SKILLGNN: LARGE LANGUAGE MODEL-GUIDED COMPONENT EVOLUTION FOR KNOWLEDGE GRAPH COMPLETION

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph neural networks (GNNs) for knowledge graph completion still rely heavily on manually designed components. Large language models (LLMs) can generate alternative designs, but feedback on individual candidates does not by itself ensure that design experience accumulates across iterations. We present SkillGNN, a framework that couples component-wise GNN evolution with the refinement of a reusable textual design Skill. The Skill guides an LLM agent to propose local computations within a fixed component interface while preserving the surrounding GNN architecture. A task-grounded evaluator combines validity and structural checks with subgraph training, providing diagnostic feedback that SkillOpt uses to refine the Skill for subsequent proposals. Experiments on WN18RR and FB15K-237 show that the evolved NBFNet scorers improve full-graph link prediction over NBFNet and one-shot LLM component design when trained from scratch. Additional experiments examine the influence of the design LLM and show improvements from evolving NBFNet's node-update component and CompGCN's scoring component.

1 INTRODUCTION

Graph neural networks (GNNs) have been developed for relational reasoning and knowledge graph completion, with representative models including R-GCN, CompGCN, and NBFNet (Schlichtkrull et al., 2018; Vashishth et al., 2020; Zhu et al., 2021). Despite this progress, the design of many internal computations in these models remains manual, with researchers specifying how messages encode relations, how neighborhood evidence is aggregated, and how propagated representations are converted into predictions. Learning model parameters therefore leaves a separate design problem unresolved: selecting the structural inductive biases that define the computations performed with those parameters.

This design problem is particularly important for strong graph reasoning models, where component-level choices determine how relational evidence is transformed, combined, and ultimately used for prediction (Vashishth et al., 2020; Zhu et al., 2021). These choices are often made manually and therefore reflect the designer's existing expertise. Even when a hand-designed component performs well, this design process can explore only a limited set of possible computations, leaving open whether another component could better complement the representations produced by the rest of the model. NBFNet provides a strong framework for studying this question by combining query-conditioned propagation with a final scoring component for link prediction (Zhu et al., 2021). Its propagation layers produce informative conditional representations, while the scoring component maps these representations to entity scores. This motivates a more focused question of whether systematic component exploration can identify a local computation that makes better use of the representations already available in a strong model.

LLMs provide a practical mechanism for such exploration because they can generate executable candidates and refine them through external evaluation (Romera-Paredes et al., 2024; Liu et al., 2024). Recent coding systems provide a concrete example of this feedback loop. Agentic Verifier improves solution selection through multi-turn interaction with code execution environments and targeted test generation, while SWE-RM shows that fine-grained learned feedback can further improve software-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Table 1: Comparison of representative paradigms for model design, where ✓ and ✗ indicate whether a capability is explicitly incorporated into the method's design process.

Paradigm / Method	Automated design	Task feedback	LLM-guided design	Strategy evolution	Local component evolution
Hand-designed KGE (TransE, RotatE)	✗	✗	✗	✗	✗
Automated KGE search (AutoSF, AutoBLM+)	✓	✓	✗	✗	✗
LLM-based heuristic evolution (EoH)	✓	✓	✓	✗	✗
SkillGNN	✓	✓	✓	✓	✓

engineering agents when execution-based signals are sparse (Ma et al., 2026; Shum et al., 2026). Feedback-driven language agents similarly use textual feedback or external memory to guide subsequent decisions without updating model weights (Shinn et al., 2023; Yang et al., 2026). However, feedback on individual attempts alone does not ensure that design expertise accumulates across iterations. A syntactically valid component may still make limited use of the model's representations or differ only superficially from an existing computation. A single performance score also provides limited guidance about what should change next. Without an explicit mechanism for updating the design guidance, lessons from unsuccessful attempts may not be carried forward as reusable design strategies.

To address this gap, we present **SkillGNN**, a framework for component-wise GNN evolution guided by a reusable and iteratively refined design *Skill*. The Skill organizes knowledge of existing methods, interface constraints, and experience from previous attempts, and an LLM agent uses it to propose a local component. A task-grounded evaluator returns evidence about executability, structural distinctness, and predictive performance after training. Microsoft SkillOpt (Yang et al., 2026) then uses this evaluation feedback to propose Skill revisions, linking component exploration with design-strategy refinement. We apply SkillGNN to the scoring component of NBFNet, preserving its propagation architecture and training objective while evolving only the computation that maps its final representations to entity scores. For efficient repeated evaluation, scoring components that pass the program checks are trained jointly with NBFNet and evaluated on a fixed development subgraph of the original knowledge graph. The selected component structure remains fixed during full-graph training from scratch and subsequent evaluation.

