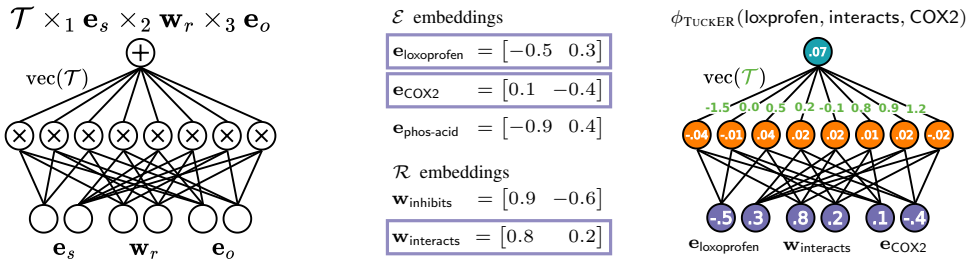


(a) (b)

Figure A.1: **Evaluation of circuit representations of score functions as in neural networks.** Feed-forward evaluation of the TUCKER score function as a circuit over 2-dimensional embeddings and parameterised by the core tensor $\mathcal{T}$ (see proof of Prop. 1 below) (a). Given a triple (loxoprofen, interacts, COX2), the input units (Def. 1) map subject, predicate and object to their embedding entries (in violet boxes). Then, the circuit is evaluated similarly to neural networks: the products (in orange) are evaluated before the weighted sum (in blue), which is parameterised by the core tensor values (in green) (b). The output of the circuit is the score of the input triple.

A Proofs

A.1 KGE Models as Circuits

Proposition 1 (Score functions of KGE models as circuits)**.** The computational graphs of the score functions ϕ of CP, RESCAL, TUCKER and COMPLEX are smooth and decomposable circuits over $\mathbf{X} = \{S, R, O\}$, whose evaluation cost is $\text{cost}(\phi) \in \Theta(|\phi|)$, where $|\phi|$ denotes the number of edges in the circuit, also called its size. For example, the size of the circuit for CP is $|\phi_{\text{CP}}| \in \mathcal{O}(d)$.

Proof. We present the proof by construction for TUCKER [3], as CP [40], RESCAL [47] and COMPLEX [65] define score functions that are a specialisation of it [3] (see below). Given a triple $(s, r, o) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, the TUCKER score function computes

$$\phi_{\text{TUCKER}}(s, r, o) = \mathcal{T} \times_1 \mathbf{e}_s \times_2 \mathbf{w}_r \times_3 \mathbf{e}_o = \sum_{i=1}^{d_e} \sum_{j=1}^{d_r} \sum_{k=1}^{d_e} \tau_{ijk} e_{si} w_{rj} e_{ok} \tag{4}$$

where $\mathcal{T} \in \mathbb{R}^{d_e \times d_r \times d_e}$ denotes the core tensor, $\times_n$ denotes the tensor product along the n-th mode, and d_e, d_r denote respectively the embedding sizes of entities and predicates (which might not be equal). To see how this parametrization generalises that of CP, RESCAL and COMPLEX, consider for example the score function of CP on d-dimensional embeddings. It can be obtained by (i) setting the core tensor $\mathcal{T}$ to be a *diagonal tensor* having ones on the superdiagonal, and (ii) having two distinct embedding instances for each entity that are used depending on their role (either subject or object) in a triple. The embeddings $\mathbf{e}_s, \mathbf{e}_o \in \mathbb{R}^{d_e}$ (resp. $\mathbf{w}_r \in \mathbb{R}^{d_r}$) are rows of the matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times d_e}$ (resp. $\mathbf{W} \in \mathbb{R}^{|\mathcal{R}| \times d_r}$), which associates an embedding to each entity (resp. predicate).

Constructing the circuit. For the construction of the equivalent circuit it suffices to (i) create an input unit for each i-th entry of an embedding for subjects, predicates and objects, as to implement a look-up table that computes the corresponding embedding value for an entity or predicate, and (ii) transform the tensor multiplications into corresponding sum and product units. We start by introducing the input units l_i^S, l_j^R and l_k^O for $1 \leq i, k \leq d_e$ and $1 \leq j \leq d_r$ as parametric mappers over variables S, R and O, respectively. The input units l_i^S and l_k^O (resp. l_j^R) are parameterised by the matrix $\mathbf{E}$ (resp. $\mathbf{W}$) such that $l_i^S(s; \mathbf{E}) = e_{si}$ and $l_i^O(o; \mathbf{E}) = e_{oi}$ for some $s, o \in \mathcal{E}$ (resp. $l_j^R(r; \mathbf{W}) = w_{ri}$ for some $r \in \mathcal{R}$). To encode the tensor products in Eq. (4) we introduce $d_e^2 \cdot d_r$ product units ϕ_{ijk}, each of them computing the product of a combination of the outputs of input units.

$$\phi_{ijk}(s, r, o) = l_i^S(s) \cdot l_j^R(r) \cdot l_k^O(o)$$

Finally, a sum unit ϕ_{out} parameterised by the core tensor $\mathcal{T} \in \mathbb{R}^{d_e \times d_r \times d_r}$ computes a weighted summation of the outputs given by the product units, i.e.,

$$\phi_{\text{out}}(s, r, o) = \sum_{\substack{(i,j,k) \in \\ [d_e] \times [d_r] \times [d_e]}} \tau_{ijk} \cdot \phi_{ijk}(s, r, o)$$

Table A.1: **Score functions as compact circuits.** Asymptotic size of circuits encoding the score functions of CP, RESCAL, COMPLEX and TUCKER, with respect to the embedding size. For TUCKER, d_e and d_r denote the embedding sizes for entities and predicates, respectively.

KGE Model	Circuit Size	KGE Model	Circuit Size
CP	$\mathcal{O}(d)$	RESCAL	$\mathcal{O}(d^2)$
COMPLEX	$\mathcal{O}(d)$	TUCKER	$\mathcal{O}(d_e^2 \cdot d_r)$

where $[d]$ denotes the set $\{1, \dots, d\}$ and τ_{ijk} is the (i, j, k)-th entry of $\mathcal{T}$. We now observe that the constructed circuit ϕ_{out} encodes the TUCKER score function (Eq. (4)), as $\phi_{\text{TUCKER}}(s, r, o) = \phi_{\text{out}}(s, r, o)$ for any input triple $(s, r, o) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$.

Circuit evaluation and properties. Evaluating the score function of TUCKER corresponds to performing a feed-forward pass of its circuit representation, where each computational unit is evaluated once, as we illustrate in Fig. A.1. As such, the cost of evaluating the score function is proportional to the size of its circuit representation, i.e., $\text{cost}(\phi) \in \Theta(|\phi|)$ where $|\phi| \in \mathcal{O}(d_e^2 \cdot d_r)$ is the number of edges. In Table A.1 we show how the sizes of the circuit representation of the other score functions increases with respect to the embedding size. Finally, since each product unit ϕ_{ijk} is defined on the same scope (see Def. 1) $\{S, R, O\}$ and fully decompose it into its inputs (i.e., into $\{S\}, \{R\}, \{O\}$), and the inputs of the sum unit ϕ_{out} are all defined over the same scope, we have that the circuit satisfies smoothness and decomposability (Def. 2). □

Furthermore, in Lem. A.1 we show that the circuit representations of CP, RESCAL, TUCKER and COMPLEX and the proposed GeKCs (Section 4) satisfy a structural property known as *omni-compatibility* (see Def. B.2). In a nutshell, the score functions of the aforementioned KGE models and GeKCs are circuits that fully decompose their scope $\{S, R, O\}$ into $(\{S\}, \{R\}, \{O\})$. The satisfaction of this property will be useful to prove both Thm. 1 and Thm. 2 later in this appendix.

Lemma A.1 (KGE models and derived GeKCs are omni-compatible)**.** The circuit representation of the score functions of CP, RESCAL, TUCKER, COMPLEX and their GeKCs counterparts obtained by non-negative restriction (Section 4.1) or squaring (Section 4.2) are omni-compatible (see Def. B.2).

Proof. To begin, we note that to comply with Def. B.2 every omni-compatible circuit shall contain product units that fully factorise over their scope. In other words, for every product unit n with scope $\mathsf{sc}(n) = \mathbf{X}$, its scope shall decompose as $(\{X_1\}, \{X_2\}, \dots, \{X_{|\mathsf{sc}(n)|}\})$. To see why, consider a circuit ϕ with a product unit n whose scope is decomposed as $\mathsf{sc}(n) = (\mathbf{X}, \mathbf{Y})$. It is easy to construct another circuit ϕ' that is not compatible with ϕ by having a product unit m with scope $\mathsf{sc}(m) = \mathsf{sc}(n)$ decomposed in a way that it cannot be rearranged by introducing additional decomposable product units (Def. 2), e.g., $\mathsf{sc}(m) = (\mathbf{Z}, \mathbf{W})$ with $\mathbf{Z} \cap \mathbf{X} \neq \varnothing$ and $\mathbf{W} \cap \mathbf{X} \neq \varnothing$. As such, every omni-compatible circuit over $\mathbf{X}$ must be representable in the form $\sum_{i=1}^{N} \theta_i \prod_{k=1}^{|\mathbf{X}|} l_{ik}(X_k)$ without any increase in its size.

Now, it is easy to verify that the circuit representations of CP, RESCAL, TUCKER and COMPLEX follow the above form, with a different number N of product units feeding the single sum unit, but each one decomposing its scope $\{S, R, O\}$ into $(\{S\}, \{R\}, \{O\})$ (see Fig. 2). From this it follows that CP$^+$, RESCAL$^+$, TUCKER$^+$ and COMPLEX$^+$ are omni-compatible as well, as they share the same structure of their energy-based counterpart, while just enforcing non-negative activations via reparametrisation (see Section 4.1).

Finally, we note that CP^2, RESCAL^2, TUCKER^2 and COMPLEX^2 are still omni-compatible because the square operation yields the following fully-factorised representation: $(\sum_{i=1}^{N} \theta_i \prod_{k=1}^{|\mathbf{X}|} l_{ik}(X_k))^2 = \sum_{i=1}^{N} \sum_{j=1}^{N} \theta_i \theta_j \prod_{k=1}^{|\mathbf{X}|} l_{ik}(X_k) \prod_{k=1}^{|\mathbf{X}|} l_{jk}(X_k)$ which can be easily rewritten as $\sum_{h=1}^{N^2} \omega_h \prod_{k=1}^{|\mathbf{X}|} l_{hk}(X_k)$ where now h ranges over the Cartesian product of $i \in [N]$ and $j \in [N]$, ω_h is the product of $\theta_i \theta_j$ and l_{hk} is a new input unit that encodes $l_{ik}(X_k) l_{jk}(X_k)$ for a certain variable index k. □

A.2 Efficient Summations over Circuits

Proposition 2 (Efficient Summations)**.** Let ϕ be a smooth and decomposable circuit over $\mathbf{X} = \{S, R, O\}$ that encodes the score function of a KGE model. The sum $\sum_{s\in\mathcal{E}}\sum_{r\in\mathcal{R}}\sum_{o\in\mathcal{E}}\phi(s,r,o)$ or any other summation over subjects, predicates or objects can be computed in time $\mathcal{O}((|\mathcal{E}|+|\mathcal{R}|)\cdot|\phi|)$.

