A Community-driven Hub for High-Quality Text2Cypher Benchmarks and Resources

Supplementary Material

Emanuele Cavalleri, Nada Bou Kanaan, Marco Mesiti

This Supplementary Material is organized as follows. Section S1 provides additional structural and schema details for miRNA-KG, the Russian Twitter Trolls, and the Bloom graph. Section S2 provides some details on the benchmarks' distribution. Section S3 reports representative examples of NL question–Cypher pairs across the five query categories within *bio2C*, *twt2C*, and *fin2C*. Section S4 presents a complete `S+RAG+O` prompt example. Finally, Section S5 reports the detailed performance of the RAG pipelines across evaluation metrics and query categories.

S1 Details on the property graphs

miRNA-KG. miRNA-KG is the biomedical PG used to construct *bio2C*. It contains approximately 99K nodes and 1.6M relationships. Figure S1a reports the distribution of the main node labels. Gene Ontology concepts constitute the largest group, accounting for 41.2% of the nodes (41K), followed by diseases (26.3%, 26K) and phenotypes (19.9%, 20K). Figure S2 reports the distribution of the principal miRNA, GO, and disease subtypes represented through multiple node labels. RNAs include approximately 3K mature miRNAs and 2K pre-miRNAs. GO terms are predominantly biological processes (27K), followed by molecular functions (10K) and cellular components (4K). The disease component is heterogeneous: in addition to approximately 18K diseases not assigned to specialized subtypes, the graph contains cancers (5K), infectious diseases (1K), neurodegenerative diseases (944), cardiovascular diseases (752), autoimmune diseases (473), and mental diseases (12).

As shown in Figure S1b, phenotype-related relationships dominate the graph. The inverse relationship pair `has_phenotype`/`phenotype_of` accounts for 63.8% of all relationships, corresponding to approximately 1M edges. Subclass relationships represent a further 9.4% (152K). Figure S1c presents the distribution of node and relationship properties. Relationship properties are the dominant component. `Source` accounts for 63.3% of all property occurrences (1.6M). `URI`, `ID`, and `Label` are the most frequent node properties, each occurring on $\sim$99K nodes.

Finally, Figure S3 illustrates the schema of the selected view. The schema connects RNA entities with genes, diseases, phenotypes, Gene Ontology concepts, and specialized disease classes. Although RNA entities form a comparatively small fraction of the graph (4.8%), they occupy a central position in the schema and are connected to multiple biomedical entity classes. It also contains multiple semantically distinct relationship types between the same classes of nodes, as well as inverse relationship pairs.

Russian Twitter Trolls. The Russian Twitter Trolls (RTT) graph is the social-media PG used to construct *twt2C*. It contains approximately 281K nodes and 493K relationships. Figure S4a reports the distribution of node labels. `Tweet` is the most frequent label, accounting for 82.4% of the nodes (232K). The `Troll` label represents a subtype of `User`: all 453 troll accounts also carry the `User` label. As shown in Figure S4b, `POSTED` is the most frequent relationship type, accounting for 40.7% of all relationships (201K).

Figure S4c presents the distribution of node properties (RTT contains no relationship properties). The identifier property `id` is the most frequent, occurring approximately 233K times (21.9%). The properties `text`, `created_at`, and `created_str` each occur approximately 201K times (18.9%), reflecting their broad presence on tweet nodes.

Bloom graph. The Bloom graph is the financial PG used to construct *fin2C*. It contains approximately 31K nodes and 30K relationships. Figure S5a reports the distribution of node labels. `BankAccount` and `Address` are the most frequent labels, each accounting for approximately 16.3% of the nodes (5K). Multiple labels are almost absent: only one node carries both the `AccountHolder` and `Flagged` labels. As shown in Figure S5b, `FROM` is the most frequent relationship type, accounting for 17.0% of all relationships (5K).

Figure S5c presents the distribution of node and relationship properties. `accountNumber` and `balance` are the most frequent properties, each accounting for 10.3% of all occurrences (10K). Geographic properties, including `city`, `state`, `streetAddress`, and `zip`, each occur approximately 5K times. Bloom contains only one relationship-property type, `lastUsed`, which occurs approximately 1K times.

S2 Details on benchmarks distribution

Figures S6–S8 show the distributions of node labels and relationship types occurring in the Cypher queries. For readability, only the ten most frequently represented individual types are shown. Since a query may involve multiple node labels and relationship types, the reported frequencies are not mutually exclusive.

S3 Representative queries

Tables S1– S3 provide examples of meaningful queries for the benchmarks across each of the five categories. For each category, the tables report the domain-specific motivation, the NL question, its manually curated Cypher translation, and the execution output.

S4 Example of prompt for the complete pipeline

Figure S9 shows an example of the prompt generated by the `S+RAG+O` pipeline for *fin2C*. The highlighted boxes distinguish the "Enhanced Schema" representation, the retrieved NL–Cypher examples, and their execution outputs. For readability, the figure displays only the top-ranked retrieved example; the experiments reported in the main manuscript use top-3 retrieval.

S5 Detailed performance for the RAG pipelines

This section provides a detailed analysis of the experimental results for the RAG pipelines reported in the main manuscript. We first compare the pipelines across syntactic and execution-based metrics and then examine how their performance varies across the five query categories.

S5.1 Detailed performance across metrics

Tables S4– S6 report the complete results for the RAG pipelines applied on the test sets of *bio2C*, *twt2C*, and *fin2C*. In addition to *Coverage* (`Cov`), which is discussed in the main article, the tables report Jaro–Winkler (`JW`), Jaccard (`Jac`), and `Pass`. These metrics provide complementary views of query quality. As a syntactic similarity measure, we used *Jaro–Winkler* (`JW`) to assess

