Preprint

What Quantum Encoding to Use for your Knowledge Graph?

Christos Polimatidis
Computer Science Department,
University of Crete
Heraklion, Crete, Greece
polimatidischris@gmail.com

Haridimos Kondylakis
Computer Science Department,
University of Crete
Heraklion, Greece
Institute of Computer Science, &
Center for Quantum Science &
Technologies (FORTH-QuTech),
Foundation for Research and
Technology - Hellas
Heraklion, Greece
kondylak@ics.forth.gr

Yannis Tzitzikas
Computer Science Department,
University of Crete
Heraklion, Greece
Institute of Computer Science, &
Center for Quantum Science &
Technologies (FORTH-QuTech),
Foundation for Research and
Technology - Hellas
Heraklion, Greece
tzitzik@ics.forth.gr

Abstract

Knowledge Graphs (KGs) are widely used for integration, search, and reasoning, but quantum processors cannot operate directly on symbolic RDF data. Therefore, processing KG workloads on quantum hardware depends critically on how KG information is encoded into quantum states. This paper compares three encoding families for KGs: basis, amplitude, and phase encodings, analyzing their qubit requirements, state-preparation costs, and measurement/readout implications, and identifying the KG tasks for which each encoding is most suitable. We illustrate the differences using a common running example and validate the main observations through a Qiskit-based prototype executed on local quantum simulators over the running example, synthetic RDF graphs with up to 1000 triples, and real RDF samples with up to 500 triples.

VLDB Workshop Reference Format:
Christos Polimatidis, Haridimos Kondylakis, and Yannis Tzitzikas. What Quantum Encoding to Use for your Knowledge Graph?. VLDB 2026 Workshop: QC&DKM 2026 - 2nd Workshop on Quantum Computing and Data/Knowledge Management.

VLDB Workshop Artifact Availability:
The source code, input RDF example, generated plots, and runtime logs used in the experiments are available at: https://github.com/ChristosPolimatidis/Qiskit-KG-Encodings.

1 Introduction

Motivation. Quantum Computing (QC) is advancing rapidly, and it is widely viewed as a promising paradigm for accelerating classes of computations where classical approaches face scalability limits. Quantum computing research spans several distinct layers, each addressing a fundamental set of questions:
(a) *Physical Substrates:* Selecting the optimal quantum hardware platform (e.g., utilizing electrons/superconducting qubits or photons),
(b) *Architectures and Models:* Determining how to construct a quantum computer and which computational model to adopt (e.g., the gate-based circuit model versus measurement-based or adiabatic quantum computing) [18],
(c) *Algorithmic Foundations:* Developing algorithms tailored specifically for quantum architectures, such as Shor's algorithm [32] and Grover's algorithm [13].
(d) *Practical Applications:* Identifying real-world tasks and industries that would benefit most from being mapped to quantum computing workflows [39] and leverage quantum computing in our software [27].

Here we focus on the last layer, in particular on applications of QC for *data and knowledge management*. We confine ourselves to *Knowledge Graphs* (KGs), since any dataset can be transformed to a KG [25, 26]. However, exploiting QC for KG workloads is not straightforward: quantum processors do not operate directly on symbolic RDF data. Any end-to-end KG-to-QC pipeline, therefore, involves (i) classical preprocessing and numerical encoding, (ii) quantum state preparation, (iii) execution of quantum routines, and (iv) measurement and decoding of results. As a consequence, the encoding choice is not a minor representational detail; it directly impacts qubit requirements, state-preparation complexity, and ultimately the feasibility of KG-related tasks on current (and near-term) quantum hardware.
Novelty. The first analysis of how QC could be leveraged for KG management was described in [35], that investigated tasks like quantum storage, lookup, indexing, satisfiability, and others. Subsequently, [34] reported extensive experimental results related to encoding and superposition. These works focused on *basis encoding*. Here we investigate alternative encodings, i.e. *amplitude* and *phase* encodings. To the best of our knowledge, a systematic side-by-side treatment of basis, amplitude, and phase encodings *specifically for KGs*, together with a task-oriented comparison of when each family is appropriate for representative KG tasks, is still missing.
Approach. For each family of encodings, we (i) describe precisely how KG information is mapped to quantum states, (ii) analyze the main trade-offs from a data-management perspective (qubit counts, state-preparation overhead, measurement/readout implications, and interpretability), and (iii) identify the KG tasks for which the encoding is most suitable. To ensure comparability, we use a single running example across all encodings and use it to highlight

Proceedings of the VLDB Endowment. ISSN 2150-8097.

how encoding decisions affect circuit design and result extraction. The outcome is a task-driven guide that supports informed encoding selection; that is, it clarifies *which encoding to use for which KG task*.

In brief, the **key contributions** of this work are: (a) A unified encoding framework for KGs across basis, amplitude, and phase representations, and (b) A task-oriented mapping between KG workloads and encoding strategies.

Organization. The remainder of this paper is structured as follows. Section 2 reviews related work on QC for data management and KGs, positioning our study with respect to basis-encoding approaches. Section 3 provides background on QC and KGs. Section 4 introduces the running example and presents the three encoding families along with a comparative analysis of their resource and workflow implications. Section 5 reports our experimental results. Finally, Section 6 concludes and outlines directions for future work.

2 Related Work

Works on QC and data management have started to appear; however, papers that report results of experiments on real QC hardware and on realistic data sizes are much fewer. A general remark is that in most of these works, the data/knowledge management perspective is still largely unexplored and quite far from technological and practical exploitation, in the sense that the datasets used are typically too small and/or the evaluation relies mainly on classical simulation. Below we list, and comment in brief, representative related works:

There are a few papers that attempt to form the data management *landscape*, e.g. [14, 15]. There are works on *quantum data structures*, like Quantum Random Access Memory (QRAM) [12], Quantum Bloom filters [31], Quantum B^+-tree [23].

[24] proposes a *quantum machine learning* algorithm for knowledge graphs; however, the reported experimental results are obtained on classical computers (simulators), rather than on quantum hardware. [20] elaborates on *quantum storage design* for tables in RDBMS and provides illustrative examples over very small data instances. [**?**] focuses on *entity matching* with Quantum Neural Networks; it evaluates quantum machine learning models on a small handcrafted dataset and compares them to classical baselines. There are works on *quantum-assisted query optimization* [9, 22, 28], e.g. [22] focused on query optimization, in particular, it formulated join ordering as a Quadratic Unconstrained Binary Optimization (QUBO) problem enriched with database statistics and structural constraints, and showed how quantum-inspired optimization can already offer practical benefits for join ordering. There are also works on *query containment* [11], as well as works on *schema discovery in property graphs*, like [21] that proposes replacing the LSH clustering step with a Quadratic Unconstrained Binary Optimization (QUBO) formulation.

Closer to our focus are works that adopt a data-management viewpoint for Knowledge Graphs (KGs) by explicitly studying *basis* encodings for RDF triples. In particular, [35] introduces and analyzes alternative basis encodings for RDF data and discusses their implications in KG-related workflows, including encoding/decoding aspects and core primitives such as storage and lookup. Subsequently, [34] reported encoding results for datasets of size up to 10^5 bitswords. That work revealed that (a) basis encoding can be done fast,(b) superposition can be done fast, and (c) Grover's algorithm is too slow to apply to a large dataset.

To the best of our knowledge, no prior work provides a unified, KG-centric comparative treatment of *basis, amplitude, and phase* encodings under common criteria, together with explicit guidance on encoding selection for representative KG tasks.

3 Background

3.1 Quantum Computing

In Quantum Computing (QC), information is represented using *qubits* rather than classical bits. A qubit is a unit vector in a two-dimensional complex Hilbert space and can be written as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$ [35]. Unlike a classical bit, which is either 0 or 1, a qubit may exist in a superposition of basis states until measurement. Upon measurement in the computational basis, the outcome is 0 with probability $|\alpha|^2$ and 1 with probability $|\beta|^2$. The state space grows exponentially with the number of qubits. An n-qubit register is described by a superposition over 2^n basis states, $|\psi\rangle = \sum_{i=0}^{2^n-1} c_i|i\rangle$, where the complex amplitudes c_i satisfy $\sum_i |c_i|^2 = 1$. This property is one of the main reasons QC is considered promising for certain computational tasks, since quantum operations act on all amplitudes coherently. At the same time, the result of a quantum computation is not obtained directly from the full state vector, but only through measurement, which makes state preparation and readout central concerns in practice [35].

Quantum computations are performed through unitary transformations, usually expressed as quantum circuits composed of gates acting on one or more qubits. Two quantum-mechanical phenomena are especially important for computation: *superposition*, which allows the simultaneous representation of many basis states, and *interference*, through which amplitudes can be amplified or suppressed so that desired outcomes become more likely after measurement. Another key phenomenon is *entanglement*, which captures correlations between qubits that cannot be described classically and is often exploited in quantum algorithms.

Although the set of practically useful quantum algorithms is still limited, some foundational results illustrate the potential of the paradigm. Shor's algorithm showed that integer factorization can be solved in polynomial time on a quantum computer [32], while Grover's algorithm provides a quadratic speedup for unstructured search, requiring $O(\sqrt{N})$ oracle calls instead of $O(N)$ in the classical case [13]. For this paper, these algorithms are important not only historically, but also conceptually: they show that the choice of encoding must be aligned with the type of computation that follows. In particular, an encoding that is suitable for exact symbolic lookup may be substantially different from one that is suitable for similarity-oriented or interference-based processing.

3.2 Knowledge Graphs and RDF

Knowledge Graphs (KGs) are structured representations of entities, concepts, and their relationships, and they are widely used for integration, search, analytics, and reasoning over heterogeneous data sources [35]. They rely on graph-based data models for representing information in a machine-processable form, and they appear in

a wide range of domains, from general-purpose resources such as DBpedia and Wikidata to domain-specific repositories in science, culture, and industry [4, 37].

