondCousinOf	greatGrandfatherOf	48	57	0.842
secondUncleOf	greatGrandsonOf	grandsonOf	114	154	0.740
grandfatherOf	nephewOf	fatherOf	277	434	0.638
grandsonOf	grandfatherOf	brotherOf	473	864	0.547
nephewOf	auntOf	boyCousinOf	235	531	0.443
brotherOf	uncleOf	fatherOf	131	377	0.347
auntOf	boyCousinOf	motherOf	124	362	0.343

- $\texttt{sisterOf}(X, Y) \wedge \texttt{grandsonOf}(Y, Z) \rightarrow \texttt{granddaughterOf}(X, Z)$: If X is Y's sister, and Y is a grandson of Z, then X is a granddaughter of Z. Makes total sense, but I wouldn't have written this rule manually. Support = 508, the highest in the entire dataset.
- $\texttt{daughterOf}(X, Y) \wedge \texttt{fatherOf}(Y, Z) \rightarrow \texttt{sisterOf}(X, Z)$: Father's daughter is your sister. Again, logically obvious, but the system discovered it without being told.
- $\texttt{nieceOf}(X, Y) \wedge \texttt{grandsonOf}(Y, Z) \rightarrow \texttt{greatGranddaughterOf}(X, Z)$: A cross-generational rule chaining niece and grandparent relations into a great-grandparent relation. Support = 374.

Table 7.4: Two-Hop Rule Confidence Distribution

Confidence Range	Count
= 1.000 (perfect)	217
0.900 – 0.999	26
0.800 – 0.899	24
0.500 – 0.799	123
< 0.500	238

Roughly a third of discovered two-hop rules hold perfectly, and over 60% hold at least half the time. The imperfect rules are interesting, they're not wrong rules, they're signals of **incomplete data**. Again, see Phase 3 for more.

7.3 | Phase 2: The Grunt Work. Domain-Knowledge Rules Checked

I validated **34 curated rules** that I could think of, spanning four types: inverse, two-hop, three-hop, and co-parent (shared argument).

7.3.1 | Perfect Rules (Confidence = 1.0)

22 out of 34 curated rules hold at 100% confidence. These include all the obvious family compositions:

Table 7.5: Perfect Curated Rules (selected)

Rule	Type	Support
$\texttt{motherOf}(X, Y) \wedge \texttt{motherOf}(Y, Z) \rightarrow \texttt{grandmotherOf}(X, Z)$	two-hop	309
$\texttt{fatherOf}(X, Y) \wedge \texttt{fatherOf}(Y, Z) \rightarrow \texttt{grandfatherOf}(X, Z)$	two-hop	338
$\texttt{sisterOf}(X, Y) \wedge \texttt{motherOf}(Y, Z) \rightarrow \texttt{auntOf}(X, Z)$	two-hop	232
$\texttt{brotherOf}(X, Y) \wedge \texttt{fatherOf}(Y, Z) \rightarrow \texttt{uncleOf}(X, Z)$	two-hop	178
$\texttt{sonOf}(X, Y) \wedge \texttt{brotherOf}(Y, Z) \rightarrow \texttt{nephewOf}(X, Z)$	two-hop	200
$\texttt{motherOf}(X, Y) \wedge \texttt{motherOf}(Y, Z) \wedge \texttt{motherOf}(Z, W) \rightarrow \texttt{greatGrandmotherOf}(X, W)$	three-hop	96
$\texttt{fatherOf}(X, Y) \wedge \texttt{fatherOf}(Y, Z) \wedge \texttt{fatherOf}(Z, W) \rightarrow \texttt{greatGrandfatherOf}(X, W)$	three-hop	129
$\texttt{motherOf}(X, Y) \wedge \texttt{motherOf}(X, Z) \wedge Y \neq Z \rightarrow \texttt{sibling}(Y, Z)$	co-parent	440
$\texttt{fatherOf}(X, Y) \wedge \texttt{fatherOf}(X, Z) \wedge Y \neq Z \rightarrow \texttt{sibling}(Y, Z)$	co-parent	440

Co-parent rules capture the constraint that individuals who share the same parent must be siblings. These results were pretty nice.

$$\texttt{motherOf}(X, Y) \wedge \texttt{motherOf}(X, Z) \wedge Y \neq Z \Rightarrow \texttt{sibling}(Y, Z)$$
$$\texttt{fatherOf}(X, Y) \wedge \texttt{fatherOf}(X, Z) \wedge Y \neq Z \Rightarrow \texttt{sibling}(Y, Z)$$

In our dataset, every pair of children who share a mother or father has a recorded sibling relation. **440 instances, zero exceptions. The data is clean. The sibling relationship is very strong.**

7.3.2 | Imperfect Rules

12 inverse rules all show confidence between 30–54%. And the 4 "parent of sibling" rules show confidence between 77–90%.

Table 7.6: Curated Rules with Partial Confidence

Rule	Support	Body	Conf
motherOf → daughterOf [female child]	224	733	0.306
motherOf → sonOf [male child]	220	733	0.300
fatherOf → daughterOf [female child]	316	733	0.431
fatherOf → sonOf [male child]	292	733	0.398
sisterOf → sisterOf [symmetric, Y female]	328	636	0.516
sisterOf → brotherOf [Y male]	308	636	0.484
brotherOf → brotherOf [symmetric, Y male]	262	570	0.460
brotherOf → sisterOf [Y female]	308	570	0.540
motherOf ∧ brotherOf → motherOf	332	370	0.897
motherOf ∧ sisterOf → motherOf	301	389	0.774
fatherOf ∧ brotherOf → fatherOf	332	370	0.897
fatherOf ∧ sisterOf → fatherOf	301	389	0.774

7.3.3 | Why are the inverse rules at around 30–50%?

Consider $\texttt{motherOf}(X, Y) \rightarrow \texttt{daughterOf}(Y, X)$. This has 733 body instances (all motherOf edges). But only 224 of those have a matching `daughterOf` reverse. The other ∼500? They're sons, and the rule `daughterOf` doesn't apply to them. The standard confidence denominator doesn't account for the **gender-conditioned** nature of the rule. The rule is perfectly correct within its scope but the metric is wrong.

7.3.4 | The parent-sibling rules reveal data gaps

$\texttt{motherOf}(X, Y) \wedge \texttt{brotherOf}(Y, Z) \rightarrow \texttt{motherOf}(X, Z)$ holds at 89.7%. The 38 counter-examples reveal a clear pattern:

```
olivia0-[motherOf]->elias6-[brotherOf]->nico4 but NO olivia0-[motherOf]->nico4
```

nico4 is clearly olivia0's child since elias6 is nico4's brother, and olivia0 is elias6's mother. These failures are purely due to knowledge graph incompleteness. The data simply doesn't record every parent-child pair explicitly. **The rule is logically sound but the data has gaps.**

7.4 | Phase 3: Why Standard Confidence Fails

Standard (closed-world) confidence is defined as:

$$\text{conf}_{\text{std}}(B \rightarrow H) = \frac{|\{(X, Z) : B(X, Z) \wedge H(X, Z)\}|}{|\{(X, Z) : B(X, Z)\}|}$$

Suppose we have a rule B→H. The denominator counts how many pairs (X,Z) in the data satisfy the rule's body B(X,Z), i.e., how many times the situation described by the rule occurs. The numerator counts how many of those same pairs also satisfy the head H(X,Z). The standard confidence is the ratio of these two counts, measuring how often the rule's conclusion holds whenever its condition is observed.