Proof. A proof for the computation of marginal probabilities in smooth and decomposable *probabilistic circuits* (PCs) defined over discrete variables in linear time with respect to their size can be found in [13]. This proof also applies for computing summations in smooth and decomposable circuits that do not necessarily corresponds to marginal probabilities [76]. The satisfaction of smoothness and decomposability (Def. 2) in a circuit ϕ permits to push outer summations inside the computational graph until input units are reached, where summations are actually performed independently and on smaller sets of variables (i.e., $\{S\}$, $\{R\}$, $\{O\}$ in our case), and then to evaluate the circuit only once.

Here we take into account the computational cost of summing over each input unit (see proof of Prop. 1), which is $\mathcal{O}(|\mathcal{E}|)$ (resp. $\mathcal{O}(|\mathcal{R}|)$) for those defined on variables S, O (resp. R). Since the size of the circuit $|\phi|$ must be at least the number of input units, we retrieve that the overall complexity for computing summations as stated in the proposition is $\mathcal{O}((|\mathcal{E}|+|\mathcal{R}|)\cdot|\phi|)$.

As an example, consider the CP score function computing $\phi_{\text{CP}}(s,r,o) = \langle \mathbf{e}_s, \mathbf{w}_r, \mathbf{e}_o\rangle$ for some triple (s,r,o) and embeddings $\mathbf{e}_s, \mathbf{w}_r, \mathbf{e}_o \in \mathbb{R}^d$. We can compute $\sum_{s\in\mathcal{E}}\sum_{r\in\mathcal{R}}\sum_{o\in\mathcal{E}}\phi_{\text{CP}}(s,r,o)$ by pushing the outer summations inside the trilinear product, i.e., by computing it as $\langle\sum_{s\in\mathcal{E}}\mathbf{e}_s, \sum_{r\in\mathcal{R}}\mathbf{w}_r, \sum_{o\in\mathcal{E}}\mathbf{e}_o\rangle$, which requires time $\mathcal{O}((|\mathcal{E}|+|\mathcal{R}|)\cdot d)$. □

A.3 Efficient Summations over Squared Circuits

Theorem 1 (Efficient summations of squared GeKCs)**.** Performing summations as stated in Prop. 2 on CP^2, RESCAL^2, TUCKER^2 and COMPLEX^2 can be done in time $\mathcal{O}((|\mathcal{E}|+|\mathcal{R}|)\cdot|\phi|^2)$.

Proof. In Lem. A.1 we showed that the circuit representations ϕ of CP, RESCAL, TUCKER and COMPLEX are omni-compatible (see Def. B.2). As a consequence, ϕ is compatible (see Def. B.1) with itself. Therefore, Prop. B.1 ensures that we can construct the product circuit $\phi\cdot\phi$ (i.e., ϕ^2) as a smooth and decomposable circuit having size $\mathcal{O}(|\phi|^2)$ in time $\mathcal{O}(|\phi|^2)$. Since ϕ^2 is still smooth and decomposable, Prop. 2 guarantees that we can perform summations in time $\mathcal{O}((|\mathcal{E}|+|\mathcal{R}|)\cdot|\phi|^2)$. □

A.4 Circuits encoding Domain Constraints

In Def. A.1 we introduce the concepts of *support* and *determinism*, whose definition is useful to describe *constraint circuits* in Def. A.2.

Definition A.1 (Support and Determinism [13, 76])**.** In a circuit the *support* of a computational unit n over variables $\mathbf{X}$ computing $\phi_n(\mathbf{X})$ is defined as the set of value assignments to variables in $\mathbf{X}$ such that the output of n is non-zero, i.e., $\mathsf{supp}(n) = \{\mathbf{x}\in\mathsf{val}(\mathbf{X}) \mid \phi_n(\mathbf{X})\neq 0\}$. A sum unit n is *deterministic* if its inputs have disjoint *supports*, i.e., $\forall i,j\in\mathsf{in}(n), i\neq j\colon \mathsf{supp}(i)\cap\mathsf{supp}(j)=\varnothing$.

Definition A.2 (Constraint Circuit [1])**.** Given a propositional logic formula K, a *constraint circuit* c_K is a smooth and decomposable PC over variables $\mathbf{X}$ with *deterministic* sum units (Def. A.1) and indicator functions as input units, such that $c_K(\mathbf{x}) = \mathbb{1}\{\mathbf{x}\models K\}$ for any $\mathbf{x}\in\mathsf{val}(\mathbf{X})$.

In general, we can compile any propositional logic formula into a constraint circuit (Def. A.2) by leveraging knowledge compilation techniques [19, 18, 52]. For domain constraints (Def. 4) this compilation process is straightforward, as we detail in the following proposition and proof.

Proposition A.1 (Circuit encoding domain constraints)**.** Let $K = K_{r_1}\vee\ldots\vee K_{r_m}$ be a disjunction of domain constraints defined over a set of predicates $\mathcal{R} = \{r_1,\ldots,r_m\}$ and a set of entities $\mathcal{E}$ (Def. 4). We can compile K into a constraint circuit c_K (Def. A.2) defined over variables $\mathbf{X} = \{S,R,O\}$ having size $\mathcal{O}(|\mathcal{E}|\cdot|\mathcal{R}|)$ in the worst case and $\mathcal{O}(|\mathcal{E}|+|\mathcal{R}|)$ in the best case.

Proof. Let $K = K_{r_1}\vee\ldots\vee K_{r_m}$ be a disjunction of domain constraints (Def. 4) where

$$K_r \equiv S\in\kappa_S(r)\wedge R = r\wedge O\in\kappa_O(r) \equiv (\vee_{u\in\kappa_S(r)}S=u)\wedge R=r\wedge(\vee_{v\in\kappa_O(r)}O=v).$$

Note that the disjunctions in K are deterministic, i.e., only one of their argument can be true at the same time. This enables us to construct the constraint circuit c_K such that $c_K(s, r, o) = \mathbb{1}\{(s, r, o) \models K\}$ for any triple by simply replacing conjunctions and disjunctions with product and sum units, respectively. Note that c_K is indeed smooth and decomposable (Def. 2), as the inputs of the sum units are product units having scope $\{S, R, O\}$ that are fully factorised into $(\{S\}, \{R\}, \{O\})$. Moreover, K is a disjunction of $|\mathcal{R}|$ conjunctive formulae having $\mathcal{O}(|\mathcal{E}|)$ terms, and therefore $|c_K| = \mathcal{O}(|\mathcal{E}| \cdot |\mathcal{R}|)$ in the worst case. In the best case of every predicate sharing the same subject and object domains $\kappa_S, \kappa_O \subseteq \mathcal{E}$, we can simplify K into a conjunction of three disjunctive expressions, i.e.,

$$K \equiv (\vee_{u \in \kappa_S} S = u) \wedge (\vee_{r \in \mathcal{R}} R = r) \wedge (\vee_{v \in \kappa_O} O = v)$$

that can be easily compiled into a constraint circuit c_K having size $\mathcal{O}(|\mathcal{E}| + |\mathcal{R}|)$, by again noticing that disjunctions are deterministic. In real-world KGs like ogbl-biokg [32] several predicates share the same subject and object domains, and this permits to have much smaller constraint circuits. □

A.5 Efficient Integration of Domain Knowledge in GeKCs

Theorem 2 (Tractable integration of constraints in GeKCs)**.** Let c_K be a constraint circuit encoding a logical constraint K over variables $\{S, R, O\}$. Then exactly computing the partition function Z_K of the product $\phi_{\mathsf{pc}}(s, r, o) \cdot c_K(s, r, o) \propto p_K(s, r, o)$ for any GeKC ϕ_{pc} derived from CP, RESCAL, TUCKER or COMPLEX (Section 4) can be done in time $\mathcal{O}((|\mathcal{E}| + |\mathcal{R}|) \cdot |\phi_{\mathsf{pc}}| \cdot |c_K|)$.

Proof. In Lem. A.1 we showed that the GeKCs ϕ_{pc} derived from CP, RESCAL, TUCKER and COMPLEX via non-negative restriction (Section 4.1) or squaring (Section 4.2) are omni-compatible (see Def. B.2). As a consequence, ϕ_{pc} is always compatible with c_K regardless of the encoded logical constraint K, since constraint circuits are by definition smooth and decomposable (Def. A.2). By applying Prop. B.1, we retrieve that we can construct $\phi_{\mathsf{pc}} \cdot c_K$ as a smooth and decomposable circuit of size $\mathcal{O}(|\phi_{\mathsf{pc}}| \cdot |c_K|)$ and in time $\mathcal{O}(|\phi_{\mathsf{pc}}| \cdot |c_K|)$. As the resulting product circuit is smooth and decomposable, Prop. 2 guarantees that we can compute its partition function $Z_K = \sum_{s \in \mathcal{E}} \sum_{r \in \mathcal{R}} \sum_{o \in \mathcal{E}} (\phi_{\mathsf{pc}}(s, r, o) \cdot c_K(s, r, o))$ in time $\mathcal{O}((|\mathcal{E}| + |\mathcal{R}|) \cdot |\phi_{\mathsf{pc}}| \cdot |c_K|)$. □

B Circuits

B.1 Tractable Product of Circuits

In this section, we provide the formal definition of *compatibility* (Def. B.1) and *omni-compatibility* (Def. B.2), as stated by Vergari et al. [76]. Given two compatible circuits, Prop. B.1 guarantees that we can represent their product as a smooth and decomposable circuit efficiently.

Definition B.1 (Compatibility)**.** Two circuits ϕ, ϕ' over variables $\mathbf{X}$ are *compatible* if (1) they are smooth and decomposable, and (2) any pair of product units $n \in \phi, m \in \phi'$ having the same scope can be rearranged into binary products that are mutually compatible and decompose their scope in the same way, i.e., $(\mathsf{sc}(n) = \mathsf{sc}(m)) \implies (\mathsf{sc}(n_i) = \mathsf{sc}(m_i),\ n_i \text{ and } m_i \text{ are compatible})$ for some rearrangements of the inputs of n (resp. m) into n_1, n_2 (resp. m_1, m_2).

Definition B.2 (Omni-compatibility)**.** A circuit ϕ over variables $\mathbf{X}$ is *omni-compatible* if it is compatible with any smooth and decomposable circuit over $\mathbf{X}$.

Proposition B.1 (Tractable product of circuits)**.** Let ϕ, ϕ' be two compatible (Def. B.1) circuits. We can represent the product circuit $\phi \cdot \phi'$ computing the product of the outputs of ϕ and ϕ' as a smooth and decomposable circuit having size $\mathcal{O}(|\phi| \cdot |\phi'|)$ in time $\mathcal{O}(|\phi| \cdot |\phi'|)$. Moreover, if both ϕ and ϕ' are omni-compatible (Def. B.2), then also the product circuit $\phi \cdot \phi'$ is omni-compatible.