Category	Biomedical motivation	NL question	Cypher query	Execution output
Node	Identifying diseases assigned to multiple biomedical classes supports the assessment of ontology-based disease classifications.	In which additional disease classes is the infectious disease labeled "bulbar polio" classified? Return its identifier and all its node labels.	`MATCH (i:Infectious_disease {Label:'bulbar polio'}) RETURN i.ID, LABELS(i)`	{'i.ID': 'MONDO:0005684', 'LABELS(i)': ['Neurodegenerative_disease', 'Disease', 'Infectious_disease']}
1-hop	Characterizing direct regulatory interactions between miRNAs and their associated evidence supports the reconstruction of post-transcriptional regulatory networks.	Which miRNAs are directly regulated by "hsa-miR-320d"? Return the regulator and target labels together with all properties of the regulatory relationship.	`MATCH (m1:miRNA {Label:'hsa-miR-320d'}) -[r:directly_regulates_activity_of] ->(m2:miRNA) RETURN m1.Label, m2.Label, PROPERTIES(r)`	{'m1.Label': 'hsa-miR-320d', 'm2.Label': 'microRNA hsa-mir-6501 precursor', 'PROPERTIES(r)': {'Source': ['TargetScan'], 'Weighted_CS_score': −0.02}}
2-hop	Linking alcohol-response processes to miRNAs implicated in brain ischemia supports the identification of molecular mediators connecting environmental response pathways with cerebrovascular disease.	Which miRNAs participate in the biological process "response to alcohol" and cause or contribute to "brain ischemia"?	`MATCH (bp:Biological_process {Label:'response to alcohol'}) -[:has_participant] ->(m:miRNA) -[:causes_or_contributes_to_condition] ->(d:Disease {Label:'brain ischemia'}) RETURN m.Label AS miRNA`	{'miRNA': 'hsa-miR-340-5p'}
3-hop	Tracing miRNA regulation of *ELOVL6* through cardiovascular disease associations to their phenotypes supports the investigation of regulatory mechanisms connected to heart-failure manifestations.	Which cardiovascular diseases are caused or contributed to by the gene "ELOVL6", whose activity is regulated by miRNAs, and which phenotypes are features of these diseases? Return the miRNA, gene, disease, and phenotype labels.	`MATCH (m:miRNA) -[:regulates_activity_of] ->(g:Gene {Label:'ELOVL6'}) -[:causes_or_contributes_to_condition] ->(card:Cardiovascular_disease) -[:disease_has_feature] ->(ph:Phenotype) RETURN m.Label AS miRNA, g.Label AS Gene, card.Label AS CardiovascularDisease, ph.Label AS Phenotype`	{'miRNA': 'hsa-miR-204-5p', 'Gene': 'ELOVL6', 'CardiovascularDisease': 'congestive heart failure', 'Phenotype': 'Dyspnea'}, {'miRNA': 'hsa-miR-211-5p', 'Gene': 'ELOVL6', 'CardiovascularDisease': 'congestive heart failure', 'Phenotype': 'Dyspnea'}
Advanced	Identifying genes involved in cardiovascular diseases and regulated by the largest number of miRNAs supports the prioritization of highly connected regulatory targets in pathologies.	Which genes that cause or contribute to cardiovascular diseases are regulated by the maximum number of distinct miRNAs, including ties? Return the gene label and the number of regulating miRNAs.	`MATCH (m:miRNA) -[:regulates_activity_of]->(g:Gene) -[:causes_or_contributes_to_condition] ->(:Cardiovascular_disease) WITH g, COUNT(DISTINCT m) AS Num_miRNAs WITH MAX(Num_miRNAs) AS maxNumMiRNAs MATCH (m:miRNA) -[:regulates_activity_of]->(g:Gene) -[:causes_or_contributes_to_condition] ->(:Cardiovascular_disease) WITH g, COUNT(DISTINCT m) AS Num_miRNAs, maxNumMiRNAs WHERE Num_miRNAs = maxNumMiRNAs RETURN g.Label AS Gene, Num_miRNAs AS NumberOfRegulatingMiRNAs`	{'Gene': 'VEGFA', 'NumberOfRegulatingMiRNAs': 55}

Supp. Table S1: Representative biomedical queries across the five *bio2C* categories.

Category	Social-media motivation	NL question	Cypher query	Execution output

Node	Estimating the minimum audience size of geographically localized accounts supports the characterization of the potential reach of specific user groups involved in online discussions.	Return the minimum followers count for people from Chicago in the year 2014.	`MATCH (u:User)`
			`WHERE u.location CONTAINS 'Chicago'`
			`AND u.created_at CONTAINS '2014'`
			`RETURN MIN(u.followers_count)`
			`AS minFollowersChicago2014`
				{'minFollowersChicago2014': 19525}

1-hop	Quantifying the use of hashtags referring to Hillary Clinton supports the measurement of campaign-related narrative amplification across tweets.	How many `HAS_TAG` relationships exist for the hashtag "hillary"? Return the number of relationships.	`MATCH (:Tweet)-[r:HAS_TAG]`
			`->(h:Hashtag)`
			`WHERE h.tag CONTAINS 'hillary'`
			`RETURN COUNT(r) AS relCount`
				{'relCount': 5259}

2-hop	Tracing mentions between malicious accounts supports their interactions.	Find tweets posted by the troll with screen name "MarissaImStrong" that mention a troll. Return the posting troll's screen name, the mentioned troll's screen name, and the tweet text.	`MATCH (tr1:Troll`
			`{screen_name:'MarissaImStrong'})`
			`-[:POSTED]->(t:Tweet)`
			`-[:MENTIONS]->(tr2:Troll)`
			`RETURN tr1.screen_name,`
			`tr2.screen_name AS mentioned_troll,`
			`t.text`
				{'tr1.screen_name': 'MarissaImStrong',
				'mentioned_troll': 'JeffreyKahunas',
				't.text': '@JeffreyKahunas Only a blind and dumb fool would vote for this maggot lying Clinton.'}

3-hop	Reconstructing reply chains involving malicious accounts based in St. Petersburg that posted Russia-related hashtags supports geopolitical analysis.	Identify troll accounts whose time zone is "St. Petersburg" that posted a reply to a tweet having the hashtag "russian". Return the troll screen name and the creation dates of both tweets.	`MATCH (tr:Troll)-[:POSTED]->(t1:Tweet)`
			`-[:IN_REPLY_TO]->(t2:Tweet)`
			`-[:HAS_TAG]->(h:Hashtag)`
			`WHERE tr.time_zone = 'St. Petersburg'`
			`AND h.tag = 'russian'`
			`RETURN tr.screen_name,`
			`t1.created_str,`
			`t2.created_str`
				{'tr.screen_name': 'WarfareWW',
				't1.created_str': '2016-09-29 17:11:39',
				't2.created_str': '2016-09-29 15:30:21'}

Advanced	Aggregating links to Washington Post articles propagated through retweets supports the identification of external news narratives amplified within the campaign and the analysis of source-specific information dissemination.	Find tweets that retweet tweets having URLs whose `expanded_url` contains "washingtonpost" and where the retweeted tweet has retweet number greater than 0. For each such retweeting tweet, return its id, the number of distinct matching URLs, and the list of distinct matching `expanded_url` values.	`MATCH (t1:Tweet)-[:RETWEETED]->(t2:Tweet)`
			`-[:HAS_LINK]->(u:URL)`
			`WHERE u.expanded_url CONTAINS 'washingtonpost'`
			`AND t2.retweet_count > 0`
			`WITH t1,`
			`COUNT(DISTINCT u) AS url_count,`
			`COLLECT(DISTINCT u.expanded_url) AS urls`
			`RETURN t1.id, url_count, urls`
				{'t1.id': '766433870010994688',
				'url_count': 1,
				'urls': [https://www.washingtonpost.com/...]}

Supp. Table S2: Representative social-media queries across the five *twt2C* categories.

Category	Financial motivation	NL question	Cypher query	Execution output

Node	Determining the maximum credit limit supports the assessment of the largest potential exposure associated with a credit card.	What is the maximum limit of all the credit cards?	`MATCH (c:CreditCard)`
			`RETURN MAX(c.limit)`
			`AS MaximumCreditCardLimit`
				{'MaximumCreditCardLimit': 75000.0}

1-hop	Identifying the financial instruments associated with a flagged individual supports customer investigation.	What are the credit card account numbers owned by the `Flagged` individual with `UniqueId` "!6GO30ND9309"?	`MATCH (f:Flagged`
			`{UniqueId:'!6GO30ND9309'})`
			`-[:HAS_CREDITCARD]`
			`->(c:CreditCard)`
			`RETURN c.accountNumber`
				{'c.accountNumber':
				'3697 9150 5720 8632'}

2-hop	Resolving bank accounts from a Social Security Number supports identity-based account tracing.	Find the bank account number and balance for the account holder with the SSN "002-51-3952".	`MATCH (ah:AccountHolder)`
			`-[:HAS_SSN]`
			`->(s:SSN`
			`{ssn:'002-51-3952'}),`
			`(ah)-[:HAS_BANKACCOUNT]`
			`->(ba:BankAccount)`
			`RETURN ba.accountNumber,`
			`ba.balance`
				{'ba.accountNumber':
				'0200 2288 7598 9221',
				'ba.balance': 6760.32}