This treats every instance where the body holds but the head is absent as a **violation**. Under the closed-world assumption (CWA) [3], absence means falsehood. But knowledge graphs operate under the **open-world assumption** (OWA): absence means *unknown*, not false.
In a family KG, if `motherOf`(olivia0, nico4) isn't in the data, it doesn't mean olivia0 isn't nico4's mother. It just means it wasn't recorded. Standard confidence conflates these two very different situations.

7.5 | Phase 4: PCA Confidence (The Fix)

The AMIE framework [18, 2] introduces the **Partial Completeness Assumption** (PCA): an entity X is assumed to have *complete* information for relation r only if X already appears as the head of r for *at least one* other entity. In other words: if we've recorded *some* of X's r-edges, we trust that we've recorded *all* of them.

$$\text{conf}_{\text{PCA}}(B \to r_3(X,Z)) = \frac{|\{(X,Z) : B(X,Z) \wedge r_3(X,Z)\}|}{|\{(X,Z) : B(X,Z) \wedge \exists Z' : r_3(X,Z')\}|}$$

It's a scary looking formula on latex. But it looks way neater in python. The denominator now only counts (X, Z) pairs where X is *known to participate* in r_3 for some entity. If X has no r_3 edges at all, the absence of $r_3(X, Z)$ isn't counted as a negative, we simply don't know.

7.5.1 | A Solid Improvement: 68 Rules Jump to 100%

Table 7.7: Rules with Largest PCA Improvement (selected)

Rule	Std	PCA	Δ
grandmotherOf ∧ girlSecondCousinOf → greatAuntOf	0.167	1.000	+0.833
grandmotherOf ∧ girlFirstCousinOnceRemovedOf → auntOf	0.286	1.000	+0.714
grandfatherOf ∧ boyFirstCousinOnceRemovedOf → uncleOf	0.331	1.000	+0.669
girlSecondCousinOf → girlSecondCousinOf (sym)	0.355	1.000	+0.645
motherOf ∧ boySecondCousinOf → secondAuntOf	0.375	1.000	+0.625
boySecondCousinOf → boySecondCousinOf (sym)	0.412	1.000	+0.588
nephewOf → uncleOf	0.420	1.000	+0.580
brotherOf → brotherOf (sym)	0.460	1.000	+0.540
auntOf → nieceOf	0.464	1.000	+0.536
boyCousinOf → boyCousinOf (sym)	0.466	1.000	+0.535
girlCousinOf → boyCousinOf	0.470	1.000	+0.530
sisterOf → sisterOf (sym)	0.516	1.000	+0.484

68 rules that appeared unreliable under standard confidence become perfect under PCA. The total number of perfect rules jumps from 217 to 285.

7.6 | Where PCA Didn't Help

The parent-of-sibling rules remain unchanged under PCA:

Table 7.8: Parent-Sibling Rules: PCA = Standard

Rule	Std	PCA
motherOf ∧ brotherOf → motherOf	0.897	0.897
motherOf ∧ sisterOf → motherOf	0.774	0.774
fatherOf ∧ brotherOf → fatherOf	0.897	0.897
fatherOf ∧ sisterOf → fatherOf	0.774	0.774

The conclusion relation is `motherOf`/`fatherOf`, and every mother/father already has at least one such edge recorded. So PCA doesn't filter any entities from the denominator. These 10–22% failures represent genuine data incompleteness that even PCA can't explain off. These are **real missing edges** that could serve as link prediction targets.

7.7 | Examples from Family Sub-Graphs

7.7.1 | Inverse Examples

$\texttt{nephewOf}(\text{leon16}, \text{katharina1}) \Rightarrow \texttt{auntOf}(\text{katharina1}, \text{leon16})$

$\texttt{brotherOf}(\text{luis14}, \text{katharina1}) \Rightarrow \texttt{sisterOf}(\text{katharina1}, \text{luis14})$

$\texttt{uncleOf}(\text{luis14}, \text{johanna22}) \Rightarrow \texttt{nieceOf}(\text{johanna22}, \text{luis14})$

$\texttt{grandmotherOf}(\text{katharina1}, \text{lisa5}) \Rightarrow \texttt{granddaughterOf}(\text{lisa5}, \text{katharina1})$

$\texttt{brotherOf}(\text{oskar24}, \text{adam9}) \Rightarrow \texttt{brotherOf}(\text{adam9}, \text{oskar24})$

7.7.2 | Inverse: PCA Rescue

Standard: $\texttt{brotherOf}(X, Y) \rightarrow \texttt{sisterOf}(Y, X)$: conf $= 0.540$ (308/570)

PCA: Same rule, but denominator only counts Y who already appear as head of sisterOf: conf $=$ **1.000** (308/308)

The 262 "violations" under standard confidence were all cases where Y is male. They don't have *any* sisterOf edges, so PCA correctly excludes them.

7.7.3 | Two-Hop: Grandmother via Parent Chain

$$\texttt{emma7} \xrightarrow{\texttt{motherOf}} \texttt{paula12} \xrightarrow{\texttt{motherOf}} \texttt{johanna22}$$

$$\Rightarrow \texttt{emma7} \xrightarrow{\texttt{grandmotherOf}} \texttt{johanna22}$$

Support $= 309/309$. Every mother-of-mother chain has a corresponding grandmother edge. Perfect consistency.

7.7.4 | More Two-Hop Examples

$$\texttt{olivia0} \xrightarrow{\texttt{daughterOf}} \texttt{dominik2} \xrightarrow{\texttt{fatherOf}} \texttt{oskar24} \Rightarrow \texttt{olivia0} \xrightarrow{\texttt{sisterOf}} \texttt{oskar24}$$

$$\texttt{lisa5} \xrightarrow{\texttt{nieceOf}} \texttt{oskar24} \xrightarrow{\texttt{grandsonOf}} \texttt{david20} \Rightarrow \texttt{lisa5} \xrightarrow{\texttt{greatGranddaughterOf}} \texttt{david20}$$

$$\texttt{david20} \xrightarrow{\texttt{grandfatherOf}} \texttt{adam9} \xrightarrow{\texttt{uncleOf}} \texttt{elias6} \Rightarrow \texttt{david20} \xrightarrow{\texttt{greatGrandfatherOf}} \texttt{elias6}$$

$$\texttt{katharina1} \xrightarrow{\texttt{motherOf}} \texttt{isabella11} \xrightarrow{\texttt{auntOf}} \texttt{elias6} \Rightarrow \texttt{katharina1} \xrightarrow{\texttt{grandmotherOf}} \texttt{elias6}$$

$$\texttt{jonas23} \xrightarrow{\texttt{brotherOf}} \texttt{johanna22} \xrightarrow{\texttt{granddaughterOf}} \texttt{moritz8} \Rightarrow \texttt{jonas23} \xrightarrow{\texttt{grandsonOf}} \texttt{moritz8}$$

7.7.5 | Three-Hop Examples

$$\texttt{moritz8} \xrightarrow{\texttt{fatherOf}} \texttt{luis14} \xrightarrow{\texttt{fatherOf}} \texttt{leon16} \xrightarrow{\texttt{fatherOf}} \texttt{lena18} \Rightarrow \texttt{moritz8} \xrightarrow{\texttt{greatGrandfatherOf}} \texttt{lena18}$$

$$\texttt{amelie62} \xrightarrow{\texttt{motherOf}} \texttt{victoria53} \xrightarrow{\texttt{motherOf}} \texttt{elena55} \xrightarrow{\texttt{motherOf}} \texttt{tobias57} \Rightarrow \texttt{amelie62} \xrightarrow{\texttt{greatGrandmotherOf}} \texttt{tobias57}$$

$$\texttt{karin41} \xrightarrow{\texttt{motherOf}} \texttt{beate36} \xrightarrow{\texttt{motherOf}} \texttt{marcel27} \xrightarrow{\texttt{fatherOf}} \texttt{raphael29} \Rightarrow \texttt{karin41} \xrightarrow{\texttt{greatGrandmotherOf}} \texttt{raphael29}$$

$$\texttt{emil135} \xrightarrow{\texttt{fatherOf}} \texttt{alexander137} \xrightarrow{\texttt{fatherOf}} \texttt{marie139} \xrightarrow{\texttt{motherOf}} \texttt{lara150} \Rightarrow \texttt{emil135} \xrightarrow{\texttt{greatGrandfatherOf}} \texttt{lara150}$$