Prop. B.1 allows us to compute the partition function and any other marginal probability in GeKCs obtained via squaring efficiently (see Section 4.2 and Thm. 1). In addition, Prop. B.1 is a crucial theoretical result that allows us to inject logical constraints in GeKCs in a way that enable computing the partition function exactly and efficiently (see Section 5 and Thm. 2).

C From KGE Models to PCs

C.1 Interpreting Non-negative Embedding Values

In Fig. C.1 we interpret the embedding values of GeKCs obtained via non-negative restriction – CP$^+$, RESCAL$^+$, TUCKER$^+$, COMPLEX$^+$– (Section 4.1) as the parameters of unnormalised categorical distributions over entities (elements in $\mathcal{E}$) or predicates (elements in $\mathcal{R}$).

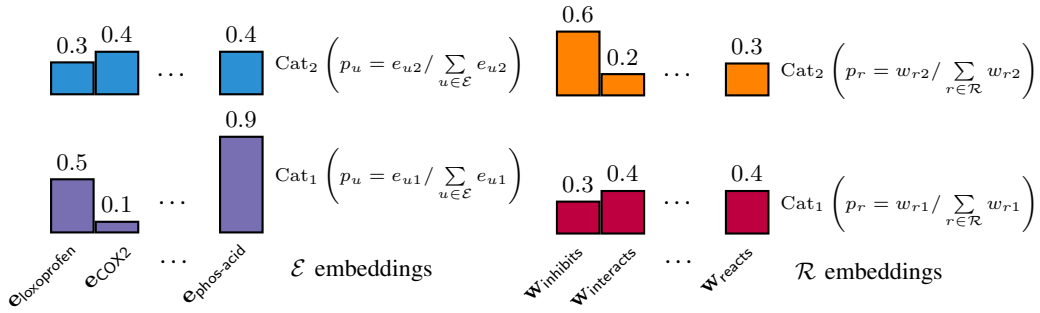


Figure C.1: **Non-negative embeddings parameterise categorical distributions.** 2-dimensional embeddings of GeKCs obtained via non-negative restrictions (Section 4.1) can be seen as the parameters of two categorical distributions over entities (left) or predicates (right) up to renormalisation.

C.2 Realising the Non-negative Restriction of COMPLEX

As anticipated in Section 4.1, for the COMPLEX [65] score function restricting the real and imaginary parts to be non-negative is not sufficient to obtain a PC due to the presence of a subtraction, as showed in the following equation.

$$\begin{aligned}\phi_{\text{COMPLEX}}(s,r,o) = \langle \text{Re}(\mathbf{e}_s), \text{Re}(\mathbf{w}_r), \text{Re}(\mathbf{e}_o)\rangle + \langle \text{Im}(\mathbf{e}_s), \text{Re}(\mathbf{w}_r), \text{Im}(\mathbf{e}_o)\rangle \\ + \langle \text{Re}(\mathbf{e}_s), \text{Im}(\mathbf{w}_r), \text{Im}(\mathbf{e}_o)\rangle - \langle \text{Im}(\mathbf{e}_s), \text{Im}(\mathbf{w}_r), \text{Re}(\mathbf{e}_o)\rangle\end{aligned} \tag{5}$$

Here $\mathbf{e}_s, \mathbf{w}_r, \mathbf{e}_o \in \mathbb{C}^d$ are the embeddings associated to the subject, predicate and object, respectively. Under the restriction of embedding values to be non-negative, we ensure that $\phi_{\text{COMPLEX}}(s,r,o) \geq 0$ for any input triple by enforcing the additional constraint

$$\langle \text{Re}(\mathbf{e}_s), \text{Re}(\mathbf{w}_r), \text{Re}(\mathbf{e}_o)\rangle \geq \langle \text{Im}(\mathbf{e}_s), \text{Im}(\mathbf{w}_r), \text{Re}(\mathbf{e}_o)\rangle, \tag{6}$$

which can be simplified into the two distinct inequalities

$$\forall u \in \mathcal{E} \quad \text{Re}(e_{ui}) \geq \text{Im}(e_{ui}) \qquad \text{and} \qquad \forall r \in \mathcal{R} \quad \text{Re}(w_{ri}) \geq \text{Im}(w_{ri}).$$

In other words, we want the real part of each embedding value to be always greater or equal than the corresponding imaginary part. We implement this constraint in practice by reparametrisation of the imaginary part in function of the real part, i.e.,

$$\forall u \in \mathcal{E} \quad \text{Im}(e_{ui}) = \text{Re}(e_{ui}) \cdot \sigma(\theta_{ui}) \tag{7}$$

$$\forall r \in \mathcal{R} \quad \text{Im}(w_{ri}) = \text{Re}(w_{ri}) \cdot \sigma(\gamma_{ri}) \tag{8}$$

where $\sigma(x) = 1/(1+\exp(-x)) \in [0,1]$ denotes the logistic function and $\theta_{ui}, \gamma_{ri} \in \mathbb{R}$ are additional parameters associated to entities $u \in \mathcal{E}$ and predicates $r \in \mathcal{R}$, respectively. The reparametrisation of the imaginary parts using Eqs. (7) and (8) is a sufficient condition for the satisfaction of the constraint showed in Eq. (6), and also maintains the same number of learnable parameters of COMPLEX.

C.3 Sampling from GeKCs with Non-negative Parameters

Parameters interpretation. Sum units with non-negative parameters in smooth and decomposable PCs can be seen as marginalised discrete latent variables, similarly to the latent variable interpretation in mixture models [55, 53]. That is, the non-negative parameters of a sum unit are the parameters of a (possibly unnormalised) categorical distribution over assignments to a latent variable. For CP$^+$ and RESCAL$^+$ (Section 4.1), the non-negative parameters of the sum unit encode a uniform and

unnormalised categorical distribution, as they are all fixed to 1 (see Fig. 2). By contrast, in TUCKER$^+$ these parameters are the vectorisation of the core tensor $\mathcal{T}$ (see the proof of Prop. 1), and hence they are learned. The input units of CP$^+$, RESCAL$^+$ and TUCKER$^+$ can be interpreted as unnormalised categorical distribution over entities or predicates, as detailed in Appendix C.1.

Sampling from CP$^+$, COMPLEX$^+$, TUCKER$^+$. Thanks to the latent variable interpretation, ancestral sampling in CP$^+$, RESCAL$^+$ and TUCKER$^+$ can be performed by (1) sampling an assignment to the latent variable associated to the single sum unit, i.e., one of its input branches, (2) selecting the corresponding combination of subject-predicate-object input units, and (3) sampling a subject, predicate and object respectively from each of the indexed unnormalized categorical distributions.

C.4 Learning Complexity of GeKCs

In Table C.1 we summarise the complexities of computing the PLL and MLE objectives (Eqs. (1) and (2)) for KGE models and GeKCs. Asymptotically, GeKCs manifest better time and space complexities with respect to the number of entities $|\mathcal{E}|$, batch size $|B|$ and embedding size. This makes GeKCs more efficient than traditional KGE models during training, both in time and memory (see Section 4.3 and Fig. 3).

Table C.1: **Summary of complexities for exactly computing the PLL and MLE objectives.** Time and space complexity of computing $\log p(o \mid s, r)$ and the partition function Z. These complexities are respectively lower bounds of the complexities of computing the PLL and MLE objectives, as we have that $|\mathcal{E}| \gg |\mathcal{R}|$ for large real-world KGs. For CP, RESCAL and COMPLEX and GeKCs derived from them, d denotes the size of both entity and predicate embeddings. For TUCKER and GeKCs derived from it, d_e and d_r denote the embedding sizes for entities and predicates, respectively.

Model	**Complexity of** $\log p(o \mid s, r)$		**Complexity of** Z	
	Time	**Space**	**Time**	**Space**
CP	$\mathcal{O}(\lvert\mathcal{E}\rvert \cdot \lvert B\rvert \cdot d)$	$\mathcal{O}(\lvert\mathcal{E}\rvert \cdot \lvert B\rvert)$	$\mathcal{O}(\lvert\mathcal{E}\rvert^2 \cdot \lvert\mathcal{R}\rvert \cdot d)$	$\mathcal{O}(d)$
RESCAL	$\mathcal{O}(\lvert\mathcal{E}\rvert \cdot \lvert B\rvert \cdot d + \lvert B\rvert \cdot d^2)$	$\mathcal{O}(\lvert\mathcal{E}\rvert \cdot \lvert B\rvert)$	$\mathcal{O}(\lvert\mathcal{E}\rvert^2 \cdot \lvert\mathcal{R}\rvert \cdot d^2)$	$\mathcal{O}(d^2)$
TUCKER	$\mathcal{O}(\lvert\mathcal{E}\rvert \cdot \lvert B\rvert \cdot d_e + \lvert B\rvert \cdot d_e^2 \cdot d_r)$	$\mathcal{O}(\lvert\mathcal{E}\rvert \cdot \lvert B\rvert)$	$\mathcal{O}(\lvert\mathcal{E}\rvert^2 \cdot \lvert\mathcal{R}\rvert \cdot d_e^2 \cdot d_r)$	$\mathcal{O}(d_e^2 \cdot d_r)$
COMPLEX	$\mathcal{O}(\lvert\mathcal{E}\rvert \cdot \lvert B\rvert \cdot d)$	$\mathcal{O}(\lvert\mathcal{E}\rvert \cdot \lvert B\rvert)$	$\mathcal{O}(\lvert\mathcal{E}\rvert^2 \cdot \lvert\mathcal{R}\rvert \cdot d)$	$\mathcal{O}(d)$
CP$^+$	$\mathcal{O}((\lvert\mathcal{E}\rvert + \lvert B\rvert) \cdot d)$	$\mathcal{O}(\lvert B\rvert \cdot d)$	$\mathcal{O}((\lvert\mathcal{E}\rvert + \lvert\mathcal{R}\rvert) \cdot d)$	$\mathcal{O}(d)$
RESCAL$^+$	$\mathcal{O}((\lvert\mathcal{E}\rvert + \lvert B\rvert \cdot d) \cdot d)$	$\mathcal{O}(\lvert B\rvert \cdot d^2)$	$\mathcal{O}((\lvert\mathcal{E}\rvert + \lvert\mathcal{R}\rvert \cdot d) \cdot d)$	$\mathcal{O}(d^2)$
TUCKER$^+$	$\mathcal{O}((\lvert\mathcal{E}\rvert + \lvert B\rvert \cdot d_e \cdot d_r) \cdot d_e)$	$\mathcal{O}(\lvert B\rvert \cdot d_e \cdot d_r)$	$\mathcal{O}(\lvert\mathcal{E}\rvert \cdot d_e + \lvert\mathcal{R}\rvert \cdot d_r + d_e^2 \cdot d_r)$	$\mathcal{O}(d_e^2 \cdot d_r)$
COMPLEX$^+$	$\mathcal{O}((\lvert\mathcal{E}\rvert + \lvert B\rvert) \cdot d)$	$\mathcal{O}(\lvert B\rvert \cdot d)$	$\mathcal{O}((\lvert\mathcal{E}\rvert + \lvert\mathcal{R}\rvert) \cdot d)$	$\mathcal{O}(d)$
CP^2	$\mathcal{O}((\lvert\mathcal{E}\rvert + \lvert B\rvert) \cdot d^2)$	$\mathcal{O}(\lvert B\rvert \cdot d)$	$\mathcal{O}((\lvert\mathcal{E}\rvert + \lvert\mathcal{R}\rvert) \cdot d^2)$	$\mathcal{O}(d^2)$
RESCAL^2	$\mathcal{O}((\lvert\mathcal{E}\rvert + \lvert B\rvert) \cdot d^2)$	$\mathcal{O}(\lvert B\rvert \cdot d^2)$	$\mathcal{O}((\lvert\mathcal{E}\rvert + \lvert\mathcal{R}\rvert \cdot d) \cdot d^2)$	$\mathcal{O}(\lvert\mathcal{R}\rvert \cdot d^2)$
TUCKER^2	$\mathcal{O}((\lvert\mathcal{E}\rvert + \lvert B\rvert \cdot d_r) \cdot d_e^2)$	$\mathcal{O}(\lvert B\rvert \cdot d_e \cdot d_r)$	$\mathcal{O}(\lvert\mathcal{E}\rvert \cdot d_e^2 + \lvert\mathcal{R}\rvert \cdot d_r^2 + d_e^2 \cdot d_r)$	$\mathcal{O}(d_e^2 \cdot d_r)$
COMPLEX^2	$\mathcal{O}((\lvert\mathcal{E}\rvert + \lvert B\rvert) \cdot d^2)$	$\mathcal{O}(\lvert B\rvert \cdot d)$	$\mathcal{O}((\lvert\mathcal{E}\rvert + \lvert\mathcal{R}\rvert) \cdot d^2)$	$\mathcal{O}(d^2)$