In this work, we focus on KGs represented in the *Resource Description Framework* (RDF), which is the dominant model in Semantic Web and Linked Data settings. RDF expresses knowledge through triples of the form (s, p, o), where s is the subject, p the predicate, and o the object. Intuitively, the subject denotes the resource being described, the predicate denotes one of its properties or relations, and the object denotes either another resource or a literal value. Formally, if U, B, and L denote the sets of URIs, blank nodes, and literals, then an RDF triple is an element of $(U \cup B) \times U \times (U \cup B \cup L)$, and any finite set of such triples forms an RDF graph [35].

This representation is attractive because it preserves the symbolic and relational structure of knowledge in a uniform way. At the same time, this symbolic nature is precisely what creates the main challenge for quantum processing: quantum hardware does not operate directly on URIs, literals, or RDF triples. Before a KG can be used as input to a quantum routine, its contents must first be transformed into a suitable numerical representation and then encoded into quantum states. Consequently, when studying quantum methods over KGs, the representation issue is not peripheral but fundamental.

From a data-management viewpoint, KGs support a broad spectrum of tasks, including exact lookup, graph traversal, querying, reasoning, ranking, similarity computation, and question answering. These tasks differ significantly in what they require from the underlying representation. For example, exact triple lookup benefits from a representation that preserves symbolic identity, whereas vector-style similarity or learning tasks are naturally associated with numerical representations. This observation motivates the comparative study of different quantum encodings for KGs carried out in this paper.

4 Basis, Amplitude, and Phase Encodings for KGs

There are several encodings, see [30] for an overview. The choice depends on the task that we want to perform. Here we shall focus on three of them, *basis encoding, amplitude encoding*, and *phase encoding*. The key idea of each one is described below. We shall present the idea of the encodings using a small example. Suppose that we want to represent a set of three integers $\{0, 3, 7\}$.

- **Basis Encoding.** Here, each integer is represented directly as a computational basis state by writing it in binary form, i.e., for the integers $0, 3, 7$ we have: $0 = 000 \rightarrow |000\rangle$, $3 = 011 \rightarrow |011\rangle$, $7 = 111 \rightarrow |111\rangle$. Each value corresponds to a distinct quantum state. If we want to represent all values together, we can form a superposition such as: $\frac{1}{\sqrt{3}}(|000\rangle + |011\rangle + |111\rangle)$. Thus, the data is stored in the choice of basis states themselves.
  Suitability. When data is discrete and symbolic (e.g., integers, bitstrings, labels), and when exact values need to be recovered after measurement, e.g., in algorithms such as Grover's search [13] or classical logic simulations.
- **Amplitude Encoding.** Here, the data is stored in the amplitudes of a quantum state. For the vector $[0, 3, 7]$ we first have to normalize it: $\sqrt{0^2 + 3^2 + 7^2} = \sqrt{58}$. The encoded state becomes: $|\psi\rangle = \frac{0}{\sqrt{58}}|0\rangle + \frac{3}{\sqrt{58}}|1\rangle + \frac{7}{\sqrt{58}}|2\rangle$. Using two qubits (four basis states), we embed this as: $|\psi\rangle = 0|00\rangle + \frac{3}{\sqrt{58}}|01\rangle + \frac{7}{\sqrt{58}}|10\rangle + 0|11\rangle$. Here, the numerical values are stored in the amplitudes of the quantum state.
  Suitability. This encoding is much more compact since multiple values are encoded in one quantum state, but harder to prepare and extract. It can be useful when working with numerical vectors, e.g. in quantum machine learning algorithms [33], in linear algebra problems. In comparison to the basis, state preparation and data extraction are more difficult.
- **Phase Encoding.** Here, the values are encoded into the phases (angles) of a quantum state. We begin with an equal superposition over 2 qubits: $|\psi\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$. We then map the values $0, 3, 7$ to phases, for example, a value x can be mapped to θ_x as follows $\theta_x = \frac{\pi}{4} \cdot x$, in our case: $0 \rightarrow 0$, $3 \rightarrow \frac{3\pi}{4}$, $7 \rightarrow \frac{7\pi}{4}$. Applying these phases gives: $|\psi\rangle = \frac{1}{2}\left(|00\rangle + e^{i\cdot 0}|01\rangle + e^{i\cdot\frac{3\pi}{4}}|10\rangle + e^{i\cdot\frac{7\pi}{4}}|11\rangle\right)$. In this case, the information is stored in the relative phases between basis states.
  Suitability. Useful for interference-based algorithms, like Quantum Fourier Transform (QFT) [7], and in phase estimation and Shor's algorithm [32]. The advantage is that it enables quantum interference, which is key to many quantum speedups, but information is not directly observable and requires interference to extract.

A diagrammatic overview of the three encodings, over the set $\{0, 3, 7\}$ is given in Figure 1. Next, we introduce a running example, that we shall use to detail the proposed encodings.

Example 4.1. We use a small example KG with the following triples:

$t_0 = (\texttt{ex:Aristotle}, \texttt{rdf:type}, \texttt{ex:Person})$
$t_1 = (\texttt{ex:Person}, \texttt{rdfs:subClassOf}, \texttt{ex:Mortal})$
$t_2 = (\texttt{ex:Aristotle}, \texttt{ex:livesAt}, \texttt{ex:Athens})$
$t_3 = (\texttt{ex:Athens}, \texttt{rdf:type}, \texttt{ex:City})$
$t_4 = (\texttt{ex:City}, \texttt{rdfs:subClassOf}, \texttt{ex:Place})$
$t_5 = (\texttt{ex:Aristotle}, \texttt{ex:teaches}, \texttt{ex:Philosophy})$

The small graph contains standard RDF triples with clearly distinguishable subject, predicate, and object components, which makes it suitable for direct symbolic representation under basis encoding. The graph contains both instance-level assertions and schema-level knowledge. For example, the triple $(\texttt{ex:Person}, \texttt{rdfs:subClassOf}, \texttt{ex:Mortal})$ expresses a class inclusion, while $(\texttt{ex:Aristotle}, \texttt{rdf:type}, \texttt{ex:Person})$ expresses a membership assertion.

4.1 Basis Encoding for RDF

Basis encoding is the most direct and symbolic of the three approaches. Each KG resource (entity or relation) is mapped to a unique integer identifier, which is then written in binary, and an RDF triple (s, p, o) is represented by concatenating the bit strings of its components. In this way, each triple corresponds to exactly one computational basis state $|s\rangle \otimes |p\rangle \otimes |o\rangle$, and the whole graph

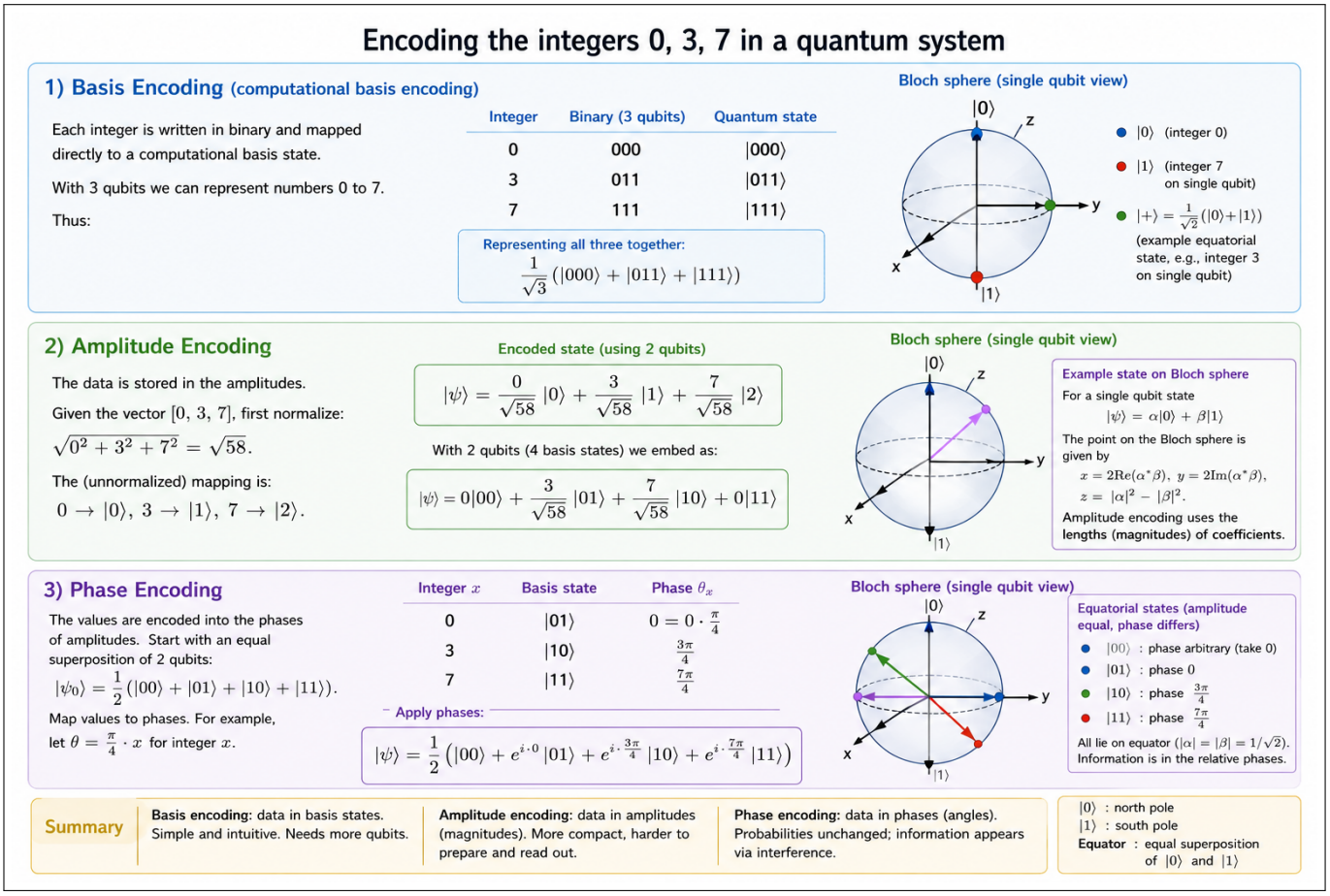

Figure 1: Overview of encodings

can be stored either as a set of such states or as a uniform superposition over them. Because symbolic identity is preserved explicitly, measurement returns a full triple, and basis encoding is naturally aligned with exact tasks such as triple lookup, membership testing, and pattern matching. Concretely, the encoding proceeds in three steps: (i) indexing entities and relations and assigning them binary codes, (ii) encoding each triple as the tensor product of the codes of its subject, predicate, and object, and (iii) representing the graph as a set or superposition of the resulting basis states. We illustrate these steps on the running example below.