SkillGNN differs from prior model-design approaches in both what is evolved and how design experience is retained. Representative knowledge graph embedding (KGE) methods rely on manually specified scoring functions, as in TransE and RotatE (Bordes et al., 2013; Sun et al., 2019), or automate scoring-function search, as in AutoSF and AutoBLM+ (Zhang et al., 2020a; 2023). EoH performs LLM-guided heuristic evolution for combinatorial optimization rather than targeting trainable GNN components for knowledge graph completion (Liu et al., 2024). SkillGNN instead applies LLM-guided design to local GNN components and refines the reusable design Skill, allowing later proposals to build on accumulated design experience. Table 1 summarizes these distinctions.

Our contributions can be summarized as the following:

- We introduce SkillGNN, a framework for automated component-wise evolution of GNNs. Its reusable Skills retain design knowledge and are refined with evaluation feedback to guide subsequent LLM proposals. The framework can be extended to other GNNs and model architectures by adapting component interfaces and evaluation protocols.

- We develop a task-grounded component evaluator that provides feedback on component validity and predictive performance to guide Skill refinement and component evolution toward improved GNN generalization.

- We demonstrate improved link prediction on WN18RR and FB15K-237 over NBFNet (Zhu et al., 2021) and one-shot LLM design. Further experiments show gains from evolving NBFNet scorers with different design LLMs, as well as from evolving NBFNet's node-update component and CompGCN's scoring component.

2 Related Work

Knowledge graph completion with embeddings and GNNs. Knowledge graph embedding methods encode relational assumptions directly in the scoring function used to rank candidate triples. This design ranges from translation in TransE (Bordes et al., 2013) to bilinear and complex-valued interactions in DistMult, ComplEx, and RotatE (Yang et al., 2015; Trouillon et al., 2016; Sun et al., 2019). ConvE further learns nonlinear entity–relation interactions with a neural scoring function (Dettmers et al., 2018). GNN-based KGC models move part of this computation into relational message passing. R-GCN aggregates relation-specific neighborhood information (Schlichtkrull et al., 2018), CompGCN composes entity and relation representations during propagation (Vashishth et al., 2020), and NBFNet constructs query-conditioned representations for link prediction (Zhu et al., 2021). More recently, EXP-GNN studies propagation-function design by examining how message, aggregation, and update functions affect GNN-based KGC (Gong et al., 2026). This line of work makes component design an explicit modeling problem, but the relevant computations remain selected within researcher-defined design spaces. SkillGNN instead studies whether a local computation can be generated and refined through accumulated design experience.

Automated design search. Once model design is treated as a search problem, automation can operate at different levels. AutoSF (AutoBLM) and its extension AutoBLM+ search structured spaces of scoring functions for knowledge graph learning (Zhang et al., 2020a; 2023). Graph neural architecture search extends this idea beyond scoring functions. GraphNAS uses reinforcement learning to search GNN architectures (Gao et al., 2020), and AutoGFM adapts graph architectures across domains and tasks (Chen et al., 2025). LLM-based methods broaden the search representation further. FunSearch generates executable programs and selects them through external evaluation (Romera-Paredes et al., 2024), whereas EoH evolves heuristic ideas and code for combinatorial optimization (Liu et al., 2024). These methods demonstrate several routes from manual design to automated search, but their search targets are scoring functions, model architectures, or task programs rather than a trainable local component inside an existing KGC GNN. SkillGNN narrows the design object to such a component and evaluates each proposal after integrating it into the existing KGC GNN.

Feedback-driven skill evolution. Automated search also raises the question of how evidence from one attempt should affect later design decisions. Reflexion stores verbal feedback and uses it to influence subsequent trajectories without updating model weights (Shinn et al., 2023). SkillOpt makes this persistent guidance explicit by treating a Skill as editable textual state and revising it from execution traces under validation feedback (Yang et al., 2026). Recent analysis of self-evolving skills further shows that repeated refinement does not produce steady improvement, with successful evolution depending on validation filtering and the feedback available to the optimizer (Liu et al., 2026). SkillGNN builds on this feedback-driven view, but changes the object that the Skill guides. The evolving Skill provides reusable design knowledge for generating trainable GNN components, so evidence from earlier attempts can shape subsequent component proposals.