C.4.1 Computing the Partition Function

In this section we derive the computational complexity of computing the partition function for GeKCs obtained via squaring (Section 4.2). For a summary of these complexities, see Table C.1.

CP^2 and COMPLEX^2. Here we derive the partition function of CP^2. For COMPLEX^2 the derivation is similar, as the score function of COMPLEX can be written in terms of trilinear products just like CP (see Eq. (5)). The score function ϕ_{CP^2} encoded by CP^2 can be written as

$$\phi_{\text{CP}^2}(s, r, o) = \langle \mathbf{e}_s, \mathbf{w}_r, \mathbf{e}_o \rangle^2 = \sum_{i=1}^{d} \sum_{j=1}^{d} e_{si} e_{sj} w_{ri} w_{rj} e_{oi} e_{oj}$$

where $\mathbf{e}_s, \mathbf{e}_o \in \mathbb{R}^d$ (resp. $\mathbf{w}_r \in \mathbb{R}^d$) are rows of the matrices $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{|\mathcal{E}| \times d}$ (resp. $\mathbf{W} \in \mathbb{R}^{|\mathcal{R}| \times d}$), which associate to each entity (resp. predicate) a vector. By leveraging the *einsum notation* for

brevity, the partition function of ϕ_{CP^2} can be written as

$$Z = \sum_{s\in\mathcal{E}}\sum_{r\in\mathcal{R}}\sum_{o\in\mathcal{E}} \phi_{\text{CP}^2}(s,r,o) = \sum_{i=1}^{d}\sum_{j=1}^{d}\left(\sum_{s\in\mathcal{E}} e_{si}e_{sj}\right)\left(\sum_{r\in\mathcal{R}} w_{ri}w_{rj}\right)\left(\sum_{o\in\mathcal{E}} e_{oi}e_{oj}\right)$$
$$= \mathbf{U}'_{ij}\mathbf{W}'_{ij}\mathbf{V}'_{ij}$$

where $\mathbf{U}' = \mathbf{U}^\top\mathbf{U}$, $\mathbf{W}' = \mathbf{W}^\top\mathbf{W}$ and $\mathbf{V}' = \mathbf{V}^\top\mathbf{V}$ are $d \times d$ matrices. With the simplest algorithm for matrix multiplication, we recover that computing Z requires time $\mathcal{O}(|\mathcal{E}| \cdot d^2 + |\mathcal{R}| \cdot d^2)$ and additional space $\mathcal{O}(d^2)$.

RESCAL^2. The score function ϕ_{RESCAL^2} encoded by RESCAL^2 can be written as

$$\phi_{\text{RESCAL}^2}(s,r,o) = \left(\mathbf{e}_s^\top\mathbf{W}_r\mathbf{e}_o\right)^2 = \sum_{(i,j,k,l)\in[d]^4} e_{si}e_{sk}w_{rij}w_{rkl}e_{oj}e_{ol}$$

where $[d]$ denotes the set $\{1,\dots,d\}$, $\mathbf{e}_s, \mathbf{e}_o \in \mathbb{R}^d$ are rows of the matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}|\times d}$ and $\mathbf{W}_r \in \mathbb{R}^{d\times d}$ are slices along the first mode of the tensor $\mathcal{W} \in \mathbb{R}^{|\mathcal{R}|\times d\times d}$, which consists of stacked matrix embeddings associated to predicates. The partition function of ϕ_{RESCAL^2} can be written as

$$Z = \sum_{s\in\mathcal{E}}\sum_{r\in\mathcal{R}}\sum_{o\in\mathcal{E}} \phi_{\text{RESCAL}^2}(s,r,o) = \sum_{(i,j,k,l)\in[d]^4}\left(\sum_{s\in\mathcal{E}} e_{si}e_{sk}\right)\left(\sum_{r\in\mathcal{R}} w_{rij}w_{rkl}\right)\left(\sum_{o\in\mathcal{E}} e_{oj}e_{ol}\right)$$
$$= \mathbf{E}'_{ik}\mathcal{W}_{rij}\mathcal{W}_{rkl}\mathbf{E}'_{jl} \tag{9}$$

where $\mathbf{E}' = \mathbf{E}^\top\mathbf{E} \in \mathbb{R}^{d\times d}$. The complexity of computing Z depends on the order of tensor contractions in the einsum operation showed in Eq. (9). By optimising the order of tensor contractions (e.g., by using software libraries like opt_einsum), we retrieve that computing Z requires time $\mathcal{O}(|\mathcal{E}|\cdot d^2 + |\mathcal{R}|\cdot d^3)$ and additional space $\mathcal{O}(|\mathcal{R}|\cdot d^2)$. Notice that the time complexity here is slightly lower than the theoretical upper bound given in Thm. 1, which would be $\mathcal{O}(|\mathcal{E}| \cdot d^2 + |\mathcal{R}| \cdot d^4)$.

TuckER^2. Lastly, we present the derivation of the partition function for TuckER^2. The score function ϕ_{TuckER^2} encoded by TuckER^2 can be written as

$$\phi_{\text{TuckER}^2}(s,r,o) = \left(\mathcal{T} \times_1 \mathbf{e}_s \times_2 \mathbf{w}_r \times_3 \mathbf{e}_o\right)^2$$
$$= \sum_{\substack{(i,j,k)\in\\ [d_e]\times[d_r]\times[d_e]}} \sum_{\substack{(l,m,n)\in\\ [d_e]\times[d_r]\times[d_e]}} \tau_{ijk}\tau_{lmn}e_{si}e_{sl}w_{rj}w_{rm}e_{ok}e_{on}$$

where $\mathbf{e}_s, \mathbf{e}_o \in \mathbb{R}^{d_e}$ are rows of the matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}|\times d_e}$, $\mathbf{w}_r$ is a row of the matrix $\mathbf{W} \in \mathbb{R}^{|\mathcal{R}|\times d_r}$, and $\mathcal{T} \in \mathbb{R}^{d_e\times d_r\times d_e}$ denotes the core tensor. The partition function of ϕ_{TuckER^2} can be written as

$$Z = \sum_{s\in\mathcal{E}}\sum_{r\in\mathcal{R}}\sum_{o\in\mathcal{E}} \phi_{\text{TuckER}^2}(s,r,o)$$
$$= \sum_{\substack{(i,j,k)\in\\ [d_e]\times[d_r]\times[d_e]}} \sum_{\substack{(l,m,n)\in\\ [d_e]\times[d_r]\times[d_e]}} \tau_{ijk}\tau_{lmn}\left(\sum_{s\in\mathcal{E}} e_{si}e_{sl}\right)\left(\sum_{r\in\mathcal{R}} w_{rj}w_{rm}\right)\left(\sum_{o\in\mathcal{E}} e_{ok}e_{on}\right)$$
$$= \mathcal{T}_{ijk}\mathcal{T}_{lmn}\mathbf{E}'_{il}\mathbf{W}'_{jm}\mathbf{E}'_{kn} \tag{10}$$

where $\mathbf{E}' = \mathbf{E}^\top\mathbf{E} \in \mathbb{R}^{d_e\times d_e}$ and $\mathbf{W}' = \mathbf{W}^\top\mathbf{W} \in \mathbb{R}^{d_r\times d_r}$. Similarly to RESCAL^2, by optimising the order of tensor contractions in the einsum operation showed in Eq. (10), we retrieve that computing Z requires time $\mathcal{O}(|\mathcal{E}|\cdot d_e^2 + |\mathcal{R}|\cdot d_r^2 + d_e^2\cdot d_r)$ and additional space $\mathcal{O}(d_e^2\cdot d_r)$. Similarly to RESCAL^2, the time complexity is lower than the theoretical upper bound given in Thm. 1, which would be $\mathcal{O}(|\mathcal{E}| \cdot d_e^2 + |\mathcal{R}| \cdot d_r^2 + d_e^4 \cdot d_r^2)$.

C.4.2 Complexity of Computing the PLL Objective

In this section, we show that GeKCs enable to better scale the computation of the PLL objective (Eq. (1)) with respect to energy-based KGE models (see Section 2). We present this concept for CP and GeKCs derived from it (Section 4), as for the other score functions it is similar.

Complexity of the PLL objective on CP. Let $\phi_{\text{CP}}(s,r,o) = \langle \mathbf{e}_s, \mathbf{w}_r, \mathbf{e}_o \rangle = \sum_{i=1}^{d} e_{si} w_{ri} e_{oi}$ be the score function of CP [40], where $\mathbf{e}_s, \mathbf{e}_o \in \mathbb{R}^d$ (resp. $\mathbf{w}_r \in \mathbb{R}^d$) are rows of the matrices $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{|\mathcal{E}| \times d}$ (resp. $\mathbf{W} \in \mathbb{R}^{|\mathcal{R}| \times d}$), which associate to each entity (resp. predicate) a vector. Given a training triple (s,r,o), the computation of the term $\log p(o \mid s,r) = \phi(s,r,o) - \log \sum_{o' \in \mathcal{E}} \exp \phi(s,r,o')$ requires evaluating $\phi_{\text{CP}}(s,r,o')$ for all objects $o' \in \mathcal{E}$. In order to fully exploit GPU parallelism [36], this is usually done with the matrix-vector multiplication $\mathbf{V}(\mathbf{e}_s \odot \mathbf{w}_r) \in \mathbb{R}^{|\mathcal{E}|}$, where $\odot$ denotes the Hadamard product [40, 12]. Therefore, computing $\log p(o \mid s,r)$ for each triple (s,r,o) in a mini-batch $B \subset \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ such that $|\mathcal{E}| \gg |\mathcal{B}|$ requires time $\mathcal{O}(|\mathcal{E}| \cdot |B| \cdot d)$ and space $\mathcal{O}(|\mathcal{E}| \cdot |B|)$. For the other terms of the PLL objective (i.e., $\log p(s \mid r,o)$ and $\log p(r \mid s,o)$) the derivation is similar. Moreover, for real-world large KGs it is reasonable to assume that $|\mathcal{E}| \gg |\mathcal{R}|$ and therefore the cost of computing $\log p(r \mid s,o)$ is negligible.