3-hop	Tracing high-value transfers conducted through a specific telephone provider and sent to low-balance accounts supports the prioritization of potentially anomalous transaction patterns.	For account holders with phone numbers from "Yellow Mobile", find money transfers above 900 sent using those numbers to bank accounts with balances below 1000. Return the account holder, phone number, transfer amount, receiving account, and the last-used timestamp.	`MATCH (ah:AccountHolder)`
			`-[:HAS_PHONENUMBER]`
			`->(pn:PhoneNumber`
			`{provider:'Yellow Mobile'})`
			`<-[r:WITH]-(mt:MoneyTransfer)`
			`-[:SEND]->(ba:BankAccount)`
			`WHERE mt.amount > 900.0`
			`AND ba.balance < 1000.0`
			`RETURN ah.fullName AS accountHolderName,`
			`pn.phone AS phoneNumber,`
			`mt.amount AS transferAmount,`
			`ba.accountNumber`
			`AS receiverBankAccountNumber,`
			`r.lastUsed AS withLastUsed`
				{'accountHolderName': 'Lorene Bacher',
				'phoneNumber': '226.4679.3971.2',
				'transferAmount': 918.59,
				'receiverBankAccountNumber':
				'0027 4971 1914 1895',
				'withLastUsed': 1518026120955}

Advanced	Identifying the pair of accounts with the largest aggregate transfer volume supports the prioritization of high-intensity financial flows.	Find the sender and receiver bank account pair with the highest total amount transferred between them. Return both account numbers and the maximum total amount.	`MATCH (sender:BankAccount)-[:SEND]`
			`->(mt:MoneyTransfer)-[:SEND]`
			`->(receiver:BankAccount)`
			`WITH sender.accountNumber AS SenderAcc,`
			`receiver.accountNumber AS ReceiverAcc,`
			`SUM(mt.amount) AS TotalTransferred`
			`WITH MAX(TotalTransferred) AS MaxTotal`
			`MATCH (sender2:BankAccount)-[:SEND]`
			`->(mt2:MoneyTransfer)-[:SEND]`
			`->(receiver2:BankAccount)`
			`WITH sender2.accountNumber AS SenderAcc,`
			`receiver2.accountNumber AS ReceiverAcc,`
			`SUM(mt2.amount) AS TotalTransferred,`
			`MaxTotal`
			`WHERE TotalTransferred = MaxTotal`
			`RETURN SenderAcc, ReceiverAcc,`
			`TotalTransferred`
				{'SenderAcc': '9801 5030 4566 9374',
				'ReceiverAcc': '0094 3982 1035 1935',
				'TotalTransferred': 1836.03}

Supp. Table S3: Representative financial queries across the five *fin2C* categories.

the character-level similarity between a generated query Q_C and the corresponding ground-truth query Q_T. The *Jaccard* metric (`Jac`) evaluates semantic similarity as the intersection over union of the execution outputs produced by Q_C and Q_T. Property names and aliases are ignored during result comparison, whereas ordering is considered only when the ground-truth query explicitly contains an `ORDER BY` clause. Finally, *Pass* is defined as the indicator function $\mathbb{I}[\texttt{Cov} = 1]$ and identifies queries achieving complete semantic coverage. Bold values indicate the best-performing pipeline for each model, whereas red values denote the highest score obtained for the corresponding benchmark and metric.

The results confirm that syntactic similarity alone is not sufficient to assess Text2Cypher performance. For example, on *bio2C*, `L8B` under vanilla prompting obtains a `JW` score of 68.7% but a `Cov` and `Pass` of 0%. Similarly, on *fin2C*, vanilla `GPT4o` reaches 80.7% in `JW`, while obtaining only 5.8% in `Cov`. Generated queries may therefore resemble the gold queries lexically while still using incorrect schema elements, traversal directions, constraints, or aggregation operations.

On *bio2C*, the strongest results are obtained by $\texttt{GPT4o}^{\texttt{FT}}$ combined with retrieval. `RAG` achieves a `Cov` of 89.3%, tied with `S+RAG+O` for the highest coverage, and a `Pass` of 86.0%. The base `GPT4o` model benefits from richer contextual configurations, reaching its highest `JW`, `Cov`, and `Pass` scores with `RAG+O` or `S+RAG+O`. These results indicate that fine-tuned models can infer much of the relevant schema information from retrieved examples, whereas base models benefit more from the explicit combination of examples, schema descriptions, and execution outputs.

Performance is generally higher on *twt2C*. The best configuration, $\texttt{GPT4o}^{\texttt{FT}}$ with `RAG+O`, obtains 96.7% in `JW`, 92.8% in `Jac`, 94.3% in `Cov`, and 92.8% in `Pass`. The contribution of the retrieved execution outputs is particularly visible for this benchmark, although the gains over `RAG` remain limited. For the base `GPT4o` and `GPT4om` models, combining the schema with retrieved examples produces the strongest or nearly strongest results. This behaviour is consistent with the compact RTT schema, for which an explicit schema representation can be incorporated without introducing excessive prompt complexity.

On *fin2C*, the proposed pipelines again achieve high scores across all execution-based metrics. Among them, $\texttt{GPT4o}^{\texttt{FT}}$ with `S+RAG` obtains 93.5% in `Jac`, 95.5% in `Cov`, and 94.0% in `Pass`. NeoDash achieves the highest individual scores for some model–metric combinations, including a `Cov` of 95.9% and a `Pass` of 95.3% with `GPT4o`. Nevertheless, the results of `S+RAG` remain close and are more consistent across the base and fine-tuned GPT models. The high performance obtained on *fin2C*, despite its smaller number of manually curated pairs, further shows that benchmark utility depends on the quality, consistency, and representativeness of the examples rather than on collection size alone.

The open-source `L8B` model displays a different pattern. Its base version consistently benefits from `S+RAG`, which raises `Cov` from 0–2% under vanilla prompting to 26.0%, 66.6%, and 52.8% on *bio2C*, *twt2C*, and *fin2C*, respectively. In contrast, the complete `S+RAG+O` configuration causes a substantial reduction across the execution-based metrics. The fine-tuned `L8B` model also fails to benefit consistently from the richer prompts. This behaviour suggests that smaller models are more sensitive to the amount and composition of the contextual information included in the prompt.

Overall, the four metrics provide complementary evidence on pipeline quality. Retrieval produces the most consistent gains across benchmarks, while schema information is especially beneficial for base models. Execution outputs can provide further improvements, but their effect depends on the target graph and model. The differences among *bio2C*, *twt2C*, and *fin2C* also confirm that the benchmarks exercise distinct aspects of Text2Cypher generation and enable the evaluation of alternative combinations of fine-tuning, retrieval, schema grounding, and execution-aware prompting.

	L8B	L8B^{FT}	GPT4om	GPT4om^{FT}	GPT4o	GPT4o^{FT}
Vanilla	68.7	**81.7**	78.3	94.0	78.7	93.6
Schema	74.7	69.9	87.5	92.6	87.9	92.4
RAG	74.7	69.9	93.2	**94.5**	93.9	**94.7**
RAG+O	74.7	69.9	93.3	94.4	**94.1**	**94.7**
S+RAG	**83.2**	76.6	93.2	93.9	93.8	**94.7**
S+RAG+O	73.1	71.4	**93.5**	94.1	93.9	94.4
NeoD	*NA*	*NA*	88.1	88.2	88.1	88.0

(a) Average `JW` scores.