Example 4.2. Our running example goes through the following steps:

Step 1: *Entity and Relation Indexing*. We assign binary encodings to entities and relations. In our case, we need 3 qubits for the entity representation:

$$\texttt{ex:Aristotle} \to |000\rangle$$
$$\texttt{ex:Person} \to |001\rangle$$
$$\texttt{ex:Mortal} \to |010\rangle$$
$$\texttt{ex:Athens} \to |011\rangle$$
$$\texttt{ex:City} \to |100\rangle$$
$$\texttt{ex:Place} \to |101\rangle$$
$$\texttt{ex:Philosophy} \to |110\rangle$$

For the relations, we only need 2 qubits:

$$\texttt{rdf:type} \to |00\rangle$$
$$\texttt{rdfs:subClassOf} \to |01\rangle$$
$$\texttt{ex:livesAt} \to |10\rangle$$
$$\texttt{ex:teaches} \to |11\rangle$$

Step 2: *Triple Encoding*. We encode each triple according to the entity/relation encoding, i.e., each triple is encoded as: $|s\rangle \otimes |p\rangle \otimes |o\rangle$,

thus our 6 triples are encoded as:

$$t_0 \rightarrow |000\rangle|00\rangle|001\rangle$$
$$t_1 \rightarrow |001\rangle|01\rangle|010\rangle$$
$$t_2 \rightarrow |000\rangle|10\rangle|011\rangle$$
$$t_3 \rightarrow |011\rangle|00\rangle|100\rangle$$
$$t_4 \rightarrow |100\rangle|01\rangle|101\rangle$$
$$t_5 \rightarrow |000\rangle|11\rangle|110\rangle$$

Step 3: *Graph Representation.* In basis encoding, the graph is represented as a set (or superposition) of basis states, i.e. $G = \{|000\rangle|00\rangle|001\rangle,\ |001\rangle|01\rangle|010\rangle,\ |000\rangle|10\rangle|011\rangle, |011\rangle|00\rangle|100\rangle, |100\rangle|01\rangle|101\rangle,\ |000\rangle|11\rangle|110\rangle\}$ which can be superimposed as: $|\psi_G\rangle = \frac{1}{\sqrt{6}} \sum_{k=0}^{5} |s_k\rangle|p_k\rangle|o_k\rangle$.

4.1.1 Search/Query over Basis-Encoded RDF Graph. A typical task supported by basis encoding is exact search over the triples of the graph. Since each triple is stored as a distinct basis state, we can retrieve the triples matching a given pattern by marking the corresponding states and then amplifying their probability, which is precisely the mechanism provided by Grover's algorithm [13]. The procedure consists of four steps: (i) constructing an oracle that flips the phase of the states satisfying the query condition, (ii) applying the oracle to the graph state, (iii) applying Grover diffusion to amplify the marked states, and (iv) measuring to recover the matching triples with high probability. We illustrate this on the query that retrieves all triples with predicate p = `rdf:type`, recalling that according to the predicate encoding `rdf:type` $\rightarrow |00\rangle$.

Example 4.3. Continuing our example:
Step 1: Oracle Construction. We define an oracle U_f that marks triples with $p = |00\rangle$, i.e. $U_f|s\rangle|p\rangle|o\rangle = (-1)^{f(p)}|s\rangle|p\rangle|o\rangle$, where:

$$f(p) = \begin{cases} 1 & \text{if } p = |00\rangle \\ 0 & \text{otherwise} \end{cases}$$

Step 2: Application of Oracle on State, i.e., application of U_f:

$$|\psi'\rangle = \frac{1}{\sqrt{6}}\Big(- |000\rangle|00\rangle|001\rangle - |011\rangle|00\rangle|100\rangle + (\text{other terms})\Big)$$

Note that only triples with `rdf:type` receive a phase flip.
Step 3: Amplitude Amplification. We apply Grover diffusion to amplify marked states:

$$|\psi_{\text{final}}\rangle \approx \frac{1}{\sqrt{2}}\Big(|000\rangle|00\rangle|001\rangle + |011\rangle|00\rangle|100\rangle\Big)$$

The amplification is done iteratively; the number of iterations k depends on the size of the search space N (typically $N = 2^n$ for an n-qubit system), and the number of solutions (or marked items) M, specifically $k \approx \frac{\pi}{4}\sqrt{\frac{N}{M}}$.
Step 4: Measurement. Measuring the state returns:
(`ex:Aristotle`, `rdf:type`, `ex:Person`) or
(`ex:Athens`, `rdf:type`, `ex:City`) with high probability.

Apart from search, note that other tasks like satisfiability (and consequently concept satisfiability, subsumption checking, instance checking) can be performed using Grover's algorithm, assuming basis encoding, as described in [35].

4.2 Amplitude Encoding

Amplitude encoding stores a classical vector in the amplitudes of a quantum state rather than in separate symbolic registers. Instead of representing each subject, predicate, and object explicitly, the triples of the graph are first indexed classically, a numerical vector over those indices is built and normalized, and the result is loaded as a quantum state in which each basis state $|i\rangle$ corresponds to one triple and its amplitude encodes the associated value. Its main advantage is qubit efficiency, since N values are encoded in $\lceil \log_2 N \rceil$ qubits, but the trade-off is that state preparation and readout become harder: measurement samples an index according to the squared amplitudes, so the symbolic triple must be recovered through a classical index-to-triple dictionary. This makes amplitude encoding well-suited for numerical and similarity-oriented tasks, where inner products between such states measure graph similarity, and the entries may carry weights, confidence values, frequencies, or embedding-derived scores. The construction proceeds in five steps: (i) entity and relation indexing, (ii) mapping each triple to a unique index, (iii) building a sparse graph vector, (iv) normalizing it, and (v) loading it as the amplitude-encoded state. We detail these steps in the running example.

Example 4.4. Our running example goes through the following steps:
Step 1: Entity and Relation Indexing, as done in Basis encoding.
Step 2: Triple Index Encoding. Each triple (s, p, o) is mapped to a unique index:

$$i = s \cdot (|P| \cdot |E|) + p \cdot |E| + o$$

where $|E| = 7$ and $|P| = 4$. Thus, the vector space has dimension:

$$|E| \times |P| \times |E| = 7 \cdot 4 \cdot 7 = 196.$$

The indices of our triples are:

$$t_0 : (0, 0, 1) \Rightarrow i_0 = 0 \cdot 28 + 0 \cdot 7 + 1 = 1$$
$$t_1 : (1, 1, 2) \Rightarrow i_1 = 1 \cdot 28 + 1 \cdot 7 + 2 = 37$$
$$t_2 : (0, 2, 3) \Rightarrow i_2 = 0 \cdot 28 + 2 \cdot 7 + 3 = 17$$
$$t_3 : (3, 0, 4) \Rightarrow i_3 = 3 \cdot 28 + 0 \cdot 7 + 4 = 88$$
$$t_4 : (4, 1, 5) \Rightarrow i_4 = 4 \cdot 28 + 1 \cdot 7 + 5 = 124$$
$$t_5 : (0, 3, 6) \Rightarrow i_5 = 0 \cdot 28 + 3 \cdot 7 + 6 = 27$$

Step 3: Graph Vector. We build a sparse vector $\mathbf{v}_G \in \mathbb{R}^{196}$:

$$v_i = \begin{cases} 1 & \text{if } i \in \{1, 17, 27, 37, 88, 124\} \\ 0 & \text{otherwise} \end{cases}$$

Step 4: Normalization. There are 6 triples, so $\|\mathbf{v}_G\| = \sqrt{6}$, Thus, the normalized vector is:

$$\tilde{v}_i = \begin{cases} \frac{1}{\sqrt{6}} & \text{if triple exists} \\ 0 & \text{otherwise} \end{cases}$$

Step 5: Quantum State. The amplitude-encoded quantum state is:

$$|\psi_G\rangle = \frac{1}{\sqrt{6}}\big(|1\rangle + |17\rangle + |27\rangle + |37\rangle + |88\rangle + |124\rangle\big)$$

In summary, each basis state $|i\rangle$ corresponds to a specific RDF triple, whereas the graph is represented as a superposition of its triples. The gain is that inner products between such states measure graph similarity, and amplitude encoding is well-suited for representing high-dimensional numerical data, such as embeddings

commonly used in KGs. Note that since the swap test is fundamentally an inner-product estimator, it can be used for computing the cosine similarity, the Euclidean distance (if the norms are known), as well as various kernel functions. It cannot be used, however, for directly computing other similarities, like Jaccard similarity.

Note that we can also use *node-centric encodings*. For example, for `ex:Aristotle`, we could encode only its local triples: $\{t_0, t_2, t_5\}$, i.e. $|\psi_{\texttt{Aristotle}}\rangle = \frac{1}{\sqrt{3}}(|1\rangle + |17\rangle + |27\rangle)$ This is particularly useful for *entity matching* [2, 10] and *blank node matching* [3, 19, 36], and other similarity-related tasks like *similarity-based browsing* [6]. Below, we describe three applications of amplitude encoding.