Complexity of the PLL objective on GeKCs. GeKCs obtained from CP either by non-negative restriction (Section 4.1) or by squaring (Section 4.2) encode $\phi_{\mathsf{pc}}(s,r,o) \propto p(s,r,o)$ for any input triple. As such, the component $\log p(o \mid s,r)$ of the PLL objective can be written as

$$\log p(o \mid s,r) = \log \phi_{\mathsf{pc}}(s,r,o) - \log \sum_{o' \in \mathcal{E}} \phi_{\mathsf{pc}}(s,r,o'). \tag{11}$$

The absence of the exponential function in the summed terms in Eq. (11) allows us to push the outer summation inside the circuit computing $\phi_{\mathsf{pc}}(s,r,o)$, and to sum over the input units relative to objects. For instance, for CP$^+$ we can write

$$\sum_{o' \in \mathcal{E}} \phi_{\text{CP}^+}(s,r,o') = \sum_{o' \in \mathcal{E}} \sum_{i=1}^{d} e_{si} w_{ri} e_{oi} = \sum_{i=1}^{d} e_{si} w_{ri} \left(\sum_{o \in \mathcal{E}} e_{o'i} \right) = (\mathbf{e}_s \odot \mathbf{w}_r)^\top (\mathbf{V}^\top \mathbf{1}_{\mathcal{E}})$$

where $\mathbf{e}_s, \mathbf{w}_r, \mathbf{e}_o \in \mathbb{R}^d_+$, $\mathbf{V} \in \mathbb{R}^{|\mathcal{E}| \times d}_+$ denotes the matrix whose rows are object embeddings, and $\mathbf{1}_{\mathcal{E}} = [1 \dots 1]^{|\mathcal{E}|}$ is a vector of ones. Note that $\mathbf{V}^\top \mathbf{1}_{\mathcal{E}} \in \mathbb{R}^d_+$ does not depend on the input triple. Therefore, given a mini-batch of triples B, computing $\log p(o \mid s,r)$ requires time $\mathcal{O}((|\mathcal{E}| + |B|) \cdot d)$ and space $\mathcal{O}(|B| \cdot d)$, which is much lower than the complexity on CP showed above, and we can still leverage GPU parallelism. For CP^2, the complexity is similar to the derivation of the partition function complexity showed in Appendix C.4.1. That is, for CP^2 we can write

$$\begin{aligned} \sum_{o' \in \mathcal{E}} \phi_{\text{CP}^2}(s,r,o') &= \sum_{o' \in \mathcal{E}} \left(\sum_{i=1}^{d} e_{si} w_{ri} e_{oi} \right)^2 = \sum_{i=1}^{d} \sum_{j=1}^{d} e_{si} e_{sj} w_{ri} w_{rj} \left(\sum_{o' \in \mathcal{E}} e_{o'i} e_{o'j} \right) \\ &= (\mathbf{e}_s \odot \mathbf{w}_r)^\top (\mathbf{V}^\top \mathbf{V}) (\mathbf{e}_s \odot \mathbf{w}_r) \end{aligned}$$

where $\mathbf{e}_s, \mathbf{w}_r, \mathbf{e}_o \in \mathbb{R}^d$, $\mathbf{V} \in \mathbb{R}^{|\mathcal{E}| \times d}$. Note that the matrix $\mathbf{V}^\top \mathbf{V} \in \mathbb{R}^{d \times d}$ does not depend on the input triple. Therefore, given a mini-batch of triples B, computing $\log p(o \mid s,r)$ requires time $\mathcal{O}((|\mathcal{E}| + |B|) \cdot d^2)$ and space $\mathcal{O}(|B| \cdot d)$. While the time complexity is quadratic in the embedding size d, it is still much lower than the time complexity on CP. A similar discussion can also be carried out for the other KGE models and the corresponding GeKCs, which retrieves the complexities showed in Table C.1.

C.4.3 Training Speed-up Benchmark Details

In this section we report the details about the training benchmark on ComplEx, ComplEx$^+$ and ComplEx^2, whose results are showed in Fig. 3. We measure time and peak GPU memory usage required for computing the PLL objective (Eq. (1)) *and* to do an optimisation step for a single batch on ogbl-wikikg2 [32], a large knowledge graph with millions of entities (see Table F.1). We fix the embedding size to $d = 100$ for the three models. For the benchmark with increasing batch size, we keep all the entities and increase the batch size from 100 to 5000. For the benchmark with increasing number of entities, we keep the batch size fixed to 500 (the maximum allowed for ComplEx by our GPUs) and progressively increase the number of entities, from about $3 \cdot 10^5$ to $2.5 \cdot 10^6$. We report the average time over 25 independent runs on a single Nvidia RTX A6000 with 48 GiB of memory.

D Distribution of Scores

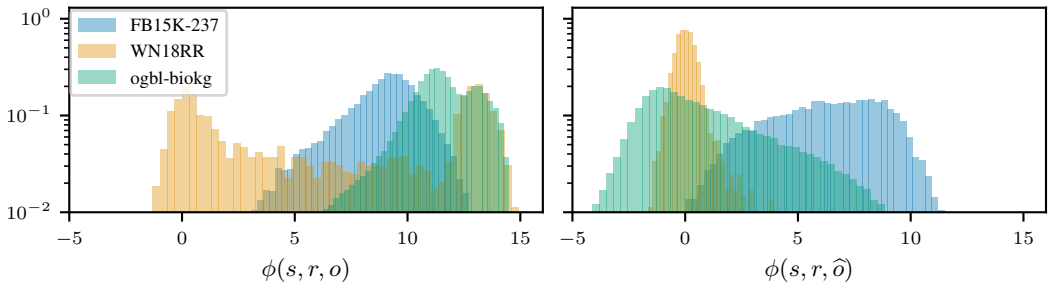


Figure D.1: **Scores are mostly non-negative.** Histograms of the scores assigned by ComplEx to existing validation triples (left) and their perturbation (right) on three data sets. The vast majority of triple scores are non-negative, suggesting that squaring them has minimal effect on the rankings.

In Fig. D.1 we show the histograms of the scores assigned to the validation triples and their perturbations of three data sets (see Appendix F.1). Following Socher et al. [59], we generate triple perturbations that are challenging for link prediction. That is, for each validation triple (s, r, o), the corresponding perturbation $(s, r, \widehat{o})$ is obtained by replacing the object with a random entity that has appeared at least once as an object in a training triple with predicate r. The bottom line is that scores are mostly non-negative, and hence can be used as a heuristic to effectively initialise GeKCs or quickly distil them (e.g., on FB15K-237), as we further discuss in Appendix F.5.1.

E Reconciling Knowledge Graph Embeddings Interpretations

Triples as boolean variables. KGE models such as CP, RESCAL, TuckER and ComplEx have been historically introduced as factorizations of a tensor-representation of a KG, which we discuss next. In fact, a KG $\mathcal{G}$ can be represented as a 3-way binary tensor $\mathbf{Y} \in \{0, 1\}^{|\mathcal{E}|\times|\mathcal{R}|\times|\mathcal{E}|}$ in which every entry Y_{sro} is 1 if $(s, r, o) \in \mathcal{G}$ and 0 otherwise [48]. Under this light, a KGE model like RESCAL factorises every slice $\mathbf{Y}_r \in \{0, 1\}^{|\mathcal{E}|\times|\mathcal{E}|}$, corresponding to the predicate r as $\mathbf{E}\mathbf{W}_r\mathbf{E}^\top$ where $\mathbf{E}$ is an $|\mathcal{E}| \times d$ matrix comprising the entity embeddings and $\mathbf{W}_r$ is an $d \times d$ matrix containing the embeddings for the predicate r. To deal with uncertainty and incomplete KGs, Y_{sro} can be interpreted as a Bernoulli random variable. As such, its distribution becomes $p(Y_{sro} = 1 \mid s, r, o)$ which is usually modelled as $\sigma(\phi(s, r, o))$, where σ denotes the logistic function and ϕ is the score function of a KGE model. Note that this distribution of triple introduces $|\mathcal{E} \times \mathcal{R} \times \mathcal{E}|$ random variables, one for each possible triple.

KGE models as estimators of a distribution over KGs. At the same time, the interpretation of triples as boolean variables *induces a distribution over possible KGs*, $q(\mathcal{G})$, which is the distribution over all possible binary tensors $p(\mathbf{Y})$. The probability of a KG $\mathcal{G}$ can therefore be computed as the product of the likelihoods of all variables Y_{sro}, i.e., $q(\mathcal{G}) = \prod_{(s,r,o)\in\mathcal{G}} p(Y_{sro} = 1 \mid s, r, o) \cdot \prod_{(s,r,o)\notin\mathcal{G}} p(Y_{sro} = 0 \mid s, r, o)$. Note that (re-)normalising this distribution is intractable in general, as it would require summing over all possible $2^{|\mathcal{E}\times\mathcal{R}\times\mathcal{E}|}$ binary tensors. This is why historically KGE models have been interpreted as energy-based models, by directly optimising for $\phi(s, r, o)$, interpreted as the negated energy associated to every triple, and not $p(Y_{sro} = 1 \mid s, r, o)$ (see Section 2). This has been done via negative sampling or other contrastive learning objectives [6, 7]. We point out that this very same interpretation can be found in the literature of probabilistic logic programming [25], probabilistic databases (PDBs) [14] and statistical relational learning (see Section 6) where the distribution over possible "worlds" is over sets of boolean assignments to ground atoms or facts, or tuples in a PDB, each interpreted as Bernoulli random variables.

Estimating a distribution over triples. In this work, instead, we interpret existing KGE models and our GeKCs as models that encode a possibly unnormalised probability distribution over three random variables, S, R, O, which induces a distribution over triples that is tractable to renormalise.[7]

[7]The polynomial cost of renormalising an energy-based KGE is unfortunately infeasible for real-world KGs, see Section 2.