3 Method

SkillGNN maintains a reusable textual Skill separately from the executable component generated with that Skill. A component design agent uses the current Skill to propose a candidate component under a fixed input–output contract. The component evaluator checks each candidate and trains eligible components jointly with the chosen GNN on a fixed development subgraph. The SkillOpt optimizer uses the resulting feedback to propose a Skill revision. A revision is accepted only if a new component generated with the revised guidance improves MRR on the fixed subgraph validation queries under the same protocol. Figure 1 shows this loop, and Algorithm 1 specifies how revisions are accepted or rejected. After evolution, SkillGNN selects the best-performing component on the fixed evaluation subgraph and trains it jointly with a freshly initialized GNN on the full graph, rather than using the final Skill to generate a new component.

3.1 Component Generation

We describe component generation using the final scoring function of NBFNet (Zhu et al., 2021). For a query $(h, r, ?)$ with head entity h and relation r, the model assigns a score to each candidate

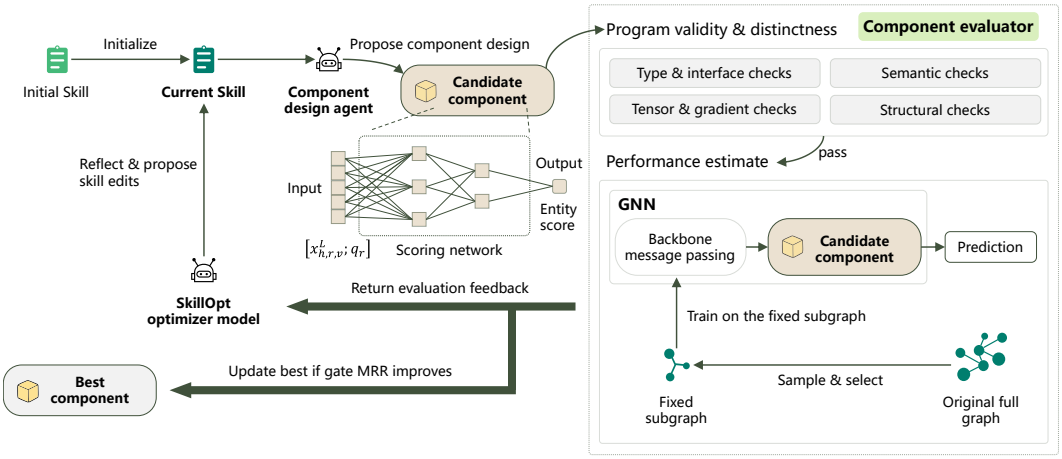


Figure 1: Overview of SkillGNN. The proposed component evaluator combines program checks with a shared GNN training protocol on a fixed development subgraph to provide diagnostic feedback beyond performance scores. SkillOpt uses this feedback to refine the design guidance. After the evolution loop, SkillGNN trains the best-performing component with a freshly initialized GNN on the full graph, rather than using the final Skill to generate a new component.

entity v through

$$z_{h,r,v} = F_\theta(\mathcal{G}_{\text{train}}, h, r, v), \qquad s(h, r, v) = c_{\phi,\omega}(z_{h,r,v}), \tag{1}$$

where $\mathcal{G}_{\text{train}}$ contains training facts and F_θ computes the scoring input using the original propagation architecture with trainable parameters θ. The scoring component $c_{\phi,\omega}$, referred to as the scorer network, has computational structure ϕ and trainable parameters ω. Evolution changes ϕ without altering the message-passing or aggregation architecture. For each evaluated structure, training jointly optimizes θ and ω under the original objective, so the propagation parameters are not frozen.

The scoring component's interface contract $\mathcal{C}$ specifies the permitted inputs and required output. For NBFNet, the input and original scoring function are

$$z_{h,r,v} = [x^L_{h,r,v}; q_r], \qquad c_0(z) = \text{MLP}(z), \tag{2}$$

where $x^L_{h,r,v}$ is the candidate entity's state after L propagation layers and q_r is the query-relation representation. The original scoring MLP is denoted by c_0, which is distinct from a component generated with the initial Skill. Unlike a standalone entity embedding, $x^L_{h,r,v}$ is conditioned on the query. Each proposed computation receives only $[x^L_{h,r,v}; q_r]$ and returns a single entity score. The interface excludes access to intermediate propagation states or the propagation graph, and existing model logits are not supplied as inputs. A residual proposal can nevertheless include c_0 as an internal branch computed from z. Head prediction uses the same interface through NBFNet's existing inverse-relation conversion.