	L8B	L8B^{FT}	GPT4om	GPT4om^{FT}	GPT4o	GPT4o^{FT}
Vanilla	0.0	**29.7**	0.0	69.2	0.1	76.1
Schema	2.1	8.7	27.3	63.8	57.8	74.1
RAG	2.1	8.7	63.0	**78.4**	72.6	**84.3**
RAG+O	2.1	8.7	65.1	76.5	71.7	83.6
S+RAG	**22.5**	0.7	68.3	74.5	76.5	83.4
S+RAG+O	0.8	1.1	**68.7**	76.5	**76.9**	84.1
NeoD	*NA*	*NA*	52.2	52.5	52.9	52.7

(b) Average `Jac` scores.

	L8B	L8B^{FT}	GPT4om	GPT4om^{FT}	GPT4o	GPT4o^{FT}
Vanilla	0.0	**40.3**	0.4	74.0	0.2	80.0
Schema	3.6	12.4	44.2	71.9	67.7	79.2
RAG	3.6	12.4	68.5	**83.1**	76.2	**89.3**
RAG+O	3.6	12.4	69.3	81.6	76.8	88.3
S+RAG	**26.0**	1.8	74.9	81.6	82.6	89.0
S+RAG+O	1.3	2.1	**75.2**	82.6	**83.1**	**89.3**
NeoD	*NA*	*NA*	59.5	60.2	60.3	60.0

(c) Average `Cov` scores.

	L8B	L8B^{FT}	GPT4om	GPT4om^{FT}	GPT4o	GPT4o^{FT}
Vanilla	0.0	**36.0**	0.4	68.8	0.0	75.6
Schema	3.2	11.2	38.4	66.0	57.6	73.2
RAG	3.2	11.2	62.0	**78.8**	70.8	**86.0**
RAG+O	3.2	11.2	63.2	77.6	72.0	85.2
S+RAG	**22.0**	1.2	**66.4**	74.8	**77.2**	85.2
S+RAG+O	0.8	1.2	**66.4**	76.4	**77.2**	85.6
NeoD	*NA*	*NA*	50.4	52.0	51.2	51.6

(d) Average `Pass` scores.

Supp. Table S4: Pipelines' performance for *bio2C* according to evaluation metrics.

S5.2 Detailed performance across categories

Table S7 reports the performance across the five query complexity levels for the proposed configuration achieving the highest average `Cov` on each benchmark: `GPT4o`^{FT} with `RAG` for *bio2C*, with `RAG+O` for *twt2C*, and with `S+RAG` for *fin2C*.

The results do not show a monotonic decrease in performance as the number of traversed relationships increases. On *bio2C*, all metrics decrease from Node to 1-hop, recover for 2-hop queries, and reach their highest or nearly highest values for 3-hop queries. Performance then decreases for the Advanced category. The strong results for Node queries reflect their comparatively simple structure, whereas 1-hop queries require the model to identify the correct relationship type, direction, and constraints from limited structural context. In contrast, multi-hop queries provide richer and more distinctive traversal patterns, which can facilitate the retrieval of structurally aligned examples. Advanced queries remain more challenging because they frequently involve aggregation, ranking, intermediate `WITH` clauses, and multi-stage query composition. A more detailed analysis of this behaviour and of the impact of category-conform retrieved examples is provided in [36].

	L8B	L8B^{FT}	GPT4om	GPT4om^{FT}	GPT4o	GPT4o^{FT}
Vanilla	73.1	76.7	82.3	**96.2**	83.6	95.6
Schema	79.4	**87.4**	86.1	94.6	86.5	94.5
RAG	79.4	**87.4**	**95.1**	96.1	95.8	**96.7**
RAG+O	79.4	**87.4**	95.0	96.0	**95.9**	**96.7**
S+RAG	**92.7**	85.4	94.8	95.9	95.4	96.5
S+RAG+O	76.5	76.5	94.9	96.0	95.5	96.4
NeoDash	*NA*	*NA*	93.4	93.7	93.2	93.8

(a) Average `JW` scores.

	L8B	L8B^{FT}	GPT4om	GPT4om^{FT}	GPT4o	GPT4o^{FT}
Vanilla	1.9	11.3	6.4	86.1	15.2	87.7
Schema	9.7	**49.1**	47.6	80.7	66.6	85.4
RAG	9.7	**49.1**	77.6	86.5	85.6	91.2
RAG+O	9.7	**49.1**	77.4	**86.7**	86.1	**92.8**
S+RAG	**61.0**	40.6	**77.9**	84.4	**88.2**	92.0
S+RAG+O	0.8	1.0	77.7	86.4	88.1	90.6
NeoDash	*NA*	*NA*	73.3	81.7	71.8	80.5

(b) Average `Jac` scores.

	L8B	L8B^{FT}	GPT4om	GPT4om^{FT}	GPT4o	GPT4o^{FT}
Vanilla	2.0	13.1	9.7	89.5	18.5	90.0
Schema	13.4	**53.6**	57.1	85.7	73.7	88.9
RAG	13.4	**53.6**	81.1	89.1	87.5	93.1
RAG+O	13.4	**53.6**	81.4	**90.0**	87.7	**94.3**
S+RAG	**66.6**	43.0	**83.2**	88.2	**91.1**	93.7
S+RAG+O	1.2	1.6	82.2	88.7	90.9	92.6
NeoDash	*NA*	*NA*	77.4	87.8	77.9	87.8

(c) Average `Cov` scores.

	L8B	L8B^{FT}	GPT4om	GPT4om^{FT}	GPT4o	GPT4o^{FT}
Vanilla	1.6	12.4	7.6	**86.8**	15.2	86.8
Schema	10.0	**46.4**	50.0	80.8	68.8	86.0
RAG	10.0	**46.4**	76.8	86.0	84.8	91.6
RAG+O	10.0	**46.4**	77.2	**86.8**	85.2	**92.8**
S+RAG	**62.4**	41.6	**78.0**	84.0	88.4	92.0
S+RAG+O	1.2	1.2	76.8	85.2	**88.8**	90.8
NeoDash	*NA*	*NA*	73.6	82.4	73.6	82.4

(d) Average `Pass` scores.

Supp. Table S5: Pipelines' performance for *twt2C* according to evaluation metrics.

Performance on *twt2C* is consistently high across all five levels. The differences among Node, 1-hop, and 2-hop queries are small, while 3-hop queries achieve the highest score for every metric, including a `Cov` and `Pass` of 96.0%. This suggests that deeper traversals are not inherently more difficult when they follow distinctive interaction patterns and are supported by representative retrieved examples. The comparatively compact, tweet-centered RTT schema may also reduce ambiguity in selecting node labels and relationship types. Advanced queries show a moderate decrease, particularly in `JW` and `Jac`, reflecting the additional difficulty introduced by aggregation and precise output requirements.

A similar pattern is observed on *fin2C*. Three-hop queries obtain the highest semantic scores, reaching 96.4% in `Jac`, 99.1% in `Cov`, and 96.7% in `Pass`. Financial multi-hop questions often follow explicit domain-specific paths connecting individuals, identifiers, transactions, and accounts, which provide strong structural cues for both retrieval and generation. Advanced queries obtain lower `JW` and `Jac` scores but retain a `Cov` and `Pass` of 93.3%. This indicates that some generated queries retrieve all the required information despite differing from the ground-truth query at the lexical level or returning additional results.