7.7.6 | Three-Hop: Great-Grandfather

$$\texttt{moritz8} \xrightarrow{\texttt{fatherOf}} \texttt{luis14} \xrightarrow{\texttt{fatherOf}} \texttt{leon16} \xrightarrow{\texttt{fatherOf}} \texttt{lena18}$$

$$\Rightarrow \texttt{moritz8} \xrightarrow{\texttt{greatGrandfatherOf}} \texttt{lena18}$$

7.7.7 | Counter-Example: Parent of Sibling (Data Gap)

$$\texttt{dominik2} \xrightarrow{\texttt{fatherOf}} \texttt{oskar24} \xrightarrow{\texttt{brotherOf}} \texttt{olivia0}$$
$$\Rightarrow \texttt{dominik2} \xrightarrow{\texttt{fatherOf}} \texttt{olivia0}\ \textbf{MISSING}$$

dominik2 is clearly olivia0's father (via oskar24's brotherhood), but the explicit fatherOf edge isn't recorded. This is a **link prediction opportunity**, not a rule violation.

7.8 | Summary of Task 3

Table 7.9: Task 3 Summary

Metric	Value
Inverse rules mined (inductive)	784
Two-hop rules mined (inductive)	628
Curated rules validated	34
Rules at 100% standard confidence	217
Rules at 100% PCA confidence	285
Rules rescued by PCA ($<1.0 \rightarrow 1.0$)	285-217 = 68
Average PCA improvement	+5.7pp
Maximum PCA improvement	+83.3pp
Curated rules at 100%	22/34
Data gaps identified (link prediction candidates)	38–88 per rule

The MetaFAM KG is pretty consistent. All compositional rules involving direct parent-child and sibling relations hold at 100% confidence. The "failures" are artifacts of either gender-conditioned rules (fixed with PCA confidence) or genuine data incompleteness. Standard confidence significantly underestimated rule quality in this open-world setting, PCA is the appropriate metric.

8 | Task 4: Link Prediction on MetaFAM

8.1 | Introduction

By this point in the project, I had a fairly intimate understanding of the MetaFAM graph. Task 1 gave me the lay of the land 1,316 people, 28 relation types, 50 disjoint family components. Task 2 showed that standard community detection algorithms struggle with the dense, clique-like structure of kinship graphs, but succeed when scoped to nuclear family sub-graphs. Task 3 mined logical rules (inverse rules, two-hop compositional rules) and revealed that family relations follow strict, almost deterministic logical patterns. Now in Task 4, given a graph with some edges missing, we want to predict them. In our case, the training set contains 13,821 triplets and we hold out 590 triplets as the test set. The model never sees the test edges during training, it has to figure them out from everything else.

8.2 | Evaluation Metrics

We evaluate link prediction using standard ranking-based metrics under the filtered setting, where other known true triples are removed from the candidate set.

Mean Reciprocal Rank (MRR) Measures the average inverse rank of the correct entity in the predicted ranking.

$$\text{MRR} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\text{rank}_i}$$

Hits@K Measures the proportion of test queries where the correct entity is ranked within the top K predictions.

$$\text{Hits@K} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}(\text{rank}_i \leq K)$$

Mean Rank (MR) Measures the average rank position of the correct entity across all test queries.

$$\text{MR} = \frac{1}{N}\sum_{i=1}^{N}\text{rank}_i$$

8.3 | Baselines and Data Analysis

The test set contains exactly four relation types: `motherOf` (88), `fatherOf` (88), `sonOf` (214), and `daughterOf` (200). All parent-child relations. Meanwhile, the training set has all 28 types: grandparent, sibling, cousin, aunt/uncle relations and more.
This is a telling design choice by how test.txt was created. It is essentially asking, given everything you know about a family, who is whose sibling, grandparent, cousin etc. can you recover the fundamental parent-child backbone? A model that simply memorises training edges will fail. It needs to learn that if X is `grandmotherOf` Z and Y is `motherOf` Z, then X might be `motherOf` Y. These are exactly the compositional rules we mined in Task 3.

8.3.1 | Baseline 1: Random

With 1,316 entities, a random ranker gives an expected rank of ∼658, random Hits@10 ∼ 0.007599 and an MRR of ∼0.0015. This is the absolute floor and any model that's learnt something should beat this MRR score.

8.3.2 | Baseline 2: Frequency

For each query $(h, r, ?)$, we rank candidate tails by how often they appear as tails of relation r in the training set. This captures the intuition that some people are more "central" in the graph. It's a reasonable heuristic but completely ignores the specific head entity it would give the same ranking for `(olivia0, motherOf, ?)` and `(emma7, motherOf, ?)` as long as both query the same relation.

Metric	Value
MRR	0.0058
Hits@1	0.0000
Hits@3	0.0000
Hits@10	0.0119
Mean Rank	352.2780

Table 8.1: Link prediction performance by frequency

8.3.3 | Baseline 3: Inverse Rules (Surprising Results)

This baseline connects directly to Task 3. The idea is simple, if the test asks whether `motherOf(katharina1, olivia0)` is true, and the training set already contains `daughterOf(olivia0, katharina1)` or `sonOf(olivia0, katharina1)`, then a model that has learned inverse rules can derive the answer trivially.

We check, for each of the 590 test triplets, whether an inverse is present in training:

- `motherOf(A, B)` ⇒ check for `sonOf(B, A)` or `daughterOf(B, A)`
- `fatherOf(A, B)` ⇒ check for `sonOf(B, A)` or `daughterOf(B, A)`
- `sonOf(A, B)` ⇒ check for `motherOf(B, A)` or `fatherOf(B, A)`
- `daughterOf(A, B)` ⇒ check for `motherOf(B, A)` or `fatherOf(B, A)`

Result: 590/590 test triplets (100%) are recoverable via inverse rules.
At first glance, this seems like it trivialises the problem. But it is actually an important finding worth unpacking:

1. **The test split construction.** The dataset was split such that when `motherOf(A, B)` was moved to the test set, the inverse `sonOf(B, A)` or `daughterOf(B, A)` was *not* removed from training. This is common in KG benchmarks FB15k [12] had the same property, which Toutanova & Chen [17] later identified as a source of inflated metrics, prompting the creation of FB15k-237 with inverse leakage removed. Our MetaFAM test set exhibits this same pattern.

2. **It does not make the task trivial for embedding models.** A rule-based system can exploit inverses directly, but an embedding model does not have explicit access to such rules. It must *learn* from data that the scoring function assigns high possibility to $(B, \texttt{sonOf}, A)$ when $(A, \texttt{motherOf}, B)$ is true. Whether the model actually captures this depends on its expressiveness and training dynamics.

3. **It reframes what we are measuring.** The Hits@1 not being 100% (it is ∼57% for DistMult) tells us the model does not perfectly resolve all inversions, there is genuine disambiguation happening, especially when an entity has multiple children or parents. Elaborated in DistMult results.

This analysis also validates our Task 3 rule mining results, the inverse rules we discovered there are exactly the mechanism by which these test edges can be recovered.