4.2.1 **Entity Matching.** Entity matching aims to detect that two entities in different KGs refer to the same real-world object (e.g., that "Paris" in KG_1 and "Paris" in KG_2 are identical). Each entity is described by an embedding vector, which is amplitude-encoded into a quantum state, and the similarity of two such states is then estimated through a swap test [1]. The swap test uses an ancilla qubit and two Hadamard gates around a controlled swap, so that the probability of measuring the ancilla in $|0\rangle$ reflects the overlap of the two states. We illustrate the idea on a pair of toy embedding vectors below.

Example 4.5. Suppose the two entities are represented by the embeddings $\mathbf{e}_1 = [0.1, 0.7, 0.2], \quad \mathbf{e}_2 = [0.12, 0.68, 0.22]$. Using amplitude encoding, these vectors are encoded into quantum states: $|\psi_1\rangle = \sum_i \alpha_i |i\rangle, \quad |\psi_2\rangle = \sum_i \beta_i |i\rangle$. A swap test then estimates their similarity through the following circuit: (1) Prepare an ancilla qubit in state $|0\rangle$, (2) Apply a Hadamard gate: $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, (3) Conditionally swap $|\psi_1\rangle$ and $|\psi_2\rangle$, (4) Apply another Hadamard and measure. The probability of measuring $|0\rangle$ is $P(0) = \frac{1}{2}\left(1 + |\langle\psi_1|\psi_2\rangle|^2\right)$. An illustration of the method is given in Figure 2.

Evidently, this approach allows for efficient representation of large embedding vectors and fast similarity estimation.

4.2.2 **Link Prediction.** In KG completion, the goal is to predict missing triples such as (Alice, lives_in, ?), a task usually called link prediction [38]. Entities and relations are embedded into vectors (e.g., TransE-style embeddings [5]), which can be amplitude-encoded so that quantum linear-algebra routines compute scores or distances. In translation-based scoring (TransE), the plausibility of a triple (h, r, t) with embeddings $\mathbf{e}_h, \mathbf{r}, \mathbf{e}_t \in \mathbb{R}^d$ is measured by $\text{score}(h, r, t) = \|\mathbf{e}_h + \mathbf{r} - \mathbf{e}_t\|$, the geometric idea being that the relation acts as a translation vector enforcing $\mathbf{e}_h + \mathbf{r} \approx \mathbf{e}_t$. A small distance indicates a likely true triple, while a large distance indicates a likely false one. We illustrate the scoring on a triple from the running example.

Example 4.6. Consider the triple $(\texttt{ex:Aristotle}, \texttt{ex:livesAt}, \texttt{ex:Athens})$. Under the translation-based interpretation, we expect: $\mathbf{e}_{\texttt{Aristotle}} + \mathbf{r}_{\texttt{livesAt}} \approx \mathbf{e}_{\texttt{Athens}}$. The plausibility of the triple is then assessed through the score:

- If $\|\mathbf{e}_h + \mathbf{r} - \mathbf{e}_t\| \approx 0$, the triple is likely *true*.
- If $\|\mathbf{e}_h + \mathbf{r} - \mathbf{e}_t\|$ is large, the triple is likely *false*.

Here $\|\mathbf{e}_h + \mathbf{r} - \mathbf{e}_t\|$ measures how well the relation $\mathbf{r}$ transforms the head embedding into the tail embedding in the vector space.

The connection to phase-based (Quantum) Models is that in translation-based models: $\mathbf{e}_h + \mathbf{r} \approx \mathbf{e}_t$. In phase-based (complex or quantum-inspired) models, relations are modeled as rotations: $\mathbf{e}_t \approx \mathbf{e}_h \cdot e^{i\theta_r}$. Thus, TransE uses *vector addition* (translations), while phase-based models use *complex multiplication* (rotations). Using amplitude encoding, vectors are mapped to quantum states, and algorithms like *HHL (Harrow-Hassidim-Lloyd)* [16] can be used for solving linear systems or distance estimation. As such, this encoding enables compact storage and potential speedups for large-scale vector operations. The limitation is that it requires efficient state preparation and well-conditioned matrices.

4.2.3 **Keyword Search.** Keyword search over RDF KGs [8, 17, 29] is another task well-suited to amplitude encoding. Each entity is associated with a classical feature vector built from the triples it participates in; both the entity vectors and the query vector are amplitude-encoded, and their similarity is computed as $\text{score}(e) = |\langle\psi_q|\psi_e\rangle|^2$ using a swap test. The entities with the highest scores are returned as results. We illustrate the full pipeline on the running example, using the feature space $\mathcal{V} = \{\texttt{Aristotle}, \texttt{Athens}, \texttt{Person}, \texttt{City}, \texttt{Philosophy}\}$.

Example 4.7. Each entity is represented as a vector indicating its contextual features (from triples). Since Aristotle appears in triples involving Person, Athens, and Philosophy, it is associated with the vector $\mathbf{v}_{\texttt{Aristotle}} = (1, 1, 1, 0, 1)$, while Athens with $\mathbf{v}_{\texttt{Athens}} = (0, 1, 0, 1, 0)$. After normalization, these become $|\psi_{\texttt{Aristotle}}\rangle = \frac{1}{2}(|1\rangle + |2\rangle + |3\rangle + |5\rangle)$, $|\psi_{\texttt{Athens}}\rangle = \frac{1}{\sqrt{2}}(|2\rangle + |4\rangle)$. Now, suppose an incoming query $q = \{\texttt{Aristotle}, \texttt{Athens}\}$. We define $\mathbf{v}_q = (1, 1, 0, 0, 0)$, and normalize $|\psi_q\rangle = \frac{1}{\sqrt{2}}(|1\rangle + |2\rangle)$. The similarity is computed by $\text{score}(e) = |\langle\psi_q|\psi_e\rangle|^2$. For Aristotle, $\langle\psi_q|\psi_{\texttt{Aristotle}}\rangle = \frac{1}{\sqrt{2}} \cdot \frac{1}{2}(1 + 1) = \frac{1}{\sqrt{2}}$, thus $\text{score}(\texttt{Aristotle}) = \frac{1}{2}$. For Athens, $\langle\psi_q|\psi_{\texttt{Athens}}\rangle = \frac{1}{\sqrt{2}} \cdot \frac{1}{\sqrt{2}}(0+1) = \frac{1}{2}$, thus $\text{score}(\texttt{Athens}) = \frac{1}{4}$. Overall, in the final ranking, we have $\texttt{Aristotle} > \texttt{Athens}$, thus `Aristotle` is ranked as more relevant.

4.3 Phase Encoding for KGs

Phase encoding stores information in the relative phases of basis states rather than in their amplitudes. The starting point is a uniform superposition over the indexed triples; a phase is then attached to each basis state according to some value, rule, or condition, while the magnitudes of the amplitudes remain unchanged. A particularly important special case is binary marking, where the phase is either 0 or π, so that selected states receive a sign flip ($e^{i\pi} = -1$). Because the magnitudes are untouched, a direct measurement in the computational basis does not reveal the phase pattern; the encoded information becomes observable only after a subsequent mixing or interference step (e.g., Hadamard transforms, diffusion operators, or the QFT) converts phase differences into amplitude differences. This makes phase encoding natural for marking, filtering, and interference-based processing, such as oracle construction and relational reasoning. For the running example, the construction proceeds in four steps: (i) indexing the triples, (ii) preparing an initial uniform superposition, (iii) assigning a phase to each basis state, and (iv) obtaining the phase-encoded state.

Example 4.8. We continue our running example:
Step 1: Indexing Triples. As in amplitude encoding, we assign each triple a unique index $i \in \{1, 17, 27, 37, 88, 124\}$. We assume a Hilbert