	L8B	L8B^{FT}	GPT4om	GPT4om^{FT}	GPT4o	GPT4o^{FT}
Vanilla	68.5	68.4	79.5	89.1	80.7	88.7
Schema	72.4	**80.6**	86.2	90.8	86.7	90.8
RAG	72.4	**80.6**	91.8	93.7	92.7	94.0
RAG+O	72.4	**80.6**	**91.9**	93.2	92.8	93.9
S+RAG	**87.6**	78.2	91.9	93.2	**92.9**	93.9
S+RAG+O	78.5	76.2	91.8	92.9	92.6	93.5
NeoDash	*NA*	*NA*	83.9	**97.5**	90.4	**95.3**

(a) Average `JW` scores.

	L8B	L8B^{FT}	GPT4om	GPT4om^{FT}	GPT4o	GPT4o^{FT}
Vanilla	0.0	0.0	5.3	44.5	5.6	41.9
Schema	10.1	**17.1**	50.6	71.8	76.7	86.6
RAG	10.1	**17.1**	73.7	86.0	78.8	87.4
RAG+O	10.1	**17.1**	76.4	85.0	81.1	85.9
S+RAG	**49.9**	10.7	79.1	88.1	90.5	**93.5**
S+RAG+O	0.7	1.1	80.1	86.2	89.8	92.9
NeoDash	*NA*	*NA*	**82.0**	**95.0**	**93.2**	90.1

(b) Average `Jac` scores.

	L8B	L8B^{FT}	GPT4om	GPT4om^{FT}	GPT4o	GPT4o^{FT}
Vanilla	0.0	0.0	5.3	45.4	5.8	43.0
Schema	12.6	**22.7**	52.9	76.1	77.9	88.8
RAG	12.6	**22.7**	75.7	87.4	80.6	89.0
RAG+O	12.6	**22.7**	78.4	86.7	82.9	87.0
S+RAG	**52.8**	12.0	80.9	89.3	91.7	**95.5**
S+RAG+O	1.3	1.2	**82.6**	87.5	90.7	94.1
NeoDash	*NA*	*NA*	82.3	**95.1**	**95.9**	90.9

(c) Average `Cov` scores.

	L8B	L8B^{FT}	GPT4om	GPT4om^{FT}	GPT4o	GPT4o^{FT}
Vanilla	0.0	0.0	5.3	43.3	5.3	40.7
Schema	9.3	**18.7**	49.3	70.0	74.7	86.0
RAG	9.3	**18.7**	70.7	84.7	75.3	86.0
RAG+O	9.3	**18.7**	73.3	82.7	78.0	85.3
S+RAG	**49.3**	10.7	76.7	86.7	90.0	**94.0**
S+RAG+O	1.3	0.7	78.7	85.3	88.7	92.7
NeoDash	*NA*	*NA*	**82.0**	**94.7**	**95.3**	90.0

(d) Average `Pass` scores.

Supp. Table S6: Pipelines' performance for *fin2C* according to evaluation metrics.

Overall, these findings show that query difficulty cannot be estimated from hop count alone. Performance also depends on the ambiguity of the schema path, the distinctiveness of the traversal pattern, the Cypher constructs required by the question, and the structural alignment between the target query and the retrieved examples. The three benchmarks therefore provide complementary settings for evaluating Text2Cypher systems across both progressively deeper traversals and advanced query-composition requirements.

S5.3 Error analysis

We analyze failure patterns for the best-performing pipeline on each benchmark and for `NeoDash`. We distinguish *Compile-time Errors (CEs)*, in which the generated query cannot be successfully compiled or executed, from *Logic Errors (LEs)*, in which the query executes but its result does not fully match the semantics of the ground-truth answer. Logic errors include hallucinated schema or value elements, violations of graph-schema constraints, incomplete or incorrect `RETURN` clauses, errors in logical query composition, and cases involving residual ambiguity in the NL formulation. The latter arise when the NL question admits multiple plausible interpretations,

Category	JW	Jac	Cov	Pass
Node	95.6	87.4	**94.3**	**92.0**
1-hop	94.2	80.1	84.5	78.0
2-hop	95.6	89.2	91.4	90.0
3-hop	**96.9**	**91.1**	92.0	**92.0**
Advanced	91.3	73.6	84.4	78.0

(a) Performance of GPT4o^{FT} with the RAG pipeline on *bio2C*.

Category	JW	Jac	Cov	Pass
Node	96.7	94.2	94.4	92.0
1-hop	96.8	92.2	94.4	94.0
2-hop	97.4	91.6	94.0	92.0
3-hop	**99.3**	**96.0**	**96.0**	**96.0**
Advanced	93.4	90.0	92.5	90.0

(b) Performance of GPT4o^{FT} with the RAG+O pipeline on *twt2C*.

Category	JW	Jac	Cov	Pass
Node	94.4	94.5	96.1	93.3
1-hop	95.6	93.3	93.3	93.3
2-hop	**95.8**	94.4	95.6	93.3
3-hop	94.3	**96.4**	**99.1**	**96.7**
Advanced	89.6	89.0	93.3	93.3

(c) Performance of GPT4o^{FT} with the S+RAG pipeline on *fin2C*.

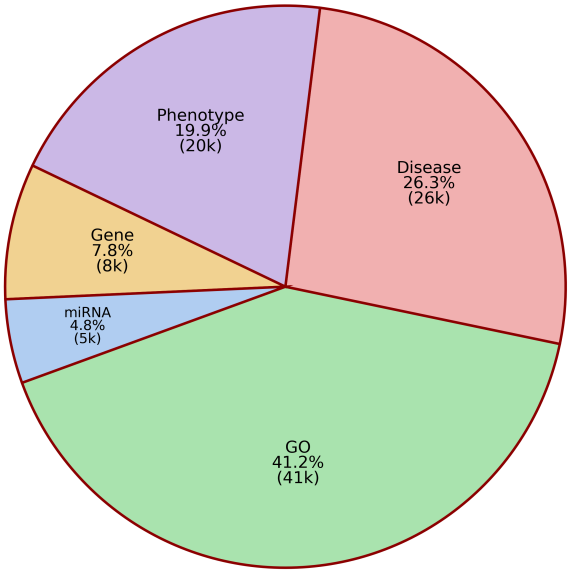
Supp. Table S7: Performance across the five query complexity levels.

and the query generated by the LLM corresponds to a valid interpretation that differs from the ground truth formulation. Finally, we analyzed the distribution of LE across query categories.

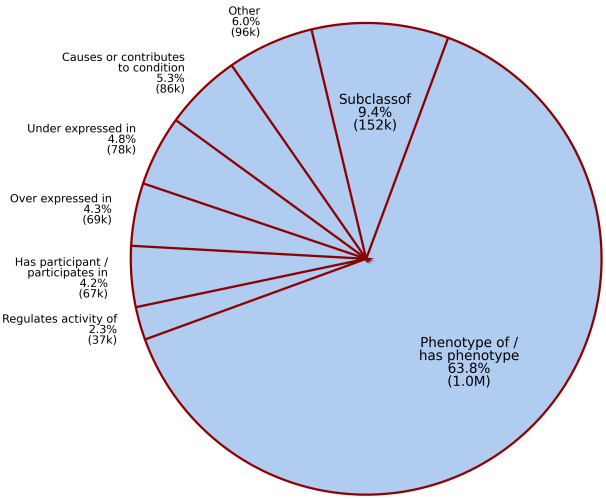
bio2C GPT4o^{FT} with RAG eliminates CEs and reduces LEs to 35 cases, i.e., 85.2% and 75.9% fewer LEs with respect to vanilla and NeoDash. Most LEs occur in the 1-hop and advanced categories (11 each), while the node and 3-hop categories exhibit the fewest LEs (4 each).

twt2C GPT4o^{FT} with RAG+O eliminates CEs and reduces LEs to 18 cases, i.e., 91.5% and 71.9% fewer LEs with respect to vanilla and NeoDash. Most LEs occur in the advanced category (5), while the 3-hop category exhibits the fewest LEs (2).