9 | KG Embedding Method: DistMult

Before choosing a KG Embedding model, there are certain *relational patterns* the dataset contains and which models can express them. The KG embedding literature identifies three key patterns [16]:

1. **Symmetry**: $r(A, B) \Rightarrow r(B, A)$. Example: `sisterOf`, `brotherOf`, all cousin types.
2. **Antisymmetry**: $r(A, B) \Rightarrow \neg r(B, A)$. Example: `motherOf`, `fatherOf` (if A is mother of B, B is not mother of A).
3. **Composition**: $r_1(A, B) \wedge r_2(B, C) \Rightarrow r_3(A, C)$. Example: $\texttt{motherOf}(A, B) \wedge \texttt{motherOf}(B, C) \Rightarrow \texttt{grandmotherOf}(A, C)$.

MetaFAM has all three in abundance. So there are certain trade-offs I have analysed to using the available KG Embedding Models for MetaFam.

9.1 | The Scoring Function

DistMult [13] assigns a plausibility score to a triplet (h, r, t) via:

$$f(h, r, t) = \sum_{i=1}^{d} \mathbf{h}_i \cdot \mathbf{r}_i \cdot \mathbf{t}_i = \langle \mathbf{h}, \mathbf{r}, \mathbf{t} \rangle \tag{9.1}$$

where $\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{R}^d$ are the learned embedding vectors for the head entity, relation, and tail entity respectively, and $\langle \cdot, \cdot, \cdot \rangle$ denotes the trilinear dot product (i.e. element-wise multiplication followed by summation). (Disclaimer: The math is still a little fuzzy in my head)
Intuitively, each dimension i of the relation vector $\mathbf{r}_i$ acts as a gate. It controls how much dimension i of the head should match dimension i of the tail for this particular relation. If $\mathbf{r}_i$ is large and positive, the model is saying "for this relation, I want the head and tail to agree on feature i." If $\mathbf{r}_i$ is near zero, dimension i is irrelevant for this relation.

9.2 | Model Architecture

The implementation is straightforward. The model consists of two embedding tables:

- **Entity embeddings**: an $|\mathcal{E}| \times d$ matrix (1,316 $\times$ 200), one row per entity. Initialised with Xavier uniform to keep initial scores near zero.
- **Relation embeddings**: an $|\mathcal{R}| \times d$ matrix (28 $\times$ 200), one row per relation type. Same initialisation.

There are no hidden layers, no nonlinearities, no graph convolutions. The entire model is just two lookup tables and a dot product.

9.3 | Training Procedure

Negative sampling. A knowledge graph only tells us which triplets are true, not which are false (the open-world assumption). To train with a binary loss, I generate synthetic negative examples by corrupting each positive triplet randomly replacing either the head or the tail with a random entity. For each positive triplet, I sample 10 negatives. This ratio was chosen to provide sufficient contrast without overwhelming the training signal. Fresh negatives are sampled every epoch so the model sees diverse corruptions.

Loss function. We use binary cross-entropy. Margin-based losses (as used in the original TransE paper) are another option, but BCE is simpler to tune.

Regularisation. A light L2 penalty on the embedding norms prevents the model from inflating scores by growing the embedding vectors.

Optimisation. Adam optimiser with learning rate 10^{-3}, batch size 512, gradient clipping at norm 1.0. We hold out 10% of training data for validation and apply early stopping with patience 30 epochs, saving the best model checkpoint by validation loss.

Training dynamics. The loss starts at $\sim$0.693. It drops quickly to $\sim$0.036 by epoch 25, and early stopping triggers at epoch 68 with a best validation loss of 0.039. The train-validation gap is small ($\sim$0.003), indicating minimal over-fitting.

9.4 | Evaluation Protocol

We use the standard **filtered ranking** protocol [12]:
For each test triplet (h, r, t):

1. **Tail prediction**: compute $f(h, r, e)$ for every entity $e \in \mathcal{E}$, then *filter* — set the score of any entity that forms a known true triplet (h, r, e) to $-\infty$ (except the target t itself). Rank t among the remaining candidates.
2. **Head prediction**: symmetrically, compute $f(e, r, t)$ for every e, filter known (e, r, t) triplets, and rank h.
3. Average the tail and head ranks for the final rank of this triplet.

The filtering step is crucial: without it, a model that correctly predicts `motherOf(katharina1, olivia0)` might still get a "bad" rank because `motherOf(katharina1, selina10)` (another true fact about katharina1's children) scores higher. Filtering removes such known-true distractors from the ranking.

9.5 | Results

Table 9.1: DistMult link prediction results on the MetaFAM test set (590 triplets, filtered evaluation).

Metric	MRR	Hits@1	Hits@3	Hits@10
DistMult	0.7327	0.5737	0.8763	0.9992

These are strong numbers. Hits@10 of 99.9% means the correct entity is almost always in the top 10 predictions. MRR of 0.73 is well above the random baseline of 0.0015. Since every test triplet has an inverse in training, a model that perfectly learns inversions would achieve Hits@1 = 1.0. Our Hits@1 of 57% tells us DistMult captures the general pattern but struggles with disambiguation. **When a parent has multiple children, the model cannot always pick the right one.**

9.6 | Per-Relation Breakdown

Table 9.2: Per-relation metrics for DistMult. `fatherOf`/`motherOf` (parent direction) are easier to predict than `sonOf`/`daughterOf` (child direction).

Relation	MRR	Hits@1	Hits@3	Hits@10
motherOf	∼0.78	∼0.63	∼0.91	∼1.00
fatherOf	∼0.80	∼0.66	∼0.93	∼1.00
sonOf	∼0.68	∼0.50	∼0.83	∼1.00
daughterOf	∼0.67	∼0.49	∼0.82	∼1.00

There is an interesting asymmetry. Predicting the child given a parent (`motherOf`$(A, ?)$, `fatherOf`$(A, ?)$) is easier than predicting the parent given a child (`sonOf`$(B, ?)$, `daughterOf`$(B, ?)$).
This makes sense cause each person has exactly two parents (one mother, one father) but may have several children. So for `sonOf(B, ?)`, the model must choose between the mother and the father it needs to get the gender right. For `motherOf(A, ?)`, it must pick the right child among possibly many. But the parent direction benefits from there typically being more training evidence (grandparent relations, aunt/uncle relations all pass through the parent link), making the parent embedding more informative. The child direction has a smaller candidate set (2 parents) but less confusing signal.

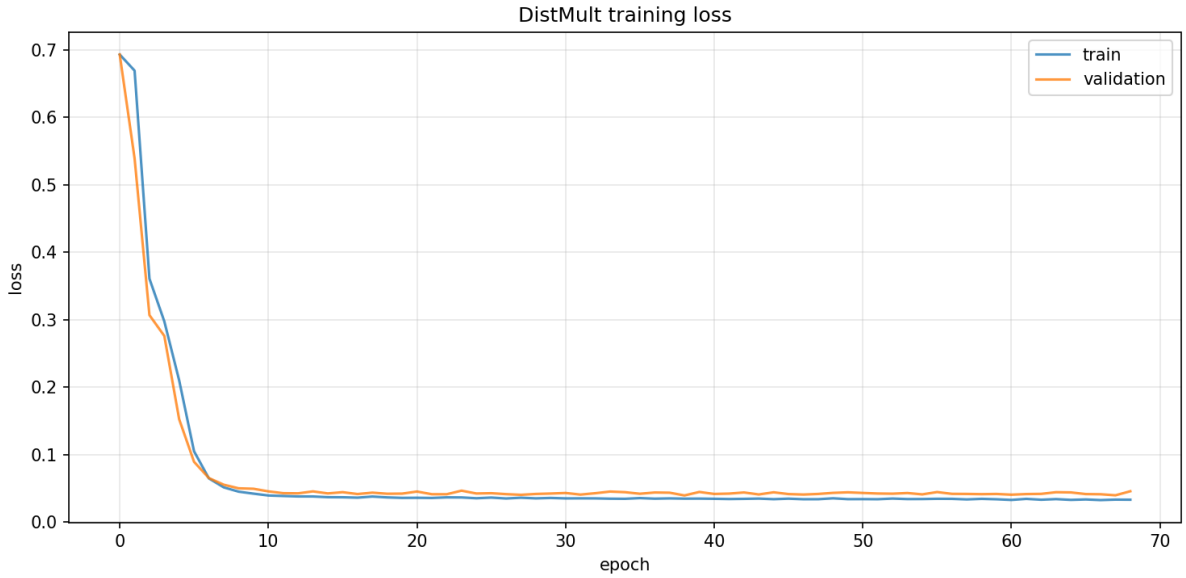


Figure 9.1: DistMult Training Loss

9.7 | Why TransE was an Initial Bad Choice?