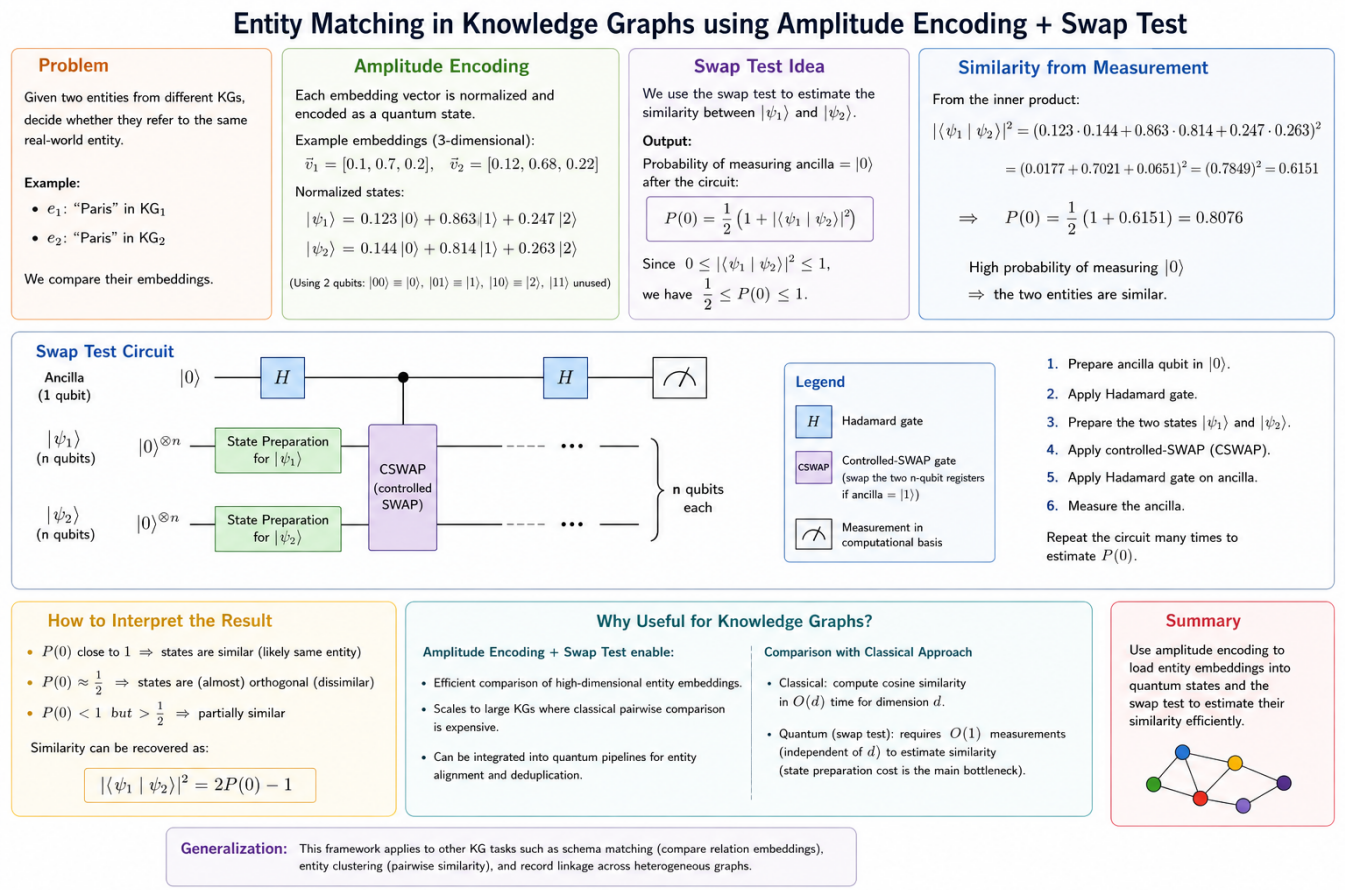

Figure 2: Entity Matching

space of dimension $2^8 = 256$ (8 qubits), where each basis state $|i\rangle$ corresponds to a possible triple.

Step 2: Initial Superposition. We start from a *uniform* superposition over all basis states: $|\psi_0\rangle = \frac{1}{\sqrt{256}} \sum_{i=0}^{255} |i\rangle$.

Step 3: Phase Assignment. We encode the *presence* of triples by applying phases. For simplicity, assign:

$$\theta_i = \begin{cases} \pi & \text{if } i \in \{1, 17, 27, 37, 88, 124\} \\ 0 & \text{otherwise} \end{cases}$$

Step 4: Phase-Encoded State. After applying phase shifts, we have: $|\psi_G\rangle = \frac{1}{\sqrt{256}} \sum_{i=0}^{255} e^{i\theta_i} |i\rangle$. [1] Equivalently

$$|\psi_G\rangle = \frac{1}{\sqrt{256}} \left(\sum_{i \notin T} |i\rangle - \sum_{i \in T} |i\rangle \right),$$

where $T = \{1, 17, 27, 37, 88, 124\}$. So triples get a minus sign, everything else stays positive. So we are not changing probabilities, we are changing relative phases (signs) and this matters because quantum algorithms use interference, not just probabilities.

As shown, all basis states have equal probability, the KG is encoded in the *relative phases*, in particular present triples introduce a phase flip (π).

An alternative relation-based phase encoding is also possible. We can also encode relations as phases: $\theta_{\texttt{rdf:type}} = \theta_0$, $\theta_{\texttt{rdfs:subClassOf}} = \theta_1$, $\theta_{\texttt{ex:livesAt}} = \theta_2$, $\theta_{\texttt{ex:teaches}} = \theta_3$. Then each triple contributes: $|s, p, o\rangle \to e^{i\theta_p} |s, p, o\rangle$. So the graph state becomes:
$|\psi_G\rangle = \frac{1}{\sqrt{6}} \sum_{k=0}^{5} e^{i\theta_{p_k}} |t_k\rangle$.

Note that phase encoding is not a full reasoning algorithm by itself. Rather, it can model limited compositional reasoning patterns, such as phase accumulation along relation chains, interference between different relational paths, and the detection of consistent relational patterns.

The key difference from amplitude Encoding is that in amplitude encoding we have $|\psi\rangle = \sum_i \alpha_i |i\rangle$ (data in magnitudes), while in phase encoding we have $|\psi\rangle = \sum_i e^{i\theta_i} |i\rangle$ (data in phases). The intuition is that amplitude encoding can enable us to answer "Which triples exist?" (since they are encoded amplitudes), while phase encoding enables us to answer "How triples relate/interfere?" (since their interaction is encoded in phases). Particular examples follow.

[1] This comes from Euler's formula: $e^{i\theta} = cos\theta + isin\theta$. We choose: $\theta_i = 0$ for non-triples, so $e^{i0} = 1$, and $\theta_i = \pi$ for triples, so $e^{i\pi} = -1$.

4.3.1 **Multi-hop Reasoning.** Multi-hop reasoning chains several relations to infer a new fact, and it maps naturally to phase encoding because composing relations corresponds to accumulating their phases. Each relation is modeled as a phase rotation $U_r|e\rangle = e^{i\theta_r}|e\rangle$; applying a sequence of relations to an initial entity state accumulates the sum of the corresponding phases, and constructive interference among aligned phases reinforces the correct inferred fact while destructive interference suppresses incorrect ones. We illustrate this on a two-hop inference from the running example, namely deriving (`ex:Aristotle`, `rdf:type`, `ex:Mortal`) from a type assertion followed by a subclass assertion.

Example 4.9. Consider the two triples: (`ex:Aristotle`, `rdf:type`, `ex:Person`) and (`ex:Person`, `rdfs:subClassOf`, `ex:Mortal`). From these, we want to infer the new fact: (`ex:Aristotle`, `rdf:type`, `ex:Mortal`). We encode each relation as a phase rotation: $U_r|e\rangle = e^{i\theta_r}|e\rangle$, with $\theta_1 = \theta_{\texttt{rdf:type}}$, $\theta_2 = \theta_{\texttt{rdfs:subClassOf}}$. Starting with the entity state $|\psi_0\rangle = |\texttt{ex:Aristotle}\rangle$, we apply the first relation: $|\psi_1\rangle = U_{\texttt{rdf:type}}|\psi_0\rangle = e^{i\theta_1}|\texttt{ex:Person}\rangle$, then the second relation: $|\psi_2\rangle = U_{\texttt{rdfs:subClassOf}}|\psi_1\rangle = e^{i\theta_2}e^{i\theta_1}|\texttt{ex:Mortal}\rangle$, so that $|\psi_2\rangle = e^{i(\theta_1+\theta_2)}|\texttt{ex:Mortal}\rangle$. The composed reasoning path, therefore, corresponds to *phase accumulation*: $\theta_{\text{total}} = \theta_{\texttt{rdf:type}} + \theta_{\texttt{rdfs:subClassOf}}$. If this phase matches the phase associated with a direct relation (or aligns constructively in a larger superposition), the inference is reinforced via interference. If multiple reasoning paths exist, we can represent them as

$$|\psi\rangle = \frac{1}{\sqrt{K}} \sum_k e^{i\theta_k} |\text{candidate}_k\rangle,$$

where constructive interference amplifies correct conclusions, such as $|\texttt{ex:Mortal}\rangle$, while destructive interference suppresses incorrect ones.

Multi-hop reasoning in knowledge graphs can thus be modeled as: (a) Sequential application of phase operators (relations), (b) Accumulation of phases along paths, (c) Interference to select valid inferred facts. This is suited for transitive and compositional rules, where reasoning corresponds to chaining relations, not any kind of inference rule. Other reasoning rules that are not purely compositional cannot be captured by simple phase addition.

4.3.2 **Schema Matching.** Schema matching aims to detect that relations from different KGs are semantically equivalent (e.g., "author_of" and "written_by"). If relations are encoded as phases, and semantically similar relations yield aligned phases ($e^{i\theta_1} \approx e^{i\theta_2}$), and the Quantum Fourier Transform (QFT) [7] can be used to compare their phase patterns by converting phase information (in index space) into frequency information (in Fourier space). This matters because schema matching is not merely about whether two relations are equal, but about whether they behave similarly across the graph. We sketch the idea below.

Example 4.10. Suppose two KGs use different schemas: "author_of" vs. "written_by". These relations are encoded as phases, and if they are semantically similar, their phases align: $e^{i\theta_1} \approx e^{i\theta_2}$. We then apply the QFT to the phase-encoded relations to extract frequency/phase information: $\text{QFT}\left(\sum_x e^{i\theta_x}|x\rangle\right)$, which converts phase patterns (in time/index space) into frequency patterns (in Fourier space). In this way, two relations that behave similarly across the graph produce similar Fourier patterns.

This encoding can be used for identifying similar relation patterns, and detecting periodic or structural similarities across schemas.

Note. Above, we have described some KG tasks in their plain form. Sophisticated versions of these tasks may need more care. For instance, if we want similarity to account for semantic relationships such as synonymy (e.g., owl:sameAs), then the encoding should capture not only the lexical content but also the graph semantics. In that case, a hybrid amplitude-phase encoding should be used.

4.4 Comparison of Encodings

In brief, basis encoding is useful for exact symbolic representation, amplitude encoding for compact numerical representation, while phase encoding for relational/interference representation Table 1 summarizes, i.e., the first column contains tasks, the second, the suitable encoding, and the last, the quantum algorithm to be used.