The CompGCN study follows the same local-replacement principle under a different interface from that in Equation (2). We replace only the terminal scoring operator, preserving the architectures of the circular-correlation encoder and ConvE query transformation. CompGCN's original training objective and optimization protocol are also retained. The design procedure therefore uses a model-specific contract for each GNN rather than treating the representations as interchangeable.

Under the contract for the chosen GNN, the design Skill S_t provides reusable textual guidance at outer iteration t, distinct from model weights or executable programs. The initial Skill S_0 records the permitted inputs and describes how established scoring ideas can be combined. The task and interface specifications remain fixed together with the training constraints. Design hypotheses and records of observed successes or failures can be revised as evidence accumulates. The design agent uses this guidance to propose new interactions or residual corrections. A Skill revision therefore changes how a program is generated, whereas replacing ϕ alters the computation executed by the GNN.

For the current design task u_t, the component design agent A_{LLM} uses S_t and $\mathcal{C}$ to generate a structured patch p_t specifying the proposed computation,

$$p_t = A_{\mathrm{LLM}}(S_t, \mathcal{C}, u_t), \qquad \widehat{\phi}_t = \mathrm{Compile}(p_t). \tag{3}$$

The compiler converts p_t into a typed expression tree $\widehat{\phi}_t$ or returns an error if compilation fails. It assembles supported operations rather than executing unrestricted generated Python. Addition and elementwise products can be combined with normalization or projections. Inner products and small trainable interaction modules provide additional ways to combine representations. For NBFNet, one optional residual construction is $c_0(z) + \tanh(a)R_\psi(z)$, where R_ψ is the proposed correction and the scalar gate a is initialized to zero. At initialization, the residual vanishes and the computation reduces to $c_0(z)$, but other proposals are not required to retain this original scoring branch.

The generated component executes the typed expression tree $\widehat{\phi}_t$ on $[x^L_{h,r,v}; q_r]$ to produce the single entity score required by the interface contract. The tree has the same structure for every query, with its relation-dependent values computed from q_r as $g_j(q_r) = \Pi_j(W_j q_r + b_j)$, where the transformation Π_j restricts the output to positive values or a bounded range of phase angles, and W_j and b_j are trainable parameters.

3.2 Component Evaluation

The evaluator checks component validity and estimates predictive performance on a development subgraph fixed before evolution.

We construct this subgraph from training and validation data, using only training triples as propagation edges (Appendix A). Disjoint subsets of the selected validation queries form the *reflect* and *gate* splits. The evaluator uses the *reflect* split for design feedback and checkpoint selection and the *gate* split to assess component performance and proposed Skill updates, while the official test set remains excluded from component search.

Before training, type and interface checks verify compliance with $\mathcal{C}$. Tensor and gradient checks use synthetic inputs to test output shapes, finite outputs and gradients, and gradient connectivity. Semantic checks examine input sensitivity and contradictions between directional claims and head–tail symmetry. Structural checks compare candidates with reference formulas, prior proposals, and fixed excluded structures (Figure 1). These checks determine training eligibility, not mathematical novelty. If a check fails, the evaluator returns the diagnostic without starting training. Previous scores are reused only for identical programs, data, and training and evaluation settings.

Eligible components are trained jointly with the chosen GNN using its original implementation and training settings. For component structure ϕ, checkpoint selection and scoring use mean reciprocal rank (MRR),

$$\begin{aligned} e^*(\phi) &= \arg\max_{e \in \mathcal{E}_{\mathrm{eval}}} \mathrm{MRR}_{\mathrm{reflect}}(\phi, e), \\ m_{\mathrm{gate}}(\phi) &= \mathrm{MRR}_{\mathrm{gate}}(\phi, e^*(\phi)), \end{aligned} \tag{4}$$

where e is a training epoch, $\mathcal{E}_{\mathrm{eval}}$ contains the scheduled evaluation epochs, and the two MRR functions measure performance at epoch e on their respective splits. The selected epoch is $e^*(\phi)$, and $m_{\mathrm{gate}}(\phi)$ is its MRR on the *gate* split. Skill-update decisions use this score without clipping or baseline normalization. An execution trace records check outcomes, failures, and measured training and ranking diagnostics.