To reconcile these two perspectives, we interpret the probability of a triple $p(s, r, o)$ to be proportional to the probability of all KGs $\mathcal{G}$ where (s, r, o) holds, i.e., those $\mathcal{G}$ such that $(s, r, o) \in \mathcal{G}$. Intuitively, a triple will be more probable to exist if it does exist in highly probable KGs. More formally, given q a probability distribution over KGs, we define p as an unnormalised probability distribution over triples, i.e., $\mu(s, r, o) \propto p(s, r, o)$, where

$$\mu(s, r, o) = \sum_{\substack{\mathcal{G} \in \mathcal{H} \\ (s,r,o) \in \mathcal{G}}} q(\mathcal{G}) = \sum_{\mathcal{G} \in \mathcal{H}} q(\mathcal{G}) \cdot \mathbb{1}\{(s, r, o) \in \mathcal{G}\} = \mathbb{E}_{\mathcal{G} \sim q}[\mathbb{1}\{(s, r, o) \in \mathcal{G}\}] \tag{12}$$

and $\mathcal{H} = 2^{\mathcal{E} \times \mathcal{R} \times \mathcal{E}}$ denotes the set of all possible KGs. Computing the expectation in Eq. (12) exactly is equivalent to solving a *weighted model counting* (WMC) problem [10], where we sum the probabilities of all possible KGs containing (s, r, o). Alternatively, it is equivalent to computing the probability of the simplest possible query in a PDB (i.e., asking for a single tuple), where each stored tuple is interpreted as an independent Bernoulli random variable Y_{sro}. Therefore, we have that $\mu(s, r, o)$ is simply the likelihood that Y_{sro} is true, i.e., $p(Y_{sro} = 1 \mid s, r, o)$. Furthermore, the normalisation constant of μ (Eq. (12)) can be written as

$$Z = \sum_{\substack{(s,r,o) \in \\ \mathcal{E} \times \mathcal{R} \times \mathcal{E}}} \mu(s, r, o) = \sum_{\mathcal{G} \in \mathcal{H}} q(\mathcal{G}) \cdot \sum_{\substack{(s,r,o) \in \\ \mathcal{E} \times \mathcal{R} \times \mathcal{E}}} \mathbb{1}\{(s, r, o) \in \mathcal{G}\} = \sum_{\mathcal{G} \in \mathcal{H}} q(\mathcal{G}) \cdot |\mathcal{G}| = \mathbb{E}_{\mathcal{G} \sim q}[|\mathcal{G}|]$$

which is the expected size of a KG according to the probability distribution q. Written in this way, however, computing Z through $q(\mathcal{G})$ is intractable. For this reason, we directly encode μ with GeKCs and compute Z by summing over all triples, and therefore without modelling $q(\mathcal{G})$.

Further interpretations in related works. Under the interpretation of a KG as a PDB, Friedman and Van den Broeck [26] further decompose the likelihood that Y_{sro} is true as

$$p(Y_{sro} = 1 \mid s, r, o) = p(E_s = 1 \mid s) \cdot p(T_r = 1 \mid r) \cdot p(E_o = 1 \mid o)$$

where E_s, E_o, T_r are new Bernoulli variables that are assumed to be conditionally independent given the parameters of the PDB. That is, instead of introducing one random variable per triple, they introduce one random variable per entity and predicate. In this framework, they reinterpret the score function of DISTMULT, a simplified variant of CP, as an implicit circuit that models an unnormalized distribution over the collection of variables $\mathbf{Z} = \{E_u\}_{u \in \mathcal{E}} \cup \{T_r\}_{r \in \mathcal{R}}$, trained by negative sampling. This decomposition permits to compute the probability of any database query efficiently, which otherwise is known to be either a PTIME or a #P-hard problem, depending on the query type [15]. If we were to interpret our distribution $\mu(S = s, R = r, O = o)$ as the unnormalized marginal distribution $p(E_s = 1, T_r = 1, E_o = 1) = \sum_{\mathbf{z}'} p(E_s = 1, T_r = 1, E_o = 1, \mathbf{Z}' = \mathbf{z}')$, where $\mathbf{Z}' = \mathbf{Z} \setminus \{E_s, T_r, E_o\}$, we could equivalently compute any probabilistic query efficiently. Note that under this interpretation, training our GeKCs by MLE over S, R, O would be equivalent to maximise a composite marginal log-likelihood [70] over $\mathbf{Z}$.

F Empirical Evaluation

F.1 Datasets Statistics

Table F.1 shows statistics of commonly-used datasets to benchmark KGE models for link prediction. We employ standard benchmark datasets [62, 21, 32] whose number of entities (resp. predicates) ranges from ≈14k to ≈2.5M (resp. from 11 to ≈500).

Table F.1: **Dataset statistics.** Statistics of multi-relational knowledge graphs: number of entities ($|\mathcal{E}|$), number of predicates ($|\mathcal{R}|$), number of training/validation/test triples.

Dataset	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{R}\rvert$	# Train	# Valid	# Test
FB15k-237 [62]	14,541	237	272,115	17,535	20,466
WN18RR [21]	40,943	11	86,835	3,034	3,134
ogbl-biokg [32]	93,773	51	$4{,}763 \cdot 10^3$	$163 \cdot 10^3$	$163 \cdot 10^3$
ogbl-wikikg2 [32]	$2.5 \cdot 10^6$	535	$16{,}109 \cdot 10^3$	$429 \cdot 10^3$	$598 \cdot 10^3$

F.2 Metrics

Mean reciprocal rank and hits at k. Given a test triple (s, r, o), we rank the possible object o' (resp. subject s') completions to link prediction queries $(s, r, ?)$ (resp. $(?, r, o)$) based on their scores in descending order. The position of the test triple (s, r, o) in the ranking of object (resp. subject) completed queries (s, r, o') (resp. (s', r, o)) is then used to compute the *mean reciprocal rank* (MRR)

$$\text{MRR} = \frac{1}{2|\mathcal{G}_{\text{test}}|} \sum_{(s,r,o)\in\mathcal{G}_{\text{test}}} \left(\frac{1}{\text{rank}(o \mid s, r)} + \frac{1}{\text{rank}(s \mid r, o)} \right)$$

where $\mathcal{G}_{\text{test}}$ denotes the set of test triples, and $\text{rank}(o \mid s, r), \text{rank}(s \mid r, o)$ denote respectively the positions of the true completion (s, r, o) in the rankings of object and subject completed queries. The fraction of *hits at* k (Hits@k) for $k > 0$ is computed as

$$\text{Hits@}\, k = \frac{1}{2|\mathcal{G}_{\text{test}}|} \sum_{(s,r,o)\in\mathcal{G}_{\text{test}}} \left(\mathbb{1}\{\text{rank}(o \mid s, r) \leq k\} + \mathbb{1}\{\text{rank}(s \mid r, o) \leq k\} \right).$$

Consistently with existing works on link prediction [56, 12], the MRRs and Hits@k metrics are computed under the *filtered* setting, i.e., we rank true completed triples against potential ones that do not appear in the union of training, validation and test splits.

Semantic consistency score. Let K be a logical constraint encoding some background knowledge over variables S, R and O. Given a test triple (s, r, o), we first rank the possible completions to link prediction queries in the same way as for computing the MRR. Then, the *semantic consistency score* (Sem@k) [33] for some integer $k > 0$ is computed as

$$\text{Sem@}k = \frac{1}{2k|\mathcal{G}_{\text{test}}|} \sum_{(s,r,o)\in\mathcal{G}_{\text{test}}} \left(\sum_{o'\in\mathcal{A}^k_O(s,r,o)} \mathbb{1}\{(s, r, o') \models K\} + \sum_{s'\in\mathcal{A}^k_S(s,r,o)} \mathbb{1}\{(s', r, o) \models K\} \right)$$

where $\mathcal{G}_{\text{test}}$ denotes the set of test triples, $\mathcal{A}^k_O(s, r, o)$ (resp. $\mathcal{A}^k_S(s, r, o)$) denotes the list of the top-k candidate object (resp. subject) completions to the link prediction query $(s, r, ?)$ (resp. $(?, r, o)$), and $(s, r, o) \models K$ if and only if (s, r, o) satisfies K.

F.3 Empirical KTD Score

Let $\mathcal{F} = \{x_i\}_{i=1}^m, \mathcal{G} = \{y_j\}_{j=1}^n$ two sets of triples that are drawn i.i.d. from two distributions $\mathbb{P}, \mathbb{Q}$ over triples. We compute the empirical KTD score with an *unbiased estimator* [28] $\text{KTD}_u(\mathcal{F}, \mathcal{G})$ as

$$\frac{1}{m(m-1)} \sum_{i\neq j}^{m} k(\psi(x_i), \psi(x_j)) + \frac{1}{n(n-1)} \sum_{i\neq j}^{n} k(\psi(y_i), \psi(y_j)) - \frac{2}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} k(\psi(x_i), \psi(y_j)).$$

For each data set, we compute the empirical KTD score between $n = 25,000$ triples sampled from GeKCs and m test triples. In case of $m > n$, we sample n triples randomly, uniformly and without replacement from the set of test triples. The time complexity of computing the KTD score is $\mathcal{O}(nmh)$, where h denotes the size of triple latent representations ($h = 4000$ in our case, see Section 7.3). For efficiency reasons, we therefore follow Binkowski et al. [4] and randomly extract two batches of 1000 triples each from both the generated and the test triples sets and compute the empirical KTD score on them. We repeat this process 100 times and report the average and standard deviation in Table F.6.

F.4 Experimental Setting

Hyperparameters. All models are trained by gradient descent with either the PLL or the MLE objective (Eqs. (1) and (2)). We set the weights $\omega_s, \omega_r, \omega_o$ of the PLL objective all to one, as to retrieve a classical pseudo-log-likelihood [70]. Note that Chen et al. [12] set ω_s, ω_o to one and treat ω_r as an additional hyperparameter instead that is opportunely tuned. The models are trained until the MRR computed on the validation set does not improve after three consecutive epochs. We fix the embedding size $d = 1000$ for both CP and ComplEx and use Adam [38] as optimiser with 10^{-3} as learning rate. An exception is made for GeKCs obtained via non-negative restriction (Section 4.1), for which a learning rate of 10^{-2} is needed, as we observed very slow convergence rates. We search for the batch size in $\{5 \cdot 10^2, 10^3, 2 \cdot 10^3, 5 \cdot 10^3\}$ based on the validation MRR. Finally, we perform 5 repetitions with different seeds and report the average MRR and two standard deviations in Table F.2.

Parameters initialisation. Following [40, 12], the parameters of CP and ComplEx are initialised by sampling from a normal distribution $\mathcal{N}(0, \sigma^2)$ with $\sigma = 10^{-3}$. Since the embedding values in CP$^+$ and ComplEx$^+$ can be interpreted as parameters of categorical distributions over entities and predicates (see Appendix C.1), we initialise them by sampling from a Dirichlet distribution with concentration factors set to 10^3. To allow unconstrained optimisation for CP$^+$ and ComplEx$^+$, we represent the embedding values by their logarithm and perform computations directly in log-space, i.e., summations and log-sum-exp operations instead of multiplications and summations, respectively. Moreover, the parameters that ensure the non-negativity of ComplEx$^+$ (see Appendix C.2) are initialised by sampling from a normal distribution $\mathcal{N}(0, \sigma^2)$ with $\sigma = 10^{-2}$. We initialise the parameters of CP^2 and ComplEx^2 such that the logarithm of the scores are approximately normally distributed and centred in zero during the initial optimisation steps, since this applies for the scores given by CP and ComplEx. Such initialisation therefore permits a fairer comparison. To do so, we initialise the embedding values by sampling from a log-normal distribution $\mathcal{LN}(\mu, \sigma^2)$, where $\mu = -\log(d)/3 - \sigma^2/2$ for CP and $\mu = -\log(2d)/3 - \sigma^2/2$ for ComplEx, both with $\sigma = 10^{-3}$. The mentioned values for μ can be derived via Fenton-Wilkinson approximation [24]. Even though the parameters of GeKCs obtained via non-monotonic squaring are initialised to be non-negative, they are free of becoming negative during training (as we also confirm in practice).