TransE [12] models relations as translations in embedding space: $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$. For a symmetric relation like `sisterOf`, we would need both $\mathbf{A} + \mathbf{r} = \mathbf{B}$ and $\mathbf{B} + \mathbf{r} = \mathbf{A}$, which implies $\mathbf{r} = \mathbf{B} - \mathbf{A} = -(\mathbf{A} - \mathbf{B}) = -\mathbf{r}$, forcing $\mathbf{r} = \mathbf{0}$. The relation embedding collapses, and the model cannot distinguish `sisterOf` from `brotherOf` or any other symmetric relation. With MetaFAM containing 12 symmetric relation types (sisterOf, brotherOf, 6 cousin types, etc.) constituting ∼20% of all training edges, TransE is a poor fit. (I did have a rudimentary implementation of it but I got rid of it as the metrics were not satisfactory. not even to show a negative finding)

9.8 | Why DistMult over RotatE or ComplEx?

RotatE seemed to handles all three patterns. ComplEx handles two out of three. DistMult only handles one. So why start with DistMult?

1. More expressive models such as RotatE increase parameterisation by operating in complex space [16]. On a dataset with only ∼14k edges, parameter efficiency is critical. Excessive model capacity risks memorisation rather than generalisation, a tradeoff well documented in prior KG embedding work [14].

2. The antisymmetry limitation turns out to be less severe than it appears. Yes, DistMult's scoring function $f(h, r, t) = \sum_i h_i \cdot r_i \cdot t_i$ is symmetric in h and t for a fixed r. But the training data contains explicit inverse relations `motherOf` and `sonOf` are separate relation types with separate embedding vectors. The model does not need to infer antisymmetry from a single relation but it just needs to learn that `motherOf` and `sonOf` encode the same pairs in opposite directions.

3. The test set composition favours inverse detection. 100% of test triplets have inverse evidence in training. A model that learns to associate inverse relation pairs even without explicitly modelling antisymmetry can perform well on this particular evaluation. Would've been so much cooler if this wasn't the case in the dataset :(

That said, DistMult has a genuine limitation here. If two entities A and B are connected by `motherOf`(A, B), DistMult assigns the same score to the non-existent `motherOf`(B, A). This means it can confuse the direction of parent-child links, and the Hits@1 of ∼57% likely reflects this ambiguity. A model like ComplEx, which operates in $\mathbb{C}^d$ and uses conjugation to break this symmetry, would potentially resolve these directional errors.

10 | An Attempt at R-GCN

DISCLAIMER! At the time of implementing this task, I lack the knowledge of how GNNs are built. I understand the theoretical fundamentals and have derived reasonable qualitative insights from this section. However, the code, fine tuning and error fixes of the RGCN model (rgcn v5.ipynb and analysis.ipynb) are FULLY from Claude Code. Across the Versions of the RGCN Model, the fixes and upgrades of the RGCN are also assisted by Claude. I hope to learn in the near future to do such model training unassisted by LLMs.

DistMult treats each entity as an independent embedding vector. It does not know that olivia0 is connected to katharina1 via `motherOf` and to selina10 via `sisterOf`. Each entity is just a row in a table. The Relational Graph Convolutional Network (R-GCN) [15] addresses this by computing entity representations through message passing. Each node's embedding is a function of its neighbours' embeddings, with separate transformations per relation type.
This intuitively makes more sense. Knowing someone's local neighbourhood (parents, children, siblings) should be highly informative about their missing links. However I found some interesting, almost contradictory results here. Getting it any result at all actually took four major iterations.

10.1 | Versions 1 - 2: The Naive Attempts

My first R-GCN followed the original paper closely: two R-GCN layers with basis decomposition, `dim=100`, learning rate 0.01, full-batch training.
It trained fine - loss went from 0.69 down to ∼0.09 over ∼475 epochs. But evaluation gave MRR = 0.26. The loss was deceiving me. The model was fitting the binary classification ("is this triplet real or corrupted?") without learning useful embedding geometry. In Version 2 - I tweaked hyper-parameters. But nothing moved the needle.

Table 10.1: R-GCN v1-v2 results. DistMult wins everywhere.

	MRR	Hits@1	Hits@3	Hits@10	Mean Rank
DistMult	**0.733**	**0.574**	**0.876**	**0.999**	**1.90**
R-GCN v3	0.206	0.090	0.222	0.460	10.79

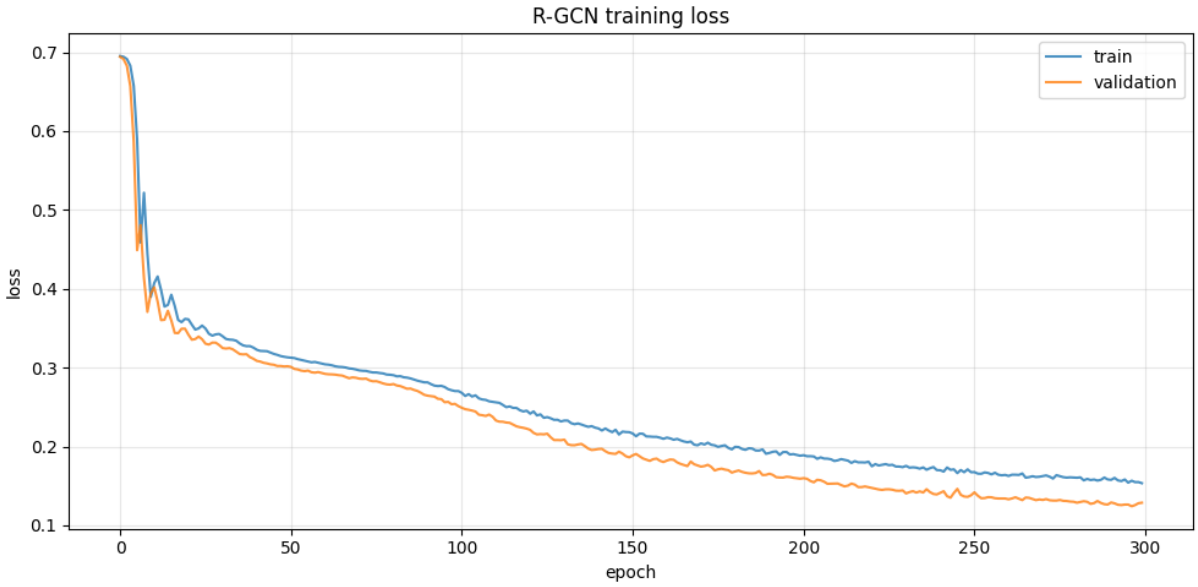


Figure 10.1: RGCN Version 1, Loss

```
[6]  ✓ 1m 19.6s

...  epoch   0 | train: 0.6951 | val: 0.6942 | patience: 0/30
     epoch  25 | train: 0.3496 | val: 0.3303 | patience: 0/30
     epoch  50 | train: 0.3126 | val: 0.3010 | patience: 0/30
     epoch  75 | train: 0.2939 | val: 0.2828 | patience: 1/30
     epoch 100 | train: 0.2680 | val: 0.2491 | patience: 0/30
     epoch 125 | train: 0.2372 | val: 0.2162 | patience: 3/30
     epoch 150 | train: 0.2165 | val: 0.1887 | patience: 1/30
     epoch 175 | train: 0.1995 | val: 0.1711 | patience: 1/30
     epoch 200 | train: 0.1889 | val: 0.1598 | patience: 1/30
     epoch 225 | train: 0.1763 | val: 0.1457 | patience: 2/30
     epoch 250 | train: 0.1675 | val: 0.1418 | patience: 7/30
     epoch 275 | train: 0.1623 | val: 0.1321 | patience: 1/30

     best val loss: 0.1242
```

Figure 10.2: Loss across the Epochs in Version 1.