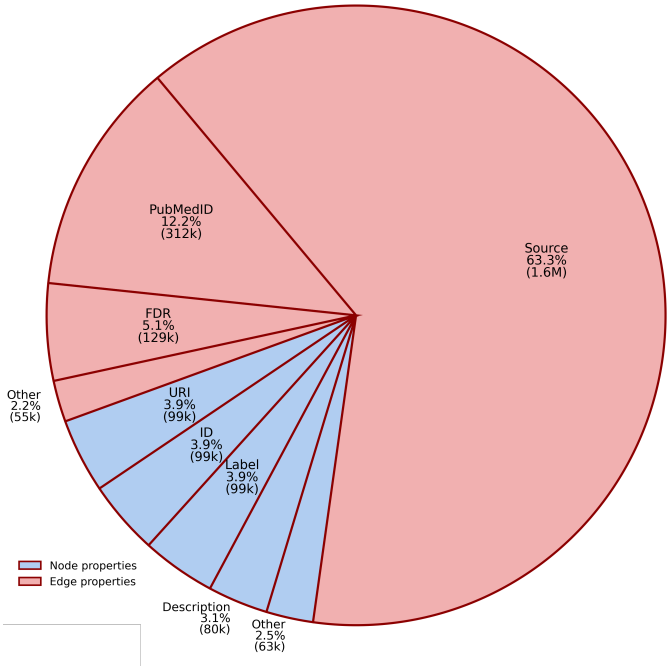
fin2C GPT4o^{FT} with S+RAG eliminates CEs and reduces LEs to 9 cases, i.e., 85.9% fewer LEs with respect to vanilla. The node, 1-hop, 2-hop, and advanced categories account for 2 LEs each, while the 3-hop category exhibits the fewest LEs (1).



(a) Distribution of node labels.

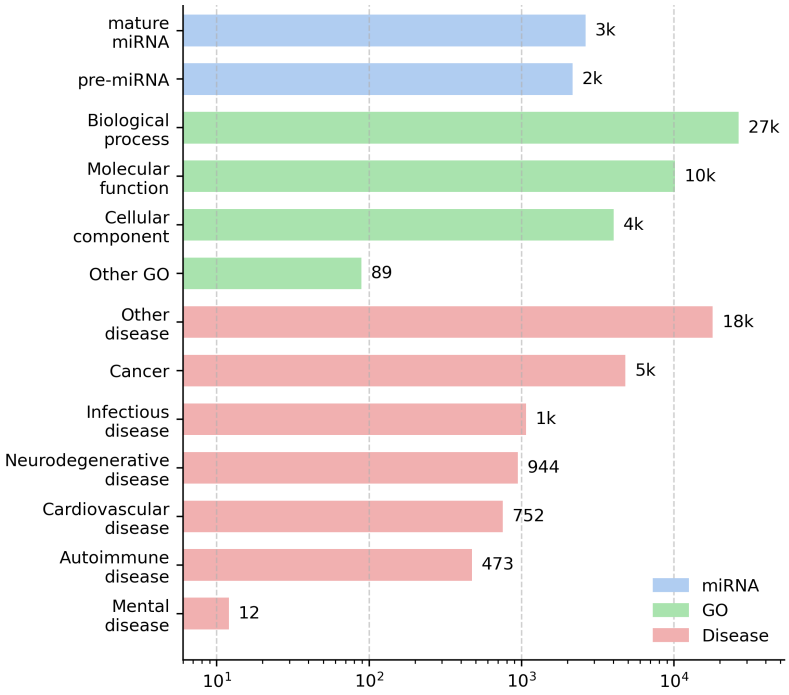


(b) Distribution of relationship types.

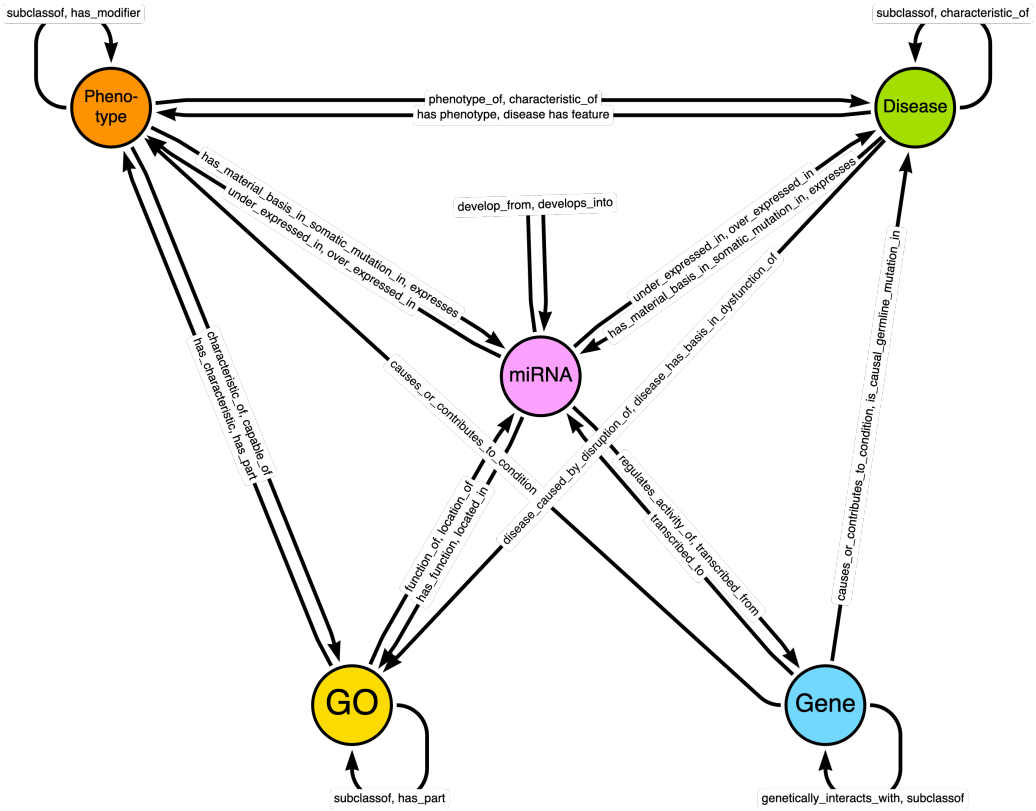


(c) Distribution of properties.