Hardware. Experiments on the smaller knowledge graphs FB15K-237 and WN18RR were run on a single Nvidia GTX 1060 with 6 GiB of memory, while those on the larger ogbl-biokg and ogbl-wikikg2 were run on a single Nvidia RTX A6000 with 48 GiB of memory.

F.5 Additional Experimental Results

F.5.1 Link Prediction Results

In this section, we present the additional results regarding the link prediction experiments showed in Section 7.1 and analyse different metrics and learning settings.

Statistical tests and hits at k. Table F.2 shows the best test average MRRs (see Appendix F.2) with two standard deviation and average training time across 5 independent runs with different seeds. We highlight the best results in bold according to a one-sided Mann–Whitney U test with a confidence level of 99%. The showed results in terms of MRRs are also confirmed in Table F.3, which shows the best average Hits@k (see Appendix F.2) with $k \in \{1, 3, 10\}$.

Average log-likelihood. For the best GeKCs for link prediction showed in Table F.2, we report the average log-likelihood of test triples and two standard deviations (across 5 independent runs) in Table F.4. We again highlight the best results in bold, according to a one-sided Mann–Whitney U test.

Quickly distilling parameters. As discussed in Section 4.2, since learned KGE models mostly assign non-negative scores to triples (see Appendix D) we can initialise the parameters of GeKCs obtained by squaring with the parameters of already-learned KGE models, without losing much in terms of link prediction performances. Here, we test this hypothesis and fine-tune GeKCs initialised in this way by using either the PLL or MLE objectives (Eqs. (1) and (2)). To do so, we first collect the parameters of the best CP and ComplEx found for link prediction (see Section 7.1). Then, we initialise GeKCs derived by squaring with these parameters and fine-tune them until the MRR computed on validation triples does not improve after three consecutive epochs. We employ Adam [38] as optimiser, and we search for the batch size in $\{5 \cdot 10^2, 10^3, 2 \cdot 10^3, 5 \cdot 10^3\}$ and learning rate in $\{10^{-3}, 10^{-4}\}$, as fine-tuning may require a lower learning rate than the one used in previous experiments (see Appendix F.4). Table F.5 shows the MRRs achieved by CP, ComplEx and the corresponding GeKCs obtained via squaring that are initialised by distilling the parameters from the already-trained CP and ComplEx. On FB15K-237 and WN18RR, distilling parameters induces a substantial improvement in terms of MRR with respect to CP^2 and ComplEx^2 whose parameters have been initialised randomly (see Appendix F.4). Furthermore, for ComplEx^2 and on WN18RR and ogbl-biokg we achieved similar MRRs with respect to ComplEx *without* the need of fine-tuning.

Table F.2: **GeKCs are competitive with their energy-based counterparts.** Best test MRRs (and two standard deviations) of CP, ComplEx and GeKCs trained with the PLL and MLE objectives (Eqs. (1) and (2)). In parentheses we show the average training time (in minutes).

Model	FB15k-237		WN18RR		ogbl-biokg	
	PLL	MLE	PLL	MLE	PLL	MLE
CP	0.310 ±0.001 (8)	—	**0.105 ±0.007** (11)	—	0.831 ±0.001 (136)	—
CP$^+$	0.237 ±0.003 (1)	0.230 ±0.003 (1)	0.027 ±0.002 (1)	0.026 ±0.001 (1)	0.496 ±0.013 (172)	0.501 ±0.010 (142)
CP^2	**0.315 ±0.003** (8)	0.282 ±0.004 (7)	**0.104 ±0.001** (23)	0.091 ±0.004 (23)	**0.848 ±0.001** (66)	0.829 ±0.001 (61)
ComplEx	**0.342 ±0.005** (36)	—	**0.471 ±0.002** (16)	—	0.829 ±0.001 (180)	—
ComplEx$^+$	0.214 ±0.003 (10)	0.205 ±0.006 (5)	0.030 ±0.001 (6)	0.029 ±0.001 (3)	0.503 ±0.014 (245)	0.516 ±0.009 (212)
ComplEx^2	0.334 ±0.001 (10)	0.300 ±0.003 (16)	0.420 ±0.011 (37)	0.391 ±0.004 (19)	**0.858 ±0.001** (71)	0.840 ±0.001 (59)

Table F.3: **Hits@k results.** Average test Hits@k for $k \in \{1, 3, 10\}$ of CP, ComplEx and CP^2 and ComplEx^2 trained with the PLL or MLE objectives.

	FB15k-237						WN18RR						ogbl-biokg					
	PLL			MLE			PLL			MLE			PLL			MLE		
Model $k=$	1	3	10	1	3	10	1	3	10	1	3	10	1	3	10	1	3	10
CP	22.4	34.1	48.2	—	—	—	7.5	12.1	16.8	—	—	—	76.4	88.1	95.0	—	—	—
CP$^+$	17.0	25.8	36.7	16.7	24.9	35.4	1.7	2.7	4.5	1.6	2.5	4.4	38.0	54.4	73.4	38.4	55.0	74.4
CP^2	23.1	34.8	48.2	20.5	30.8	43.5	6.7	12.1	17.6	5.9	10.7	15.3	78.6	89.5	95.7	76.1	88.1	95.0
ComplEx	25.2	37.5	52.5	—	—	—	43.3	48.6	54.6	—	—	—	76.1	87.9	95.0	—	—	—
ComplEx$^+$	15.7	23.1	31.7	15.0	22.1	30.4	1.5	2.7	4.5	1.6	2.5	4.4	38.8	55.1	74.1	40.0	56.7	75.9
ComplEx^2	24.5	36.9	51.1	21.6	33.0	46.7	36.0	45.6	52.4	34.5	42.3	46.9	80.0	90.1	95.8	77.5	88.8	95.4

Table F.4: **Better distribution estimation with GeKCs obtained via squaring.** Average log-likelihood of test triples achieved by baselines and GeKCs trained with the PLL or MLE objectives.

Model	FB15k-237		WN18RR		ogbl-biokg	
Uniform	-24.638		-23.638		-26.829	
NNMFAug	-19.270		-22.938		-17.562	
	PLL	MLE	PLL	MLE	PLL	MLE
CP$^+$	-16.773 ±0.040	-16.592 ±0.059	**-21.987 ±0.006**	-22.103 ±0.010	-17.900 ±0.048	-17.416 ±0.049
CP^2	-17.105 ±0.031	**-15.982 ±0.028**	-24.911 ±0.241	-26.352 ±0.077	-17.231 ±0.059	**-16.533 ±0.013**
ComplEx$^+$	-17.507 ±0.035	-17.592 ±0.039	-21.233 ±0.058	-21.432 ±0.008	-18.716 ±0.088	-17.749 ±0.019
ComplEx^2	-17.100 ±0.026	**-15.744 ±0.041**	**-19.522 ±0.530**	**-19.739 ±0.214**	-17.340 ±0.022	**-16.518 ±0.003**

Table F.5: **Distilling parameters can improve performances.** Test MRRs achieved by CP, ComplEx and GeKCs obtained by squaring (Section 4.2). For CP^2 and ComplEx^2 we report the best MRRs achieved by distilling the parameters from the already-learned CP and ComplEx (denoted with $\star$), and with † we denote those results for which further fine-tuning with the PLL or MLE objectives did not bring better results. We underline results for which distilling parameters increased the MRR.

Model	FB15k-237		WN18RR		ogbl-biokg	
	PLL	MLE	PLL	MLE	PLL	MLE
CP	0.311	—	0.108	—	0.831	—
ComplEx	0.344	—	0.470	—	0.829	—
CP^2	0.317	0.285	0.103	0.089	0.849	0.830
CP^2 $\star$	<u>0.327</u>	<u>0.315</u>	0.102	<u>0.115</u>	0.851	0.828 †
ComplEx^2	0.333	0.301	0.416	0.390	0.859	0.839
ComplEx^2 $\star$	<u>0.342</u>	<u>0.340</u>	<u>0.462</u> †	<u>0.463</u>	0.859	0.828 †

F.5.2 Quality of Sampled Triples Results

In this section, we provide additional results regarding the evaluation of the quality of triples sampled by GeKCs (see Section 7.3). For these experiments, we search for the same hyperparameters as for the experiments on link prediction (see Appendix F.4), and train GeKCs until the average log-likelihood computed on validation triples does not improve after three consecutive epochs.

Table F.6 shows the mean empirical KTD score and one standard deviation (see Appendix F.3). In addition, we visualise triple embeddings of sampled and test triples in Fig. F.1 by leveraging t-SNE [69] as a method for visualising high-dimensional data. In particular, we apply the t-SNE method implemented in scikit-learn with perplexity 50 and number of iterations $5 \cdot 10^3$, while other parameters are fixed to their default value. As showed in Fig. F.1c, an empirical KTD score close to zero translates to an high clusters similarity between embeddings of sampled and test triples.

Table F.6: **GeKCs trained by MLE generate new likely triples.** Empirical KTD scores between test triples and triples generated by baselines and GeKCs trained with the PLL objective or by MLE (Eqs. (1) and (2)). Lower is better.

Model	FB15k-237		WN18RR		ogbl-biokg	
Training set	0.055 ±0.007		0.260 ±0.013		0.029 ±0.010	
Uniform	0.589 ±0.012		0.766 ±0.036		1.822 ±0.044	
NNMFAug	0.414 ±0.014		0.607 ±0.028		0.518 ±0.035	
	PLL	MLE	PLL	MLE	PLL	MLE
CP$^+$	0.404 ±0.016	0.433 ±0.015	0.633 ±0.033	**0.578 ±0.029**	0.966 ±0.040	0.738 ±0.030
CP^2	0.253 ±0.014	**0.070 ±0.007**	0.768 ±0.036	0.768 ±0.036	0.039 ±0.009	**0.017 ±0.013**
ComplEx$^+$	0.336 ±0.016	0.323 ±0.015	0.456 ±0.018	0.478 ±0.019	0.175 ±0.019	0.097 ±0.013
ComplEx^2	0.326 ±0.016	**0.102 ±0.010**	0.338 ±0.020	**0.278 ±0.017**	0.104 ±0.010	**0.034 ±0.007**

(a) FB15k-237
KTD := 0.102 ± 0.010

(b) WN18RR
KTD := 0.278 ± 0.017

(c) ogbl-biokg
KTD := 0.034 ± 0.007

Figure F.1: **Sampled triples are *close* to test triples.** t-SNE [69] visualisations of the embeddings of test triples (in blue) and triples sampled by ComplEx^2 (in orange). The distribution shift between training and test triples on WN18RR mentioned in Section 7.3 is further confirmed in Fig. F.1b, as it shows a region of test triples (at the bottom and in blue) that is not covered by many generated triples.