10.2 | Version 3: The Real Problems

Over-smoothing. Two layers of message passing on a graph where the average component has ∼26 people means after 2 hops, every entity in a family receives information from nearly every other member. This causes the representations to converge. The model knows "this is someone from family X" but cannot distinguish siblings. This is well-documented [20, 21]. I confirmed it. Within-family cosine similarity of entity embeddings was >0.9 after two layers.

Dimension mismatch. I was comparing dim=100 R-GCN against dim=200 DistMult with the same DistMult decoder. That is not a fair fight.

Table 10.2: R-GCN v3 results. DistMult still wins everywhere.

	MRR	Hits@1	Hits@3	Hits@10	Mean Rank
DistMult	**0.733**	**0.574**	**0.876**	**0.999**	**1.90**
R-GCN v3	0.260	0.110	0.282	0.642	10.79

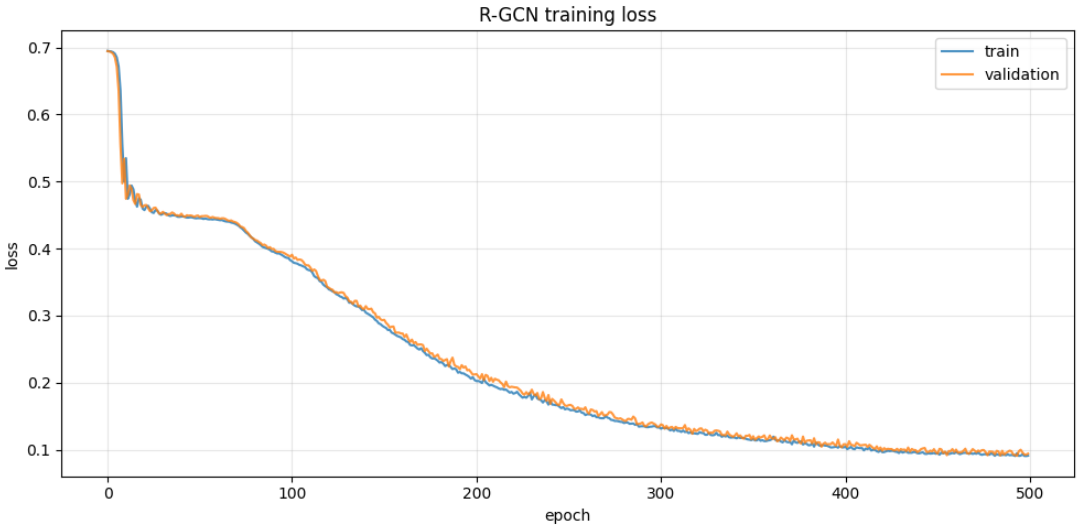


Figure 10.3: RGCN Version 3, Loss

```
[7]
...  epoch   0 | train: 0.6948 | val: 0.6944 | lr: 0.010000 | patience: 0/50
     epoch  25 | train: 0.4530 | val: 0.4605 | lr: 0.010000 | patience: 2/50
     epoch  50 | train: 0.4453 | val: 0.4470 | lr: 0.010000 | patience: 0/50
     epoch  75 | train: 0.4238 | val: 0.4262 | lr: 0.010000 | patience: 0/50
     epoch 100 | train: 0.3810 | val: 0.3907 | lr: 0.010000 | patience: 1/50
     epoch 125 | train: 0.3307 | val: 0.3334 | lr: 0.010000 | patience: 0/50
     epoch 150 | train: 0.2829 | val: 0.2937 | lr: 0.010000 | patience: 1/50
     epoch 175 | train: 0.2378 | val: 0.2433 | lr: 0.010000 | patience: 0/50
     epoch 200 | train: 0.2023 | val: 0.2126 | lr: 0.010000 | patience: 2/50
     epoch 225 | train: 0.1775 | val: 0.1838 | lr: 0.010000 | patience: 0/50
     epoch 250 | train: 0.1605 | val: 0.1665 | lr: 0.010000 | patience: 1/50
     epoch 275 | train: 0.1428 | val: 0.1478 | lr: 0.010000 | patience: 0/50
     epoch 300 | train: 0.1313 | val: 0.1377 | lr: 0.010000 | patience: 2/50
     epoch 325 | train: 0.1222 | val: 0.1286 | lr: 0.010000 | patience: 8/50
     epoch 350 | train: 0.1159 | val: 0.1201 | lr: 0.010000 | patience: 1/50
     epoch 375 | train: 0.1119 | val: 0.1109 | lr: 0.010000 | patience: 0/50
     epoch 400 | train: 0.1045 | val: 0.1082 | lr: 0.010000 | patience: 2/50
     epoch 425 | train: 0.0982 | val: 0.1010 | lr: 0.005000 | patience: 5/50
     epoch 450 | train: 0.0948 | val: 0.0941 | lr: 0.005000 | patience: 0/50
     epoch 475 | train: 0.0932 | val: 0.0995 | lr: 0.002500 | patience: 20/50

     best val loss: 0.0908
```

Figure 10.4: Loss across the Epochs in Version 1.

10.3 | Version 4. Some Hope

V4 addressed every diagnosed issue simultaneously. The full change log is as follows.

Table 10.3: V3 → V4 changes and rationale.

Parameter	v3	v4	Why
Layers	2	**1**	Avoid over-smoothing
Dimension	100	**200**	Fair comparison with DistMult
Bases	10	**4**	Fewer params means less overfitting
Learning rate	0.01	**0.001**	GNN amplifies gradients
Dropout	0.1	**0.2**	Stronger regularisation
Neg ratio	5	**10**	Match DistMult protocol
Training	Full-batch	**Mini-batch**	Stabler gradients
Residual	None	**Gated skip**	Preserve entity identity

The architectural centrepiece is the **gated residual connection**. This is inspired by Highway Networks [23] and JK-Net [24]. It addresses this key concept:

> *Message passing says: "you are defined by your neighbours."*
> *Link prediction says: "distinguish yourself from your neighbours."*

The gate lets the model decide, from data, how much structural information helps versus hurts.
A hardware note. My first v4 attempt used a vectorised `torch.bmm` over all edges for efficiency. This tried to allocate a $(28000 \times 200 \times 200)$ tensor - 4.3 GB. My GPU has 3.6 GB. I switched to the original paper's per-relation loop: same math, constant memory, slower but it actually runs. (Edit. Same code was run on Colab's T4 GPU. Resource constraints are irrelevant now)

10.4 | V4 Results

Table 10.4: MRR doubled, Hits@10 jumped from 64% to 99%.