3.3 Feedback-Driven Skill Refinement

Using evaluation traces τ_t from a proposal batch P_t and accumulated feedback, the SkillOpt optimizer (Yang et al., 2026) proposes bounded edits to the design Skill. Failed checks also provide feedback without GNN training. For each usable revision S'_t, the design agent generates a new proposal p'_t on the fixed gate task u_g, and the evaluator records a trace τ'_t while assessing its compiled component using Equation (4). Acceptance depends on this new component, not on the preceding proposal that supplied the feedback.

At outer step t, let (S_t, ϕ_t, j_t) denote the retained evaluated Skill snapshot, its program, and its score $j_t = m_{\mathrm{gate}}(\phi_t)$. Evaluation of a component generated with the initial Skill S_0 establishes the initial

Algorithm 1 SkillGNN component evolution

Input: KG $(\mathcal{V}, \mathcal{R}, \mathcal{T})$ with predefined training and validation splits, GNN with contract $\mathcal{C}$, component design agent $A = A_{\text{LLM}}$, initial Skill S_0 with valid initialization, fixed component-design instruction u_g, evolution-step budget T, fixed training and evaluation settings $\mathcal{P}$
Output: Component ϕ_T with the highest gate MRR and its design Skill S_T

$\underbrace{(\mathcal{G}^{\text{sub}}_{\text{train}}, \mathcal{T}_{\text{reflect}}, \mathcal{T}_{\text{gate}})}_{\mathcal{D}_{\text{eval}}} \leftarrow \text{SubgraphSampling}(\mathcal{V}, \mathcal{R}, \mathcal{T})$
$(\phi_0, j_0) \leftarrow \text{InitializeComponent}(S_0, \mathcal{C}, u_g, \mathcal{D}_{\text{eval}}, \mathcal{P})$ // generate and evaluate the first component
for $t = 0, \ldots, T-1$ **do**
  $\mathcal{U}_t \leftarrow \text{NextDesignInstruction}(t)$ // get the component-design instruction
  $P_t \leftarrow \{A(S_t, \mathcal{C}, u) : u \in \mathcal{U}_t\}$ // generate a component proposal
  $\tau_t \leftarrow \text{EvaluateForFeedback}(P_t, \mathcal{G}^{\text{sub}}_{\text{train}}, \mathcal{T}_{\text{reflect}}, \mathcal{P})$ // check, train, and return feedback
  $S'_t \leftarrow \text{SkillOpt}(S_t, \tau_t)$ // reflect and propose Skill edits
  $(S_{t+1}, \phi_{t+1}, j_{t+1}) \leftarrow (S_t, \phi_t, j_t)$ // retain the current state
  if $S'_t \neq \varnothing$ **then**
    $p'_t \leftarrow A(S'_t, \mathcal{C}, u_g)$ // generate a revised-Skill proposal
    $(\text{valid}, \phi'_t, j'_t, \tau'_t) \leftarrow \text{EvaluateForAcceptance}(p'_t, \mathcal{D}_{\text{eval}}, \mathcal{P})$
    if valid **and** $j'_t > j_t$ **then**
      $(S_{t+1}, \phi_{t+1}, j_{t+1}) \leftarrow (S'_t, \phi'_t, j'_t)$ // update the best component
    end if
  end if
  $\text{RecordStep}(t)$ // save proposals and outcomes
  $(S_{t+1}, \phi_{t+1}, j_{t+1}) \leftarrow \text{ScheduledSlowEdit}(t, S_{t+1}, \phi_{t+1}, j_{t+1})$
  $\text{ScheduledMetaUpdate}(t)$ // revise optimizer memory
end for
Lock (S_T, ϕ_T) for full-graph training and final evaluation
return (ϕ_T, S_T)

state. For a revision S'_t with compiled program ϕ'_t and score $j'_t = m_{\text{gate}}(\phi'_t)$, the update is

$$(S_{t+1}, \phi_{t+1}, j_{t+1}) = \begin{cases} (S'_t, \phi'_t, j'_t), & \text{if valid and } j'_t > j_t, \\ (S_t, \phi_t, j_t), & \text{otherwise,} \end{cases} \tag{5}$$

where validity requires the program to pass the eligibility checks and complete evaluation on the *gate* split. The retained state remains unchanged if the optimizer produces no usable edit or evaluation fails, and no score will be assigned for failed evaluations. Nevertheless, the feedback recorded during these attempts can still inform subsequent component proposals and Skill edits.