F.5.3 Calibration Diagrams

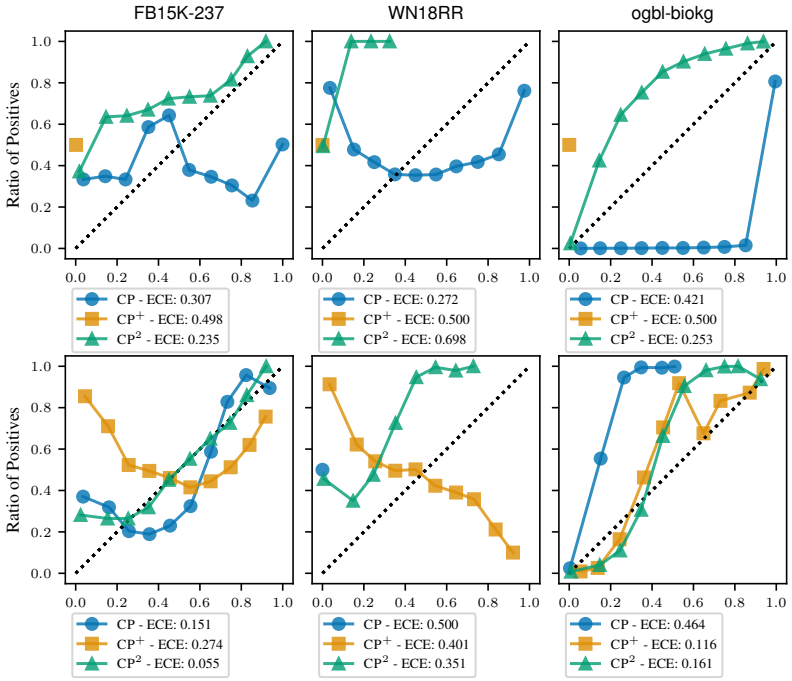
Existing works on studying the calibration of KGE models are based on interpreting each possible triple (s, r, o) as an independent Bernoulli random variable Y_{sro} whose likelihood is determined by the score function ϕ, i.e., $\Pr(Y_{sro} = 1 \mid s, r, o) = \sigma(\phi(s, r, o))$ [61, 54, 79], where σ denotes the logistic function. While GeKCs encode a probability distribution over all possible triples, this does not impede us to reinterpret them to model the likelihood of each Y_{sro} by still considering scores in log-space as negated energies (see Section 2). Therefore, to evaluate the calibration of GeKCs encoding a non-negative score function ϕ_{pc} (see Section 4) we compute the probability of a triple (s, r, o) being true as $p(Y_{sro} = 1 \mid s, r, o) := \sigma(\log \phi_{\mathsf{pc}}(s, r, o))$. However, the usage of the logistic

function might give misleading results in case of scores not being centred around zero on average. Therefore, we also report calibration diagrams (see paragraph below) where the $p(Y_{sro} = 1 \mid s, r, o)$ is obtained via min-max normalisation of the scores given by KGE models (the logarithm of the scores given for GeKCs), where the minimum and maximum are computed on the training triples. Note that several ex-post (re-)calibration techniques are available [61, 79], but they should benefit GeKCs as they do with existing KGE models.

Setting and metrics. To plot calibration diagrams, we follow Socher et al. [59] and sample challenging negative triples, i.e., for each test triple (s, r, o) we sample an unobserved perturbed one $(s, r, \widehat{o})$ by replacing the object with an entity that has appeared at least once with the predicate r in the training data. We then compute the *empirical calibration error* (ECE) [79] as $\mathrm{ECE} := \frac{1}{b}\sum_{i=1}^{b} |p_j - f_j|$, where b is the number of uniformly-chosen bins for the interval $[0, 1]$ of triple probabilities, and p_j, f_j are respectively the average probability and relative frequency of actually existing triples in the j-th bin. The lower the ECE score, the better calibrated are the predictions, as they are closer to the empirical frequency of triples that do exist in each bin. The calibration curves are plotted by considering the relative frequency of existing triples in each bin, and curves closer to the main diagonal indicate better calibrated predictions.

GeKCs are more calibrated out-of-the-box. Fig. F.2 (resp. Fig. F.3) show calibration diagrams for GeKCs derived from CP and ComplEx trained with the MLE objective (Eq. (2)) (resp. PLL objective (Eq. (1))). In 19 cases over 24, GeKCs obtained via squaring (Section 4.2) achieve lower ECE scores and better calibrated curves than CP and ComplEx. While GeKCs obtained via non-negative restriction (Section 4.1) are not well calibrated when using the logistic function, on ogbl-biokg [32] they are still better calibrated than CP and ComplEx when probabilities are obtained via min-max normalisation. Furthermore, on WN18RR GeKCs achieved the highest ECE scores (corresponding to poorly-calibrated predictions), which could be explained by the distribution shift between training and test triples that was observed for this KG in Section 7.3 and further confirmed in Appendix F.5.2.

(a)



(b)

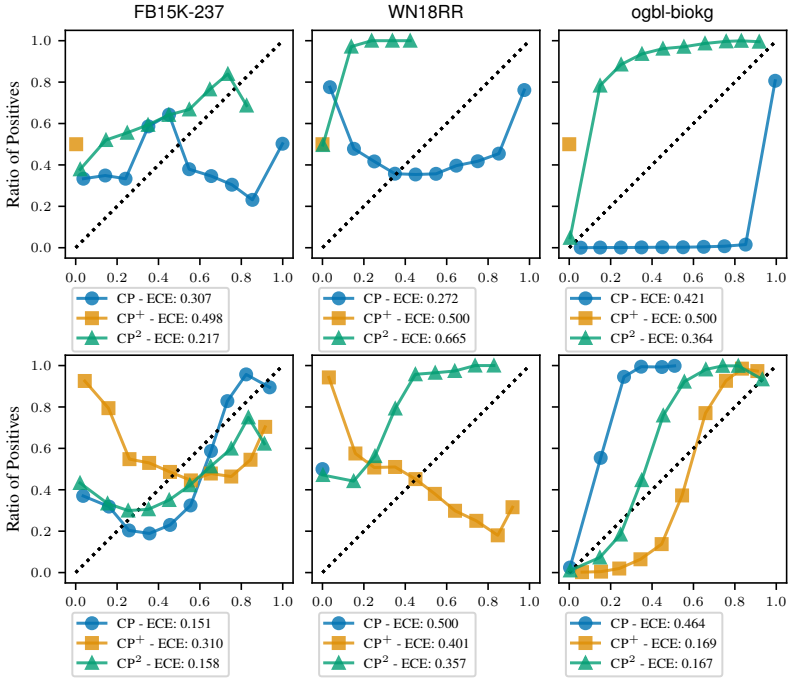
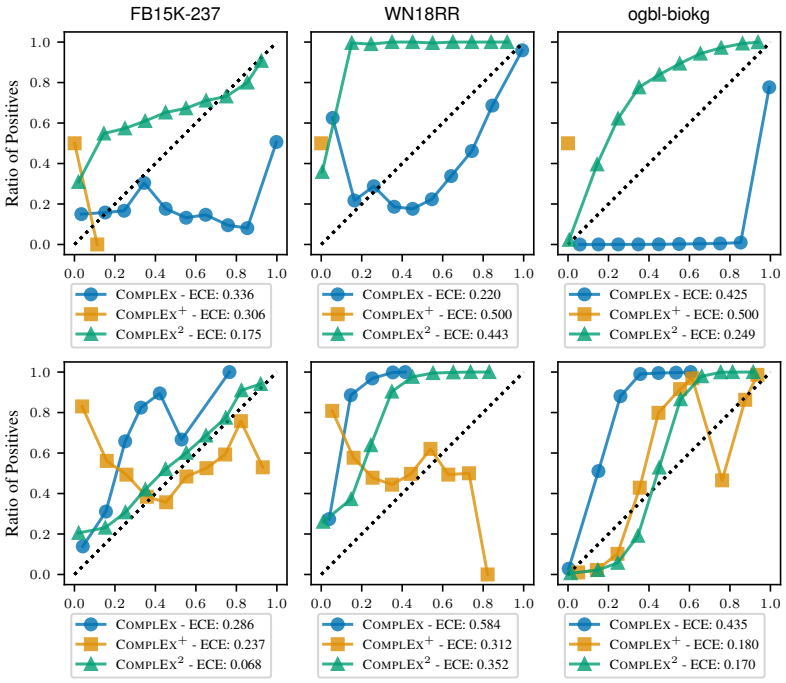


Figure F.2: **Better calibrated predictions with CP^2.** Calibration diagrams of CP, CP$^+$ and CP^2 trained with either the PLL (Fig. F.2a) or MLE (Fig. F.2b) objectives. The probability of triples are obtained via the application of the logistic function (rows above) and min-max normalisation (rows below). See Appendix F.5.3 for details. The calibration curves for CP$^+$ where triple probabilities are obtained with the logistic function do not provide any meaningful information, as the logarithm of their scores are generally distributed over large negative values.

(a)



(b)

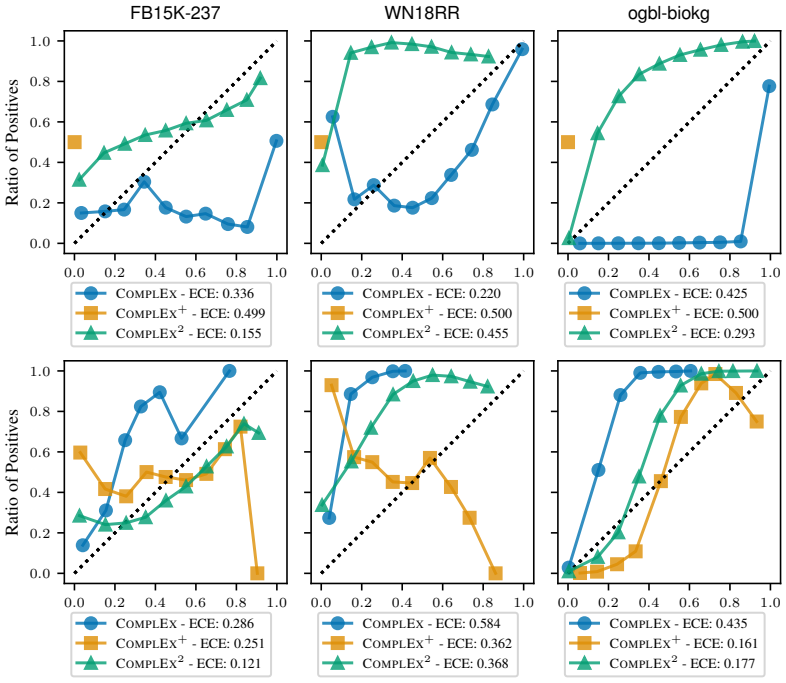


Figure F.3: **Better calibrated predictions with ComplEx^2.** Calibration diagrams of ComplEx, ComplEx$^+$ and ComplEx^2 trained with either the PLL (Fig. F.3a) or MLE (Fig. F.3b) objectives. The probability of triples are obtained via the application of the logistic function (rows above) and min-max normalisation (rows below). See Appendix F.5.3 for details. The calibration curves for ComplEx$^+$ where triple probabilities are obtained with the logistic function do not provide any meaningful information, as the logarithm of their scores are generally distributed over large negative values.