	MRR	Hits@1	Hits@3	Hits@10	Mean Rank
DistMult	**0.733**	**0.574**	**0.876**	**0.999**	**1.90**
R-GCN v3	0.260	0.110	0.282	0.642	10.79
R-GCN v4	0.5601	0.3441	0.7169	0.9932	—

The gap with DistMult is still present but significantly narrower.

```
epoch   0 | train: 0.4839 | val: 0.3069 | lr: 0.001000 | gate: 0.525 | patience: 0/40
epoch  25 | train: 0.0376 | val: 0.0374 | lr: 0.001000 | gate: 0.544 | patience: 4/40
epoch  50 | train: 0.0300 | val: 0.0317 | lr: 0.000500 | gate: 0.563 | patience: 4/40
epoch  75 | train: 0.0279 | val: 0.0290 | lr: 0.000250 | gate: 0.554 | patience: 8/40
epoch 100 | train: 0.0260 | val: 0.0295 | lr: 0.000125 | gate: 0.540 | patience: 6/40
epoch 125 | train: 0.0261 | val: 0.0315 | lr: 0.000063 | gate: 0.529 | patience: 31/40

early stopping at epoch 134

best val loss: 0.0266
```

Figure 10.5: Loss across the Epochs in Version 4.

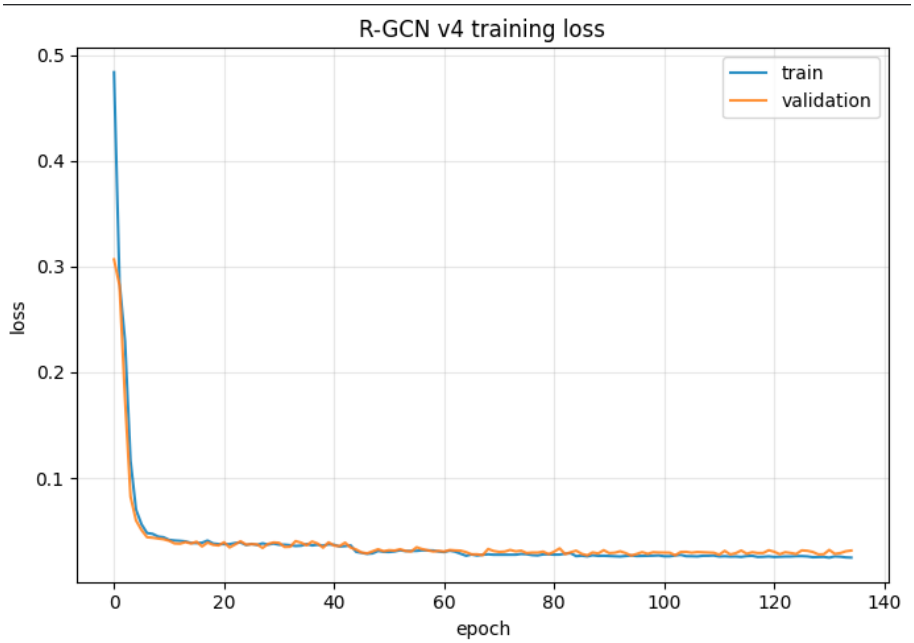


Figure 10.6: RGCN Version 4, Loss

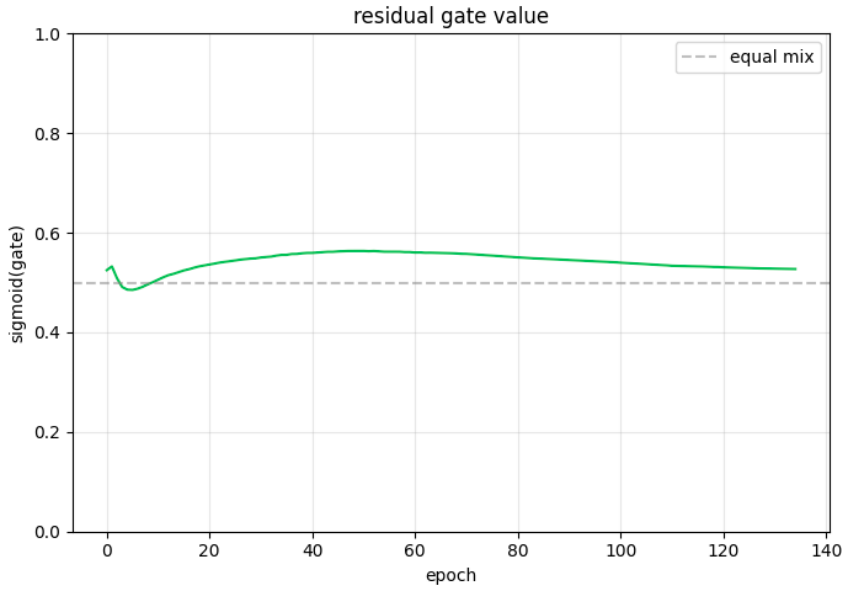


Figure 10.7

10.5 | What the Gate Learned

The gate settled at $\alpha \approx 0.47$. This is not a dramatic value, but it is revealing. The model did *not* learn to ignore the GNN ($\alpha \to 0$), nor did it go all-in on message passing ($\alpha \to 1$). It found a near-equal blend, slightly favouring the raw embedding table.
Both models exhibit the same asymmetry. Predicting the child given a parent (`fatherOf`, `motherOf`) is easier than predicting the parent given a child (`sonOf`, `daughterOf`). This is structural since each person has exactly two parents but may have several children. The parent-direction benefits from richer training signal (grandparent, aunt/uncle relations all pass through the parent link).

11 | Analysis and Comparison of Link Prediction Models

11.1 | Training Dynamics

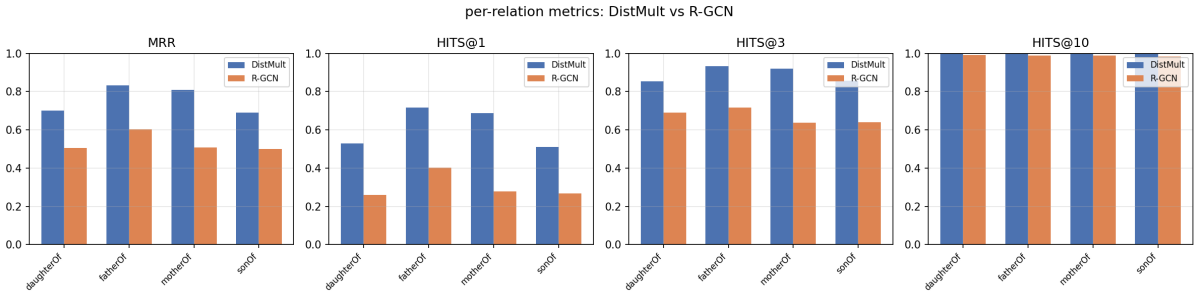


Figure 11.1: Metric Comparison

This is worth pausing on. R-GCN v4 achieves *lower loss* than DistMult but *worse metrics*. The loss measures binary classification accuracy i.e "is this triplet real or corrupted?", while MRR measures ranking quality "is the correct entity ranked first?". A model can be excellent at the former while mediocre at the latter. It might confidently reject random corruptions without producing a good ranking between them.