Specifically, Algorithm 1 executes T regular SkillOpt steps. Throughout this process, the contract, LLM configuration, subgraph, and training protocol remain fixed. Scheduled slow edits follow the same acceptance rule, while meta updates revise optimizer memory (Appendix B). We lock the generated component with the highest MRR on the *gate* split and its corresponding design Skill. Full-graph training then proceeds from scratch using this selected program, with the best checkpoint chosen via full-graph validation for official testing.

4 Experiments

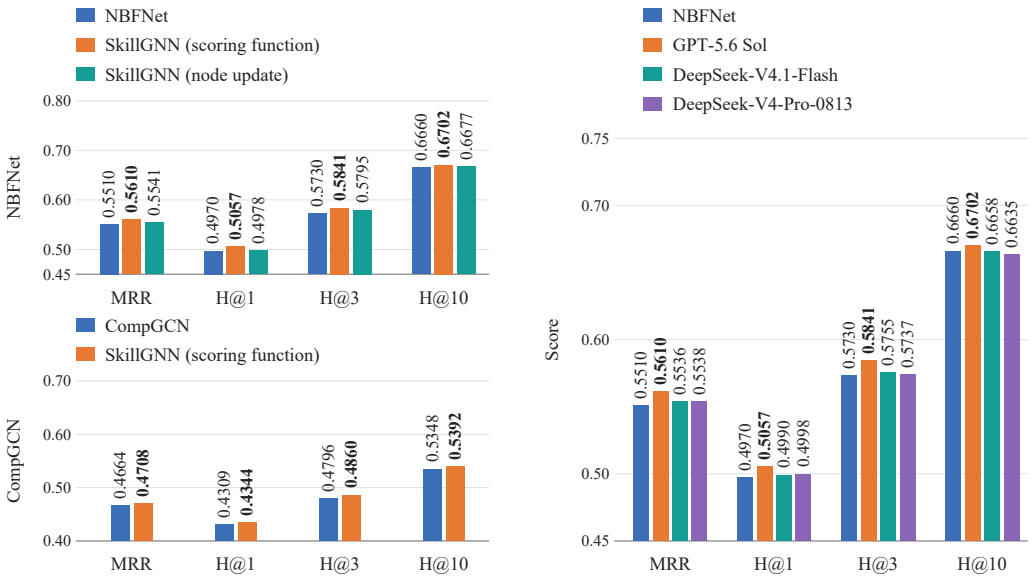
4.1 Datasets and Experimental Settings

Datasets. We evaluate SkillGNN on the standard WN18RR and FB15K-237 splits (Dettmers et al., 2018; Toutanova & Chen, 2015), with 40,943/14,541 entities, 11/237 relations, and 86,835/272,115 training triples, respectively. Evolution uses fixed subgraphs of 7,441/36,000 training triples while retaining all training relations (Appendix A).

Implementation and training settings. NBFNet-PyG uses six propagation layers with 32-dimensional states, combining DistMult messages with PNA aggregation (Corso et al., 2020). Within a 100-step evolution budget, we train each candidate for 20 epochs using Adam (Kingma & Ba, 2015) at learning rate 0.005. Checkpoints for *gate* scoring are selected by reflect MRR measured every two epochs. Full-graph checkpoints are selected by validation MRR (Appendix A).

Table 2: Full-graph link prediction results. Literature results retain their reported precision, whereas our results use four decimals. Bold marks column maxima, and "-" denotes an unreported result.