KG Task	Encoding	Quantum Method
Search	Basis	Grover's algorithm
Entity Matching	Amplitude	Swap Test
Link Prediction	Amplitude	HHL / Distance Estimation
Keyword Search	Amplitude	Swap Test
Multi-hop Reasoning	Phase	Phase Kickback
Schema Matching	Phase	QFT

Table 1: Tasks - Encoding - Quantum Algorithms

4.5 Combining Amplitude & Phase Encoding

We now combine amplitude and phase encoding so that a single state carries both (a) the presence or importance of triples, stored in the amplitudes, and (b) their relational semantics, stored in the phases. The target states have the form

$$|\psi\rangle = \sum_i \alpha_i e^{i\theta_i} |i\rangle,$$

where the amplitude α_i answers *"How important is this triple?"* and the phase θ_i answers *"What role does it play?"*. The intuition is that amplitude encoding alone is good for data, phase encoding alone is good for reasoning, while their combination is well-suited to knowledge graphs, which contain both data and reasoning rules. In effect, this is a quantum analogue of classical KG embeddings, where classical embeddings (TransE, RotatE, etc.) are real vectors, the combined encoding uses complex amplitudes. The construction proceeds in four steps: (i) indexing the triples, (ii) assigning amplitudes based on importance, (iii) assigning a relation-based phase to each triple via its predicate, and (iv) combining these into a single complex-coefficient state. We detail these steps in the running example.

Example 4.11. Implementing the aforementioned steps in our running example, we get:

Step 1: Triple Indexing. Using the same indexing as before, we have: $T = \{1, 17, 27, 37, 88, 124\}$. Each index corresponds to one RDF triple.

Step 2: Amplitude Assignment. We assign amplitudes based on importance, for simplicity assume:

$$\alpha_i = \begin{cases} \frac{1}{\sqrt{6}} & \text{if } i \in T \\ 0 & \text{otherwise} \end{cases}$$

Step 3: Phase Assignment (Relation-Based). Assign a phase to each predicate:

$$\theta_{\texttt{rdf:type}} = \theta_0$$
$$\theta_{\texttt{rdfs:subClassOf}} = \theta_1$$
$$\theta_{\texttt{ex:livesAt}} = \theta_2$$
$$\theta_{\texttt{ex:teaches}} = \theta_3$$

Each triple $t_k = (s_k, p_k, o_k)$ gets phase: $\theta_k = \theta_{p_k}$.
Step 4: Combined Quantum State. The RDF graph is encoded as:

$$|\psi_G\rangle = \sum_{k \in T} \alpha_k \, e^{i\theta_k} |k\rangle.$$

Concretely:

$$|\psi_G\rangle = \frac{1}{\sqrt{6}} \Big(e^{i\theta_0}|1\rangle + e^{i\theta_1}|37\rangle + e^{i\theta_2}|17\rangle + e^{i\theta_0}|88\rangle + e^{i\theta_1}|124\rangle + e^{i\theta_3}|27\rangle \Big).$$

Summary: The amplitudes encode *which triples exist*, the phases encode *type of relation*, the same relation ⇒ same phase, and related paths ⇒ phase accumulation.

This combined representation also supports reasoning via phase composition. As with the multi-hop case, composing two relations accumulates their phases, so that the relevant paths in the graph state can interfere. We illustrate this on the same two-hop inference used earlier.

Example 4.12. Consider the following triples: (`ex:Aristotle`, `rdf:type`, `ex:Person`) and (`ex:Person`, `rdfs:subClassOf`, `ex:Mortal`). Their combined phase is $\theta_0 + \theta_1$. This corresponds to inferred knowledge: (`ex:Aristotle`, `rdf:type`, `ex:Mortal`).

The gain is that this (a) supports similarity (via amplitudes), (b) supports reasoning (via phases), and (c) enables interference between relational paths.

Circuit construction. The state $|\psi_G\rangle = \sum_k \alpha_k e^{i\theta_k}|k\rangle$ can be prepared with a circuit that first loads the amplitudes and then decorates the basis states with the appropriate phases. This is done in three steps: (i) initializing the qubit register, (ii) preparing the amplitude superposition over the triple indices, and (iii) applying basis-state-dependent phase rotations. We outline the construction below.

Example 4.13. : The example proceeds as follows:
Step 1: Initialize Qubits. We need $n = 8$ qubits since $2^8 = 256$: $|0\rangle^{\otimes 8}$.
Step 2: Amplitude Preparation. We prepare: $|\psi_A\rangle = \frac{1}{\sqrt{6}} \sum_{k \in T} |k\rangle$. This can be done using state preparation circuits (Mottonen / QRAM-based), or a sequence of controlled rotations, conceptually:

$$|0\rangle^{\otimes 8} \xrightarrow{U_{\text{amp}}} \frac{1}{\sqrt{6}} \sum_{k \in T} |k\rangle.$$

Step 3: Phase Encoding. Apply phase rotations conditioned on the basis state: $U_{\text{phase}}|k\rangle = e^{i\theta_k}|k\rangle$. This is implemented using controlled R_z gates.

4.6 Scalability Comparison of Quantum Data Encodings

Having seen how each family represents the running example, we now compare basis, amplitude, and phase encoding along the dimensions that matter for scaling to larger RDF graphs: the number of qubits required, the cost of state preparation, the way data is accessed at measurement time, and the resulting suitability for large-scale graphs.

4.6.1 Basis Encoding. In basis encoding each triple (s, p, o) is represented explicitly, so a single triple needs $n = \lceil \log_2 |E| \rceil + \lceil \log_2 |P| \rceil + \lceil \log_2 |E| \rceil$ qubits, and a graph of N triples is handled either with separate runs of $O(n)$ qubits each or with a single $O(n)$-qubit register holding a superposition over the N triple states. State preparation is $O(N)$, since every triple must be loaded explicitly into the state. At readout, lookup can be carried out with Grover's algorithm at cost $O(\sqrt{N})$, and because measurement returns a full triple, the symbolic structure is recovered directly. Although superposition lets a large number of triples share the same n qubits, the representation does not compress the data itself. Overall, basis encoding is best for symbolic tasks rather than for large-scale compression.

4.6.2 Amplitude Encoding. Amplitude encoding is far more compact: a vector of size N fits in $n = \lceil \log_2 N \rceil$ qubits, giving exponential compression in space as N data points are stored in $\log N$ qubits. This compactness comes at the cost of state preparation, which is $O(N)$ in the general case and only drops to $O(\log N)$ under additional assumptions such as QRAM. Measurement is probabilistic, returning a single index per shot, so recovering the full distribution requires repeated sampling. Its space efficiency is therefore excellent, but state preparation is the bottleneck, making amplitude encoding ideal for large-scale numerical data and embeddings.

4.6.3 Phase Encoding. Phase encoding uses the same qubit budget as amplitude encoding, $n = \lceil \log_2 N \rceil$. Preparing the initial uniform superposition takes only $O(n)$ time (a layer of Hadamards), and the cost of the subsequent phase assignment depends on its structure: a naive, value-by-value assignment is $O(N)$, but when the phase is a function $\theta(x)$ of the index rather than an arbitrary stored value, it can be realized with a few gates in $O(\text{poly}(n))$. Phases are not directly observable and become useful only after an interference step, such as a QFT or phase estimation. The result is a good qubit efficiency that is genuinely efficient when the phase function is structured, which makes phase encoding best suited to reasoning and relational structure rather than to raw data storage.

4.6.4 Combined Encoding. The combined encoding inherits the $O(\log N)$ qubit count of the amplitude and phase representations, with a preparation cost of $O(N)$ dominated by the amplitude-loading step. In exchange, it captures both views at once: the amplitudes provide the data representation, while the phases carry the relational semantics, giving the most expressive single-state representation among the four.

4.6.5 Comparison. Table 2 summarizes the comparison. The main distinction is in qubit count: basis encoding scales with the vocabulary of the graph, $O(\log |E| + \log |P|)$, whereas the amplitude, phase, and combined encodings scale with the number of triples, $O(\log N)$.

Across all four, state preparation remains the dominant cost and the principal obstacle to scaling on near-term hardware.

Encoding	Qubits	Prep Cost	Scalability
Basis	$O(\log \lvert E\rvert + \log \lvert P\rvert)$	$O(N)$	Good (if superposition)
Amplitude	$O(\log N)$	$O(N)$	Excellent (space)
Phase	$O(\log N)$	$O(N)$ or better	Good (structured cases)
Combined	$O(\log N)$	$O(N)$	Best trade-off

Table 2: Scalability comparison

5 Experimental Validation

This section validates the implementation-level behavior of the encoding families discussed in the previous sections. The goal is deliberately limited: we test whether the proposed RDF-to-quantum pipeline can construct the corresponding states or circuits, execute the task-specific quantum primitives, and recover the expected quantities from measurements. We do not claim quantum advantage, hardware scalability, or superiority over optimized classical KG systems. In this paper, an encoding is called suitable for a task only in a relative sense: among the encodings considered here, it preserves the information required by the task and naturally supports the corresponding quantum primitive. For example, basis encoding preserves symbolic identity for lookup, amplitude encoding supports vector-similarity tasks, and phase encoding supports phase-composition and phase-pattern tasks.

Experimental setup. All experiments were executed with Qiskit Aer using a fixed seed and the same running example introduced earlier. The main validation run used 10,000 shots and three repetitions for each shot-based task. The software environment was Windows 11, Python 3.13.5, Qiskit 2.5.0, Qiskit Aer 0.17.2, NumPy 2.5.1, RDFLib 7.6.0 and Matplotlib 3.11.0. The machine had 12 logical CPUs and 39.75 GiB of RAM. All generated artifacts, including raw counts, CSV tables, JSON outputs, plots, and logs were written to a single run directory.

Validation protocol. Each task follows the same four-step protocol. First, the RDF triples and any task-specific classical vectors are constructed. Second, the corresponding quantum state or circuit is built. Third, the circuit is simulated with 10,000 shots when measurements are required. Fourth, the measured bitstrings or ancilla outcomes are decoded into the task metric and compared with an exact classical calculation on the same toy input. The validation, therefore, records not only runtimes but also the measured quantities, raw counts, derivation formulas, and pass/fail thresholds.