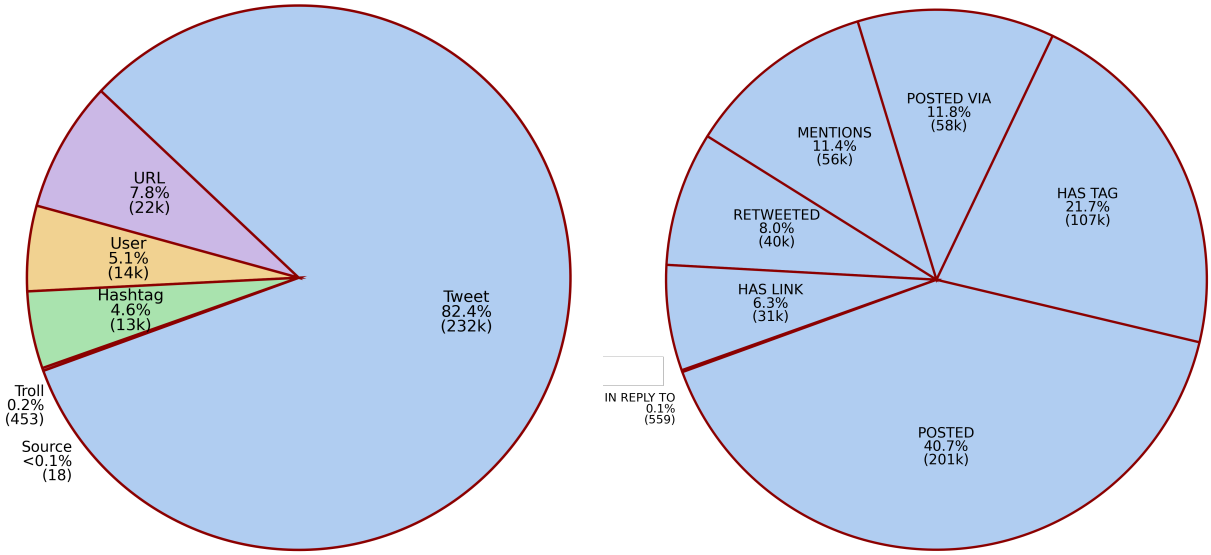
Supp. Fig. S1: Statistics for miRNA-KG.



Supp. Fig. S2: Distribution of node subtypes in miRNA-KG.

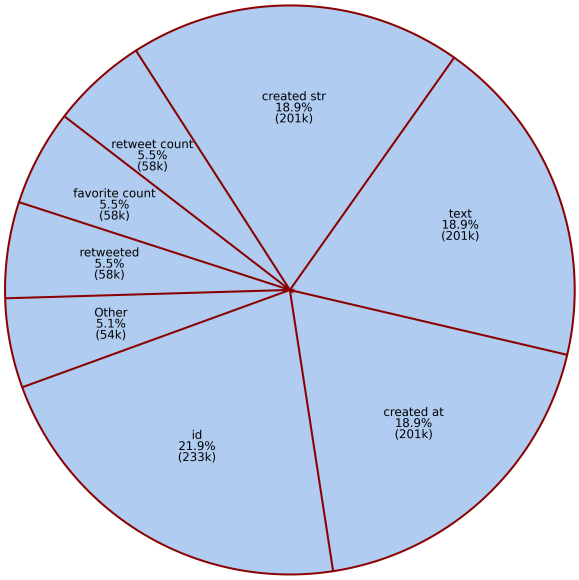


Supp. Fig. S3: Schema of miRNA-KG.



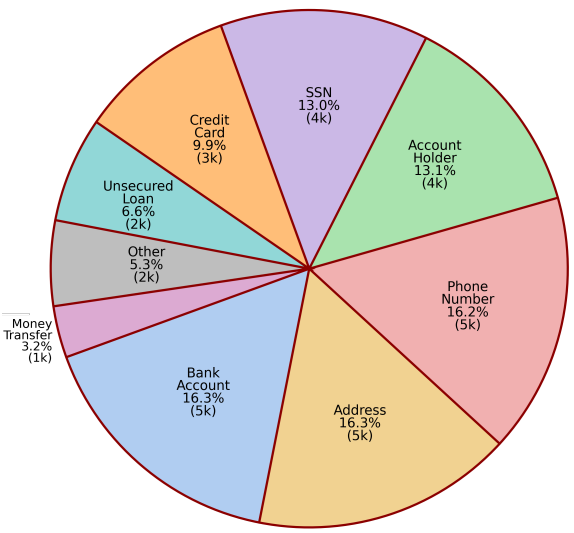
(a) Distribution of node labels.

(b) Distribution of relationship types.

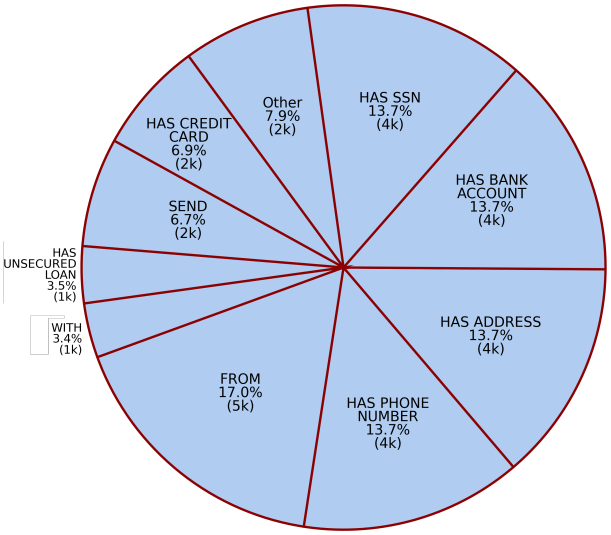


(c) Distribution of properties.

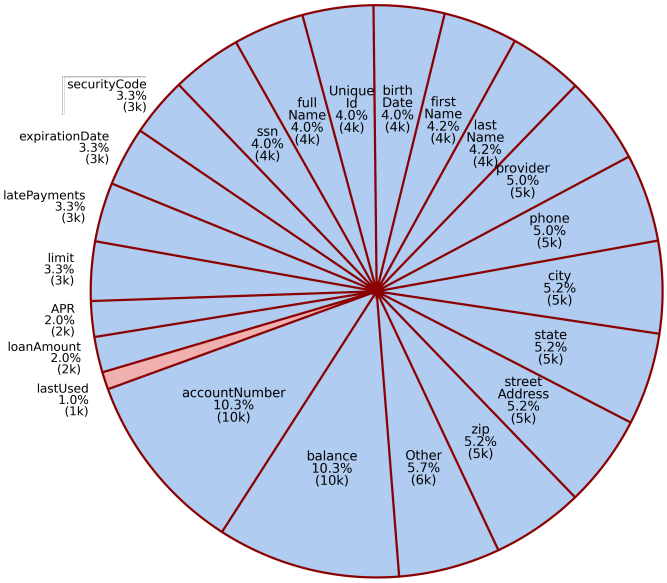
Supp. Fig. S4: Statistics for Russian Twitter Trolls.



(a) Distribution of node labels.

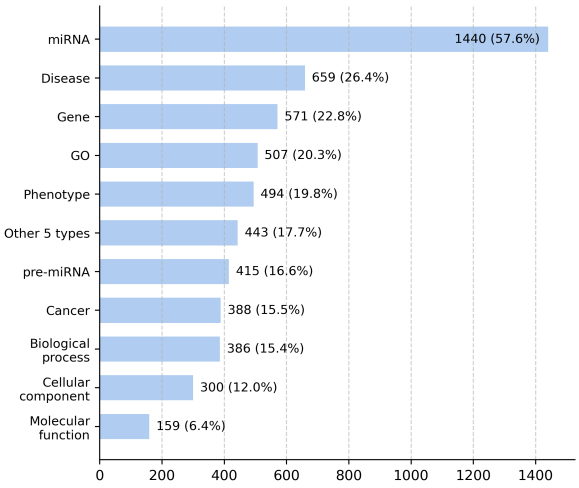


(b) Distribution of relationship types.