Category	Model / method	WN18RR				FB15K-237			
		MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10
Manually designed scoring functions	TransE (Bordes et al., 2013)	0.228	0.053	0.368	0.520	0.313	0.221	0.347	0.497
	RotatE (Sun et al., 2019)	0.478	0.439	0.494	0.553	0.333	0.240	0.368	0.522
	LineaRE (Peng & Zhang, 2020)	0.495	0.453	0.509	0.578	0.357	0.264	0.391	0.545
	HAKE (Zhang et al., 2020b)	0.497	0.452	0.516	0.582	0.346	0.250	0.381	0.542
	RESCAL (Nickel et al., 2011)	0.467	0.439	0.480	0.517	0.356	0.263	0.393	0.541
	DistMult (Yang et al., 2015)	0.452	0.413	0.466	0.530	0.343	0.250	0.378	0.531
	ComplEx (Trouillon et al., 2016)	0.475	0.438	0.490	0.547	0.348	0.253	0.384	0.536
	TuckER (Balazevic et al., 2019)	0.470	0.443	0.482	0.526	0.358	0.266	0.394	0.544
Neural networks and GNNs	ConvE (Dettmers et al., 2018)	0.442	0.411	0.451	0.504	0.339	0.248	0.369	0.521
	R-GCN (Schlichtkrull et al., 2018)	0.402	0.345	0.437	0.494	0.273	0.182	0.303	0.456
	CompGCN (Vashishth et al., 2020)	0.479	0.443	0.494	0.546	0.355	0.264	0.390	0.535
	NBFNet (Zhu et al., 2021)	0.551	0.497	0.573	0.666	0.415	0.321	0.454	0.599
Automated scoring-function search	AutoSF (AutoBLM) (Zhang et al., 2020a)	0.490	0.451	-	0.567	0.360	0.267	-	0.552
	AutoBLM+ (Zhang et al., 2023)	0.492	0.452	-	0.567	0.364	0.270	-	0.553
LLM-driven component design	One-shot LLM design (baseline)	0.5506	0.4959	0.5747	0.6599	0.4137	0.3199	0.4549	0.5955
	SkillGNN (ours)	**0.5610**	**0.5057**	**0.5841**	**0.6702**	**0.4176**	**0.3241**	**0.4572**	**0.6003**



(a) Backbones and components

(b) Design and optimization LLMs (NBFNet)

Figure 2: WN18RR full-graph results across components and design LLMs. NBFNet values follow published results (Zhu et al., 2021). Our CompGCN reproduction holds the backbone and core training settings fixed under a shared ranking protocol. Node-update metrics use the run with highest test MRR. Each LLM independently evolves a scorer before NBFNet training from scratch.

Additionally, we report full-entity filtered MRR and H@1/3/10 for both prediction directions (Appendix C).

4.2 Full-Graph Link Prediction

Table 2 compares SkillGNN with one-shot LLM design, which uses the same NBFNet scoring interface without feedback or iteration. TransE, RotatE, RESCAL, DistMult, ComplEx, and ConvE follow LibKGE (Ruffinelli et al., 2020), while R-GCN and NBFNet follow Zhu et al. (2021). Other baselines use their original reports, including AutoSF (AutoBLM) and its extension AutoBLM+.

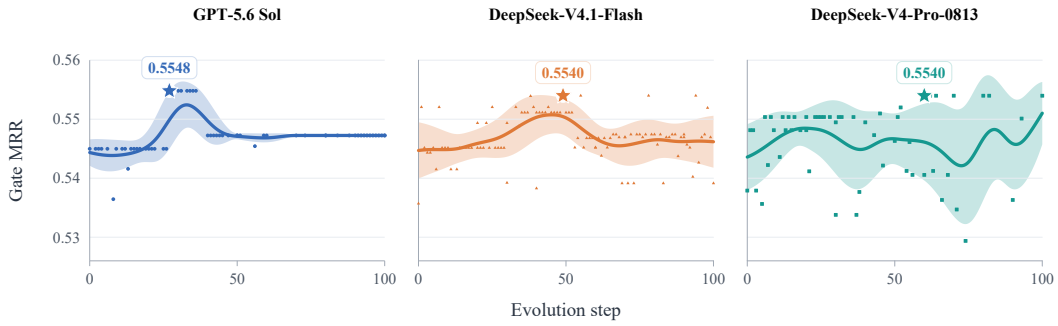


Figure 3: WN18RR gate evaluations. Triweight rolling means (± 15 steps) summarize gate MRR. Bands show local variation (± 1 weighted rolling standard deviation), and stars mark each run's best measured component.

SkillGNN achieves the highest values in all eight metric columns. Its MRR gains over published NBFNet are 0.0100 on WN18RR and 0.0026 on FB15K-237. Against one-shot design, the corresponding gains are 0.0104 and 0.0039.

4.3 Backbones, Components, and Design LLMs

For NBFNet, evolving the node update (a message-passing component) gives 0.5541 MRR, while evolving the scorer gives 0.5610 (Figure 2(a)). Node-update evolution retains the original scorer alongside the message and aggregation functions. CompGCN's normalized residual scorer reaches 0.4708 MRR, compared with 0.4664 for our reproduced baseline, while retaining the circular-correlation encoder and ConvE query transformation.

With the NBFNet scoring interface fixed, scorers from all three design LLMs exceed published NBFNet in MRR (Figure 2(b)). GPT-5.6 Sol has the highest value on each metric.