For the measured quantities, we used the following derivations. For Grover-style lookup, the success probability is the number of target-bitstring counts divided by the number of shots. For swap-test tasks, the squared similarity is estimated as $\widehat{s} = \max(0, \min(1, 2\widehat{P}(0) - 1))$, where $\widehat{P}(0)$ is the measured probability of the ancilla being 0. For the multi-hop phase experiment, X- and Y-quadrature Hadamard tests estimate the composed phase as $\widehat{\theta} = \operatorname{atan2}\left(2\widehat{P}_Y(0) - 1, 2\widehat{P}_X(0) - 1\right) \bmod 2\pi$. For schema matching, the QFT output distributions are compared with cosine similarity. These derivations are also saved in the generated score-derivation CSV file.

Table 3: Encoding construction on the six-triple running example. Times are construction times in the Qiskit/Python prototype and should not be interpreted as hardware execution times.

Encoding	Variant	Index mode	Dimension	Qubits	Time (ms)
Amplitude	Compact baseline	compact	8	3	0.25
Amplitude	Paper-aligned support	paper	256	8	0.17
Basis	Index support	paper	256	8	0.37
Combined	Amplitude+phase	paper	256	8	0.11
Phase	Predicate phase	paper	256	8	4.35
Phase	Presence phase	paper	256	8	4.14

Table 4: Task-level validation on the running example. All shot-based rows use 10,000 shots and three repetitions.

KG task	Encoding	Method	Main measured result	Time
Search	Basis	Grover lookup	success probability 0.947	84.85 ms
Entity matching	Amplitude	Swap test	similarity 0.674	92.42 ms
Keyword search	Amplitude	Swap test	top score 0.492	235.84 ms
Link prediction	Amplitude	Distance estimate	distance 0.028	91.83 ms
Multi-hop reasoning	Phase	Phase kickback	phase error 0.001	926.46 ms
Schema matching	Phase	QFT	Fourier similarity 1.000	610.75 ms

5.1 Encoding Construction on the Running Example

The first experiment checks whether each encoding can be constructed over the same six-triple RDF graph. The paper-aligned index mapping used for the running example maps the six triples to the indices

$$T = \{1, 17, 27, 37, 88, 124\}.$$

Hence, the graph-level representation must contain at least 125 basis states, and our implementation embeds it in a $2^8 = 256$-dimensional Hilbert space. For comparison, Table 3 also includes a compact amplitude baseline over the contiguous positions $0, \ldots, 5$.

The phase encodings are slower in this implementation because the prototype decorates the full 256-dimensional support with phase information. This is an implementation-level observation, not a general statement that phase encoding is always more expensive. If the phase function has an exploitable structure, it can be implemented as a structured oracle rather than as value-by-value assignment. The combined amplitude–phase state was also constructed successfully; its coefficients have the form $\alpha_k e^{i\theta_k}$, with amplitudes encoding triple presence and phases encoding predicate-dependent information, and the generated state had norm one.

5.2 Task-Level Validation

Table 4 reports the six task-level validations. These rows should not be compared as equivalent benchmarks; they validate different task primitives. The search task uses a compact three-qubit index space for the six triples, while the graph encoding in Table 3 uses eight qubits because it follows the paper-aligned RDF index mapping. The Grover search used two Grover iterations, consistent with the usual $\lfloor (\pi/4)\sqrt{N/M} \rfloor$ rule for an eight-state space with one marked item.

Table 5: Circuit statistics for the task-level validations. Unlike the previous version of the paper, the phase-based multi-hop and schema-matching rows are now measured with shots rather than reported as zero-shot quantities.

Task	Qubits	Depth	Gates	Transp. depth	Transp. gates	Shots	Reps.
Search	3	21	43	22	47	10000	3
Entity matching	5	5	7	5	7	10000	3
Keyword search	7	6	8	6	8	10000	3
Link prediction	3	4	6	4	6	10000	3
Multi-hop reasoning	2	3	4	4	7	10000	3
Schema matching	4	2	4	6	15	10000	3

The lookup validation returned the target triple with an average success probability 0.947. This supports the use of basis encoding for exact symbolic lookup, but only as an implementation-level toy validation. The entity-matching swap test estimated a squared similarity of 0.674, while the exact classical squared inner product for the same vectors is 0.667. The keyword-search experiment ranked `Aristotle` above `Athens`, with measured scores close to the exact classical values 0.5 and 0.25. The link-prediction experiment validates only a small TransE-style distance-estimation mechanism; it is not an implementation of HHL and not a complete link-prediction system.

For multi-hop reasoning, the tested path is (`ex:Aristotle`, `rdf:type`, `ex:Person`) followed by (`ex:Person`, `rdfs:subClassOf`, `ex:Mortal`). The expected composed phase is $\theta_{\texttt{rdf:type}} + \theta_{\texttt{rdfs:subClassOf}} = 2.35619$, and the shot-based phase estimate had an average error of about 0.001. This validates the phase-composition mechanism for this controlled two-hop example. It should not be read as a full RDF(S) reasoner, because arbitrary rule application and path inference are outside the scope of this experiment.

For schema matching, the QFT experiment compares phase patterns for relation pairs. This validation assumes that candidate relations have already been mapped to phases using an explicit prior, such as exact URI equality or a synonym dictionary. Under that assumption, the measured Fourier-pattern similarity was approximately 1.000. A negative control with no semantic prior produced a low measured similarity, about 0.026–0.029. This is important because the method does not discover synonyms from nothing; the encoding requires a source of relation similarity before QFT-based phase-pattern comparison can be meaningful.

5.3 Measurement Histograms and Score Derivations

Representative histograms are omitted for space but are included in the artifact repository together with the raw count files used to compute the reported values in Tables 4 and 5. These artifacts make the validation reproducible: the reported scores are not manually assigned, but are derived from the measured counts using the formulas described above.

5.4 Synthetic RDF Graphs

The six-triple graph is useful for checking correctness because its triples, indices, and expected outcomes can be inspected manually. To avoid limiting the evidence to this toy graph, we also generated deterministic synthetic RDF graphs with 6, 25, 50, 100, 250, 500, and 1000 triples. Table 6 summarizes the average qubit requirements and total software runtime across three repetitions. The combined amplitude–phase encoding is not included in these scaling rows because the current scaling runner supports basis, amplitude, and phase encodings only; the combined encoding was validated separately on the running example.

Table 6: Synthetic RDF observations. Each cell reports qubits / total software runtime in ms, averaged over three repetitions. Timings include Python/Qiskit construction and simulator overhead and are not hardware execution times.

Triples	Basis	Amplitude	Phase
6	7 / 35.3	3 / 10.0	3 / 17.2
25	11 / 353.7	5 / 15.5	5 / 26.9
50	13 / 1444.3	6 / 22.8	6 / 40.0
100	15 / 194.7	7 / 37.8	7 / 71.2
250	17 / 809.4	8 / 76.0	8 / 137.1
500	19 / 4021.6	9 / 121.9	9 / 323.9
1000	22 / 38590.9	10 / 264.5	10 / 1177.8

Table 7: Real RDF observations. Each encoding cell reports qubits/total software runtime in ms.

Dataset	Triples	Entities	Pred.	Basis	Amplitude / Phase
Aristotle	8	10	3	10 / 204.8	3 / 10.9 / 3 / 17.9
`exampleV3`	129	71	16	18 / 1634.7	8 / 36.5 / 8 / 91.4
`productsSmall`	129	71	16	18 / 1716.7	8 / 45.6 / 8 / 95.0
`DecodedOntologies_V2`	500	445	5	21 / 18219.5	9 / 246.4 / 9 / 402.8

The synthetic results support the resource-level comparison from Section 4.6. Basis encoding uses more qubits because it preserves the explicit subject–predicate–object structure. Amplitude and phase encodings use a compact index space and therefore require $\lceil \log_2 N \rceil$ qubits for N triple positions. These observations do not demonstrate scalability on quantum hardware. They show only how the current simulator-based prototype behaves when the number of triples is increased beyond the six-triple running example.

5.5 Real RDF Inputs

We also ran the encoding pipeline on four RDF inputs: the running Aristotle graph, `exampleV3`, `productsSmall`, and a 500-triple deterministic slice of `DecodedOntologies_V2`. Table 7 reports the same qubit/runtime summary for the real RDF inputs.

These real-RDF runs show the same resource pattern as the synthetic experiments. In the 500-triple `DecodedOntologies_V2` slice, basis encoding requires 21 qubits, while amplitude and phase encodings require 9 qubits. This illustrates the main trade-off: basis encoding keeps symbolic identity explicit, whereas amplitude and phase encodings provide more compact index-based representations but require task-specific decoding and, in the amplitude case, nontrivial state preparation.

5.6 Classical Sanity Baselines and Limitations

In these validations, a measured value is considered acceptable when it preserves the expected task decision relative to the exact classical computation: the target triple must be the most probable

search result, the intended entity must remain top-ranked in keyword search, and the estimated similarity or distance must remain close enough to the exact value to preserve the same match/non-match or true/false interpretation. Thus, the purpose of the thresholds is decision consistency, not proving numerical superiority over the classical computation. To make the validation more transparent, we also computed simple deterministic classical baselines for the same toy inputs. Dictionary lookup returned the target triple in 2.2×10^{-5} seconds. Exact NumPy computation produced the entity-matching squared similarity 0.667, keyword-search scores 0.5 and 0.25, and the TransE-style distance 0.028. These baselines are sanity checks only; they are not graph-database, ANN, vector-search, or compressed-index benchmarks. They are included to show how the measured quantum quantities relate to exact classical quantities on the same inputs.

The experiments, therefore, have clear limitations. They are simulator-based and small. They do not evaluate noise, hardware execution, QRAM, or large-scale KG management systems. The amplitude-encoding results must be interpreted with the state-preparation bottleneck in mind: the logarithmic qubit count is useful only if the state can be prepared and accessed efficiently. The multi-hop and schema-matching tasks validate phase accumulation and phase-pattern comparison, not complete reasoning or schema matching. Finally, the synthetic and real RDF experiments measure software-level behavior of this prototype, not asymptotic quantum advantage. Nevertheless, the experiments strengthen the empirical support for the limited claim made in this paper: different KG tasks preserve and access different kinds of information, and this makes the choice of quantum encoding central to the design of KG-to-quantum workflows.