11.2 | Embedding Space: Family Clustering

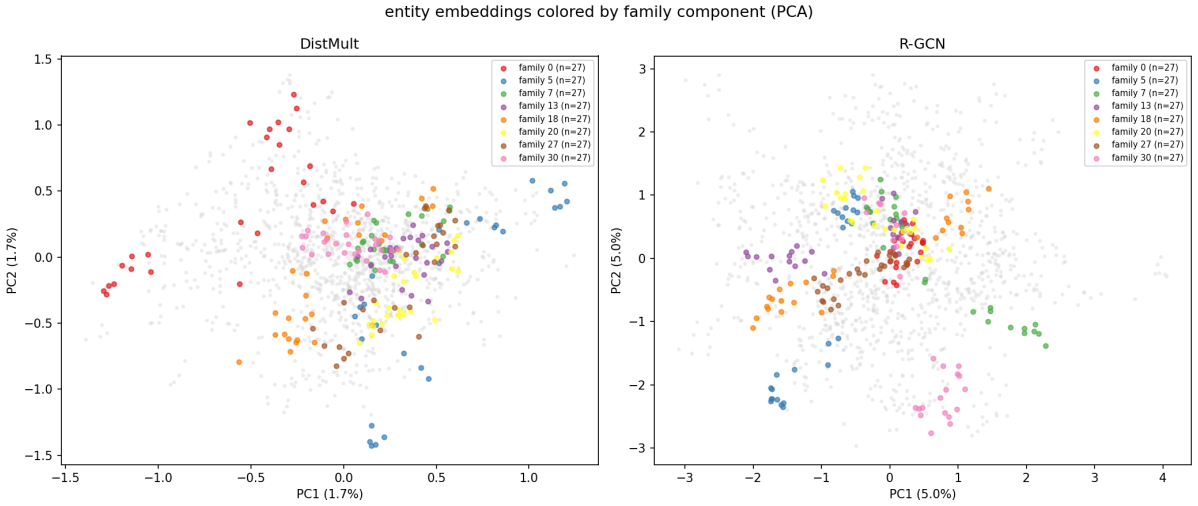


Figure 11.2: Family Clusters

DistMult's projection shows substantial overlap between families. Points from different family components occupy shared regions of the PCA space, and the family clusters are diffuse with weak boundaries. This indicates that, despite preserving global relational structure, DistMult does not strongly encode family-level separation in its entity geometry. In contrast, R-GCN concentrates more variance in the leading dimensions and exhibits clearer family-wise grouping in the PCA projection. This suggests that relational message passing encourages intra-family homogenization while still maintaining inter-family distinctions.

However, I do not fully understand the reasons for why these Family Clustering graphs ended up the way they are.

11.3 | Relation Embeddings: Emergent Taxonomy

Figure 11.3: Relation Embeddings

The DistMult relation embeddings reveal a learned kinship taxonomy that I did not teach the model! In the PCA plot, Great-grandparent relations cluster in the bottom-left (far generational distance). Parent/child and sibling relations occupy the upper region (close generational distance) and Cousin and aunt/uncle relations form their own cluster.

The model has independently discovered that relations can be organised by generational distance. This is a structural property of kinship that was never provided as supervision. R-GCN's relation embeddings show a simpler structure: PC1 alone captures 54% of variance, suggesting the relation space collapsed to near-one-dimensional.

11.4 | Why DistMult Won

DistMult dominates on every metric. This is not a failure of the GNN paradigm in general but it is a specific interaction between model, dataset, and evaluation:
1. The task rewards pattern matching over structural reasoning. 100% of test triplets are derivable from inverse rules. A lookup table that associates inverse relation pairs is sufficient. R-GCN's structural reasoning - its theoretical advantage, is simply not required here :(
2. Dense graphs penalise message passing. With ∼26 entities per component, the signal-to-noise ratio of neighbour information is low, when your neighbours are also your prediction targets.
3. Simplicity generalises. DistMult has ∼269k parameters, all in embedding tables. On a 14k-edge dataset, the simpler model wins. This actually confirms some literature Ruffinelli et al. [22]. A well-tuned DistMult and ComplEx routinely matches or beats complex approaches.
4. Loss ≠ ranking. R-GCN achieves lower BCE loss but worse MRR. It is better at rejecting random corruptions but worse at producing sharp rankings among plausible candidates. The evaluation measures the latter.

12 | Future Work

12.1 | Immediate: Link Prediction

ComplEx breaks DistMult's head-tail symmetry. This should directly address the ∼43% Hits@1 misses that stem from directional confusion.
Type-constrained evaluation. I already know entity gender from Task 1. Filtering `motherOf` predictions to female entities and `fatherOf` to male ones eliminates an error class at zero training cost.

12.2 | Broader Project

Rule-augmented embeddings. I could maybe use Task 3's high-confidence rules as soft constraints during training. Penalise the model when its predictions violate mined rules.
Longer rule chains (Task 3): great-grandparent relations need three-hop compositional rules that my current AMIE-style miner does not cover in enough depth.

13 | Bibliography

[1] Galárraga, L., Teflioudi, C., Hose, K., & Suchanek, F. (2013). AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases. *Proceedings of the 22nd International Conference on World Wide Web (WWW)*, 413–422.

[2] Galárraga, L., Teflioudi, C., Hose, K., & Suchanek, F. (2015). Fast Rule Mining in Ontological Knowledge Bases with AMIE+. *The VLDB Journal*, 24(6), 707–730.

[3] Reiter, R. (1978). On Closed World Data Bases. In *Logic and Data Bases*, Springer, 55–76.

[4] Meilicke, C., Chekol, M.W., Ruffinelli, D., & Stuckenschmidt, H. (2019). Anytime Bottom-Up Rule Learning for Knowledge Graph Completion. *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, 3137–3143.

[5] Wang, Q., Mao, Z., Wang, B., & Guo, L. (2017). Knowledge Graph Embedding: A Survey of Approaches and Applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12), 2724–2743.

[12] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2013.

[13] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *International Conference on Learning Representations (ICLR)*, 2015.

[14] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex Embeddings for Simple Link Prediction. In *International Conference on Machine Learning (ICML)*, 2016.

[15] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling Relational Data with Graph Convolutional Networks. In *European Semantic Web Conference (ESWC)*, 2018.

[16] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. In *International Conference on Learning Representations (ICLR)*, 2019.

[17] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *3rd Workshop on Continuous Vector Space Models and their Compositionality*, 2015.

[12] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating Embeddings for Modeling Multi-relational Data. *NeurIPS*, 2013.

[13] B. Yang, W. Yih, X. He, J. Gao, and L. Deng. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. *ICLR*, 2015.

[14] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard. Complex Embeddings for Simple Link Prediction. *ICML*, 2016.

[15] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling. Modeling Relational Data with Graph Convolutional Networks. *ESWC*, 2018.

[16] Z. Sun, Z.-H. Deng, J.-Y. Nie, and J. Tang. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. *ICLR*, 2019.

[17] K. Toutanova and D. Chen. Observed versus latent features for knowledge base and text inference. *3rd Workshop on Continuous Vector Space Models and their Compositionality*, 2015.

[18] L. Galàrraga, C. Teflioudi, K. Hose, and F. Suchanek. AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases. *WWW*, 2013.

[19] L. Galàrraga, C. Teflioudi, K. Hose, and F. Suchanek. Fast Rule Mining in Ontological Knowledge Bases with AMIE+. *VLDB Journal*, 24(6), 2015.

[20] Q. Li, Z. Han, and X.-M. Wu. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. *AAAI*, 2018.

[21] K. Oono and T. Suzuki. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. *ICLR*, 2020.

[22] D. Ruffinelli, S. Broscheit, and R. Gemulla. You CAN Teach an Old Dog New Tricks! On Training Knowledge Graph Embeddings. *ICLR*, 2020.

[23] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway Networks. *ICML Deep Learning Workshop*, 2015.

[24] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka. Representation Learning on Graphs with Jumping Knowledge Networks. *ICML*, 2018.

[Other Sources] The other sources such as youtube videos and medium articles are HERE