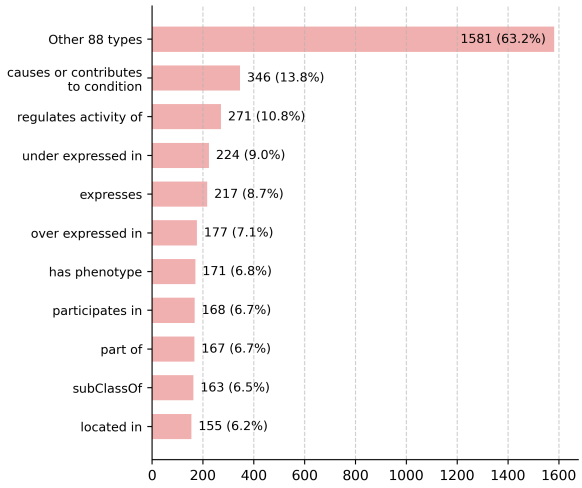


(c) Distribution of node and relationship properties.

Supp. Fig. S5: Statistics for the Bloom graph.

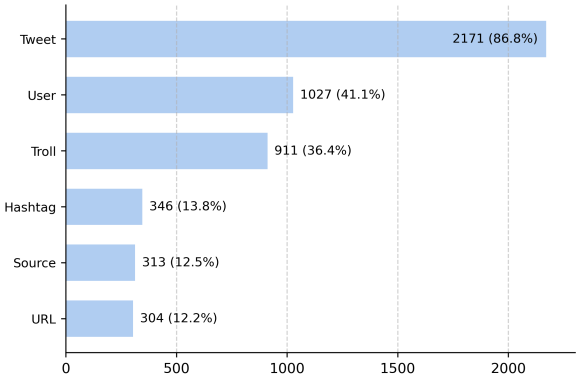


(a) Node labels distribution in *bio2C*.

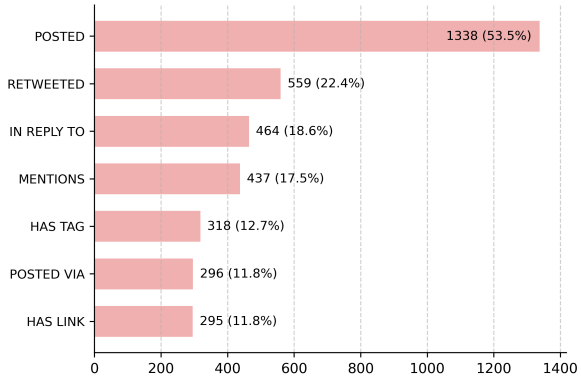


(b) Relationship types distribution in *bio2C*.

Supp. Fig. S6: Node labels and relationship types distribution in *bio2C*.

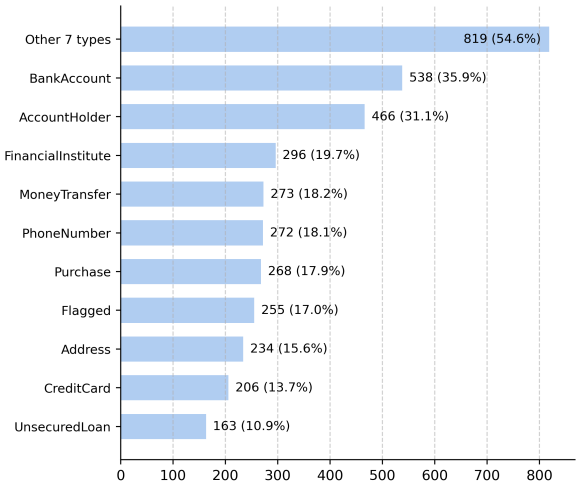


(a) Node label distribution in *twt2C*.

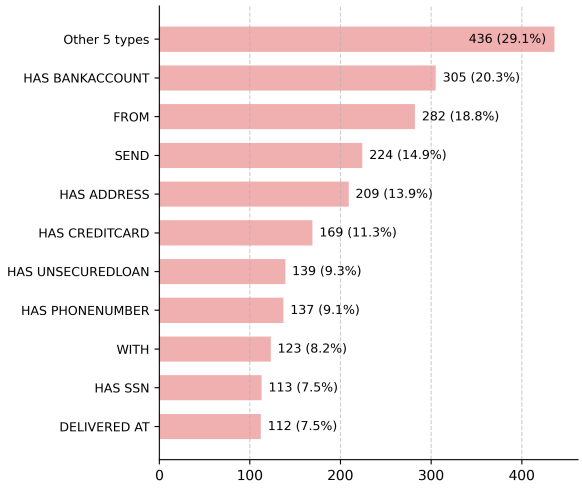


(b) Relationship type distribution in *twt2C*.

Supp. Fig. S7: Node label and relationship type distributions in *twt2C*.



(a) Node label distribution in *fin2C*.



(b) Relationship type distribution in *fin2C*.

Supp. Fig. S8: Node label and relationship type distributions in *fin2C*.

Instructions

```
Given an input question, convert it to a Cypher query.  No pre-amble.
Based on the Neo4j graph schema below, write a Cypher query that would answer the user's question.
Return only Cypher statement, no backticks, nothing else.
```

Schema

```
Node properties:
AccountHolder
  - lastName:  STRING Example:  'Kasper'
  - birthDate:  STRING Example:  '1980-08-03'
  - UniqueId:  STRING Example:  'G0WM54KI08W'
  - firstName:  STRING Example:  'Celina'
  - fullName:  STRING Example:  'Celina Kasper'
  ...

BankAccount
  - accountNumber:  STRING Example:  '3892 0649 4837 0224'
  - balance:  FLOAT Min:  100.24, Max:  9998.62
  - lastReceived:  INTEGER Min:  1518026143723, Max:  1518026143723
  - lastSend:  INTEGER Min:  1518026088096, Max:  1518026143723

PhoneNumber
  - phone:  STRING Example:  '982.5359.0729.7'
  - provider:  STRING Example:  'KPA Tel'
  - touched:  INTEGER Min:  1518026088097, Max:  1518026143723

MoneyTransfer
  - transactionDate:  STRING Example:  '2018-02-07'
  - amount:  FLOAT Min:  10.44, Max:  998.43

...

Relationships:
(:AccountHolder)-[:HAS_PHONENUMBER]->(:PhoneNumber)
(:BankAccount)-[:SEND]->(:MoneyTransfer)
(:MoneyTransfer)-[:SEND]->(:BankAccount)
(:MoneyTransfer)-[:WITH]->(:PhoneNumber)
...

Relationship properties:
WITH
  - lastUsed:  INTEGER Min:  1518026081178, Max:  1518026143737
```

Retrieved example

```
The following example provides useful patterns for querying the graph database (Neo4j outputs are
also reported).

[Query natural language]
Which money transfers above 900 are made using a phone number from the provider ''Yellow Mobile'' and
sent to a bank account, along with their amounts and the last-used date of the phone number?

[Query Cypher]
MATCH (m:MoneyTransfer)-[:SEND]->(b:BankAccount),
      (m)-[r:WITH]->(p:PhoneNumber {provider:'Yellow Mobile'})
WHERE m.amount > 900
RETURN m.amount, r.lastUsed
```

Execution output

```
[Output Neo4j]
{"m.amount":  907.1, "r.lastUsed":  1518026088127}
{"m.amount":  953.26, "r.lastUsed":  1518026091495}
...
```

NL question

Question: For account holders who own phone numbers from the provider 'Yellow Mobile', which money transfers above 900.0 occur with those phone numbers and are sent to bank accounts with balances below 1000.0? Return the account holder, phone number, transfer amount, receiver bank account number, and when the phone number was last used in the transfer.
Cypher query:

Supp. Fig. S9: Prompt for the S+RAG+O pipeline.