6 Concluding Remarks

This paper compared three quantum encoding families for Knowledge Graphs: basis, amplitude, and phase encoding. The comparison shows that the encoding choice is a central design decision in a KG-to-quantum workflow, because it determines how symbolic RDF triples are represented, how many qubits are required, how state preparation is performed, and how measurement outcomes must be interpreted.

The main conclusion is that no encoding is universally preferable. Basis encoding is the most direct option when symbolic identity and exact decoding are required, as in storage and lookup. Amplitude encoding is more appropriate when KG objects are represented through numerical vectors and the task involves similarity, matching, ranking, or distance estimation. Phase encoding is useful for marking, phase accumulation, and interference-based relational patterns, but only for limited compositional forms of reasoning rather than full RDF/S reasoning. Combined amplitude–phase encoding can represent both triple presence or importance and predicate-dependent phase information in one complex-valued state.

We provided simulator-based validation using Qiskit Aer over the running example, synthetic RDF graphs, and real RDF samples. These experiments show that the proposed encodings and task primitives can be implemented in a common RDF-to-quantum pipeline and that the measured quantities are consistent with the corresponding exact classical computations on the same inputs. The results should be interpreted as implementation-level validation, not as evidence of quantum advantage, hardware scalability, or superiority over optimized classical KG systems.

Future work includes studying each KG task in greater depth, adding stronger classical baselines such as graph indexes and vector-search systems, exploring structured state-preparation methods, evaluating hardware-aware transpilation and noisy execution, and extending the combined amplitude–phase encoding to larger synthetic and real RDF graphs.

The code, input RDF example, generated plots, and runtime logs for the experiments are publicly available[2].

Acknowledgements

This work was supported by DEDALUS - Data Management using Quantum Computing project (TA 5180519) funded by SUB1.1 Clusters of Research Excellence (CREs), Greece.

References

[1] Aharonov, D., Kitaev, A., Nisan, N.: Quantum bit commitment from poly-time verification, or, proof of $bqp \subseteq ip$. In: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing. pp. 20–29 (1997)

[2] Barlaug, N., Gulla, J.A.: Neural networks for entity matching: A survey. ACM Transactions on Knowledge Discovery from Data (TKDD) **15**(3), 1–37 (2021)

[3] Becker, A., Sherif, M.A., Ngonga Ngomo, A.C.: Blink: Blank node matching using embeddings. In: International Semantic Web Conference. pp. 218–236. Springer (2024)

[4] Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: DBpedia-A crystallization point for the web of data. Journal of web semantics **7**(3), 154–165 (2009)

[5] Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Advances in Neural Information Processing Systems 26 (NIPS 2013). vol. 26, pp. 2787–2795 (2013)

[6] Chatzakis, M., Mountantonakis, M., Tzitzikas, Y.: Rdfsim: similarity-based browsing over dbpedia using embeddings. Information **12**(11), 440 (2021)

[7] Coppersmith, D.: An approximate fourier transform useful in quantum factoring. arXiv (2002). https://doi.org/10.48550/arxiv.quant-ph/0201067

[8] Elbassuoni, S., Blanco, R.: Keyword search over rdf graphs. In: Proceedings of the 20th ACM international conference on Information and knowledge management. pp. 237–242 (2011)

[9] Fankhauser, T., Solèr, M.E., Füchslin, R.M., Stockinger, K.: Multiple query optimization using a gate-based quantum computer. IEEE Access **11**, 114031–114043 (2023)

[10] Fanourakis, N., Efthymiou, V., Kotzinos, D., Christophides, V.: Knowledge graph embedding methods for entity alignment: An experimental review. arXiv preprint arXiv:2203.09280 (2022)

[11] Gerlach, L., Köppl, T., Zander, R., Schweikardt, N., Scherzinger, S.: Quantum computing for query containment of conjunctive queries. arXiv preprint arXiv:2602.21803 (2026)

[12] Giovannetti, V., Lloyd, S., Maccone, L.: Quantum random access memory. Physical review letters **100**(16), 160501 (2008)

[13] Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. pp. 212–219 (1996)

[14] Hai, R., Hung, S.H., Coopmans, T., Littau, T., Geerts, F.: Quantum data management in the nisq era. Proceedings of the VLDB Endowment **18**(6), 1720–1729 (2025)

[15] Hai, R., Hung, S.H., Feld, S.: Quantum data management: From theory to opportunities. arXiv preprint arXiv:2403.02856 (2024)

[16] Harrow, A.W., Hassidim, A., Lloyd, S.: Quantum algorithm for linear systems of equations. Physical Review Letters **103**(15), 150502 (2009)

[17] Kadilierakis, G., Fafalios, P., Papadakos, P., Tzitzikas, Y.: Keyword search over rdf using document-centric information retrieval systems. In: European Semantic Web Conference. pp. 121–137. Springer (2020)

[18] Lall, D., Agarwal, A., Zhang, W., Lindoy, L., Lindström, T., Webster, S., Hall, S., Chancellor, N., Wallden, P., Garcia-Patron, R., et al.: A review and collection of metrics and benchmarks for quantum computers: definitions, methodologies and software. arXiv preprint arXiv:2502.06717 (2025)

[2] https://github.com/ChristosPolimatidis/Qiskit-KG-Encodings

[19] Lantzaki, C., Papadakos, P., Analyti, A., Tzitzikas, Y.: Radius-aware approximate blank node matching using signatures. Knowledge and Information Systems **50**(2), 505–542 (2017)

[20] Li, T., Yuan, G., Yao, C., Shi, M., Wang, Z., Qian, L., Lu, J.: Quantum storage design for tables in RDBMS. Proceedings of the VLDB Endowment. ISSN **2150**, 8097 (2024)

[21] Limnaios, E., Sideri, S., Kondylakis, H.: Quantum-pg-hive: Schema discovery for property graphs using quantum computing. Workshops of the EDBT/ICDT 2026 (2026)

[22] Limnaios, E., Stergiopoulos, M., Stamatiou, G.T., Efthymiou, V., Loupas, D., Tsourounis, D., Blekos, K., Tsikas, A., Plexousakis, D., Magoutis, K., et al.: Dedalus: A quantum-enhanced end-to-end framework for cost-aware join order optimization with search space pruning. EDBT 2026 (2026)

[23] Liu, H., You, X., Wong, R.C.: First tree-like quantum data structure: Quantum B+ tree. CoRR **abs/2405.20416** (2024). https://doi.org/10.48550/ARXIV.2405.20416, https://doi.org/10.48550/arXiv.2405.20416

[24] Ma, Y., Tresp, V.: Quantum machine learning algorithm for knowledge graphs. ACM Transactions on Quantum Computing **2**(3), 1–28 (2021)

[25] Marketakis, Y., Tzitzikas, Y.: Beyond prompt-to-rdf: A vision for scalable and explainable graph transformations via llm-assisted schema mappings. EDBT/ICDT 2026 Workshops (2026)

[26] Mountantonakis, M., Tzitzikas, Y.: Large scale semantic integration of linked data: A survey. ACM Computing Surveys (CSUR) **52** (2019)

[27] Murillo, J.M., Garcia-Alonso, J., Moguel, E., Barzen, J., Leymann, F., Ali, S., Yue, T., Arcaini, P., Pérez-Castillo, R., García-Rodríguez de Guzmán, I., et al.: Quantum software engineering: Roadmap and challenges ahead. ACM Transactions on Software Engineering and Methodology **34**(5), 1–48 (2025)

[28] Nayak, N., Rehfeld, J., Winker, T., Warnke, B., Çalikyilmaz, U., Groppe, S.: Constructing optimal bushy join trees by solving QUBO problems on quantum hardware and simulators. In: Proceedings of the international workshop on big data in emergent distributed environments. pp. 1–7 (2023)

[29] Nikas, C., Kadilierakis, G., Fafalios, P., Tzitzikas, Y.: Keyword search over rdf: Is a single perspective enough? Big Data and Cognitive Computing **4**(3), 22 (2020)

[30] Rath, M., Date, H.: Quantum data encoding: A comparative analysis of classical-to-quantum mapping techniques and their impact on machine learning accuracy. EPJ Quantum Technology **11**(1), 72 (2024)

[31] Shi, R.H.: Quantumbloomfilteranditsapplications. IEEE Transactions on Quantum Engineering **2**, 1–11 (2021). https://doi.org/10.1109/TQE.2021.3054623

[32] Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review **41**(2), 303–332 (1999)

[33] Stiliadou, L., Barzen, J., Beisel, M., Leymann, F., Weder, B.: Patterns for quantum machine learning. In: Proceedings of the 17th International Conference on Pervasive Patterns and Applications (PATTERNS). pp. 7–14 (2025)

[34] Theodorakis, G., Touloupakis, M., Tzitzikas, Y.: On encoding big knowledge graphs as quantum states and on running grover's algorithm. In: Proceedings of KM*QC 2025, satellite workshops of IJCKG 2025. Springer (2026)

[35] Tzitzikas, Y., Kondylakis, H.: Knowledge graphs and quantum computing: First blood. In: Proceedings of IJCKG 2025. Springer (2026)

[36] Tzitzikas, Y., Lantzaki, C., Zeginis, D.: Blank node matching and rdf/s comparison functions. In: International Semantic Web Conference. pp. 591–607. Springer (2012)

[37] Vrandečić, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Communications of the ACM **57**(10), 78–85 (2014)

[38] Wang, M., Qiu, L., Wang, X.: A survey on knowledge graph embeddings for link prediction. Symmetry **13**(3), 485 (2021)

[39] Zhao, H., Zlokapa, A., Neven, H., Babbush, R., Preskill, J., McClean, J.R., Huang, H.Y.: Exponential quantum advantage in processing massive classical data. arXiv preprint arXiv:2604.07639 (2026)