Gate MRR fluctuates as new components are proposed (Figure 3). GPT-5.6 Sol reaches 0.5548 before later proposals return to a lower range. Both DeepSeek runs reach 0.5540, with greater variation for Pro. The acceptance rule retains each run's highest-scoring component despite this variation.

4.4 Evolved Component Case Studies

The selected WN18RR and FB15K-237 scorers augment the original MLP f_0 with a shared mixture branch R_{mix} (Appendix D) and a dataset-specific auxiliary branch. For relation representation q and candidate state x, the scores are

$$s(q,x) = f_0([x;q]) + \tanh(a_1)R_{\text{mix}}(q,x) + \tanh(a_2)R_{\text{aux}}(q,x), \tag{6}$$

$$R^{\text{WN}}_{\text{aux}}(q,x) = 0.08\langle \text{Rot}(q,\psi(q)) \odot d(q), x\rangle,$$

$$R^{\text{FB}}_{\text{aux}}(q,x) = 0.10\langle q \odot d_1(q) + \text{Rot}(q,\psi(q)) \odot d_2(q), x\rangle. \tag{7}$$

The gates a_1, a_2 start at zero, so the initial score reduces to $f_0([x;q])$. Here, Rot applies planar rotations with angles $\psi(q) = \pi\tanh(W_\psi q + b_\psi)$, and d, d_1, d_2 are trainable affine maps. WN18RR uses rotated relation features, whereas FB15K-237 also includes the original features. Both components leave propagation unchanged.

4.5 Ablation Studies

The selected WN18RR scorer exceeds the first evaluated and one-shot components by 0.0027 and 0.0104 MRR, respectively (Table 3). H@1 and H@3 follow the same order, while the first component has the highest H@10.

Table 3: WN18RR full-graph test ablations. The reference repeats Table 2. Variants are trained separately, with checkpoints selected by MRR on the full validation set. Bold marks column maxima, and w/o denotes without. Phase-output bypass retains branch parameters and initialization.

Ablation type	Setting	MRR	H@1	H@3	H@10
—	Best component	**0.5610**	**0.5057**	**0.5841**	0.6702
Design process	First component (initial Skill)	0.5583	0.5041	0.5788	**0.6709**
	One-shot LLM design (w/o evolution feedback)	0.5506	0.4959	0.5747	0.6599
Component structure	Best component w/o phase output	0.5590	0.5051	0.5799	0.6693
	Best component w/o $x \odot q$ channel	0.5552	0.4979	0.5795	0.6701
	Best component with gates initialized at 1	0.5577	0.5034	0.5809	0.6683

Table 4: Triple classification accuracy with literature-reported AutoSF results (Zhang et al., 2020a). Bold marks the best accuracy, and "-" means unreported. Coverage is the fraction of test triples with validation-selected thresholds.

Model	WN18RR		FB15K-237	
	Accuracy	Test coverage	Accuracy	Test coverage
TransE	0.8815	100%	0.8256	99.9365%
RotatE	0.8671	100%	0.8243	99.9365%
AutoSF	0.8770	-	0.8120	-
NBFNet	0.9129	100%	0.8634	99.9365%
SkillGNN	**0.9135**	100%	**0.8643**	99.9365%

The WN18RR scorer described in Section 4.4 provides the reference for the structural ablations, with an MRR of 0.5610. Zeroing only the $x \odot q$ input to the interaction MLP within R_{mix} (Appendix D) gives 0.5552 MRR, the largest difference from the reference. With the auxiliary output $R^{\text{WN}}_{\text{aux}}$ bypassed, MRR is 0.5590. Changing the initial values of a_1, a_2 from 0 to 1 gives 0.5577 MRR. The reference scorer exceeds all three structural variants on each metric.

4.6 Triple Classification

We complement entity ranking with triple classification. With validation-selected relation thresholds, SkillGNN's accuracy exceeds NBFNet's by 0.0006 on WN18RR and 0.0009 on FB15K-237 at identical coverage (Table 4 and Appendix E).

4.7 Discussion and Limitations

SkillGNN can continue component evolution across runs by reusing its evaluator and Skills. The Skills retain design knowledge and can be refined with new feedback. Adapting the evaluator to new component interfaces could extend evolution to other GNNs and machine learning models. We also acknowledge that SkillOpt does not guarantee convergence and that generated components have limited interpretability. Repeated LLM calls may incur substantial token costs.

5 Conclusion

We presented SkillGNN, which evolves GNN components through reusable textual Skills. Its evaluator provides feedback from program checks and subgraph training for SkillOpt to refine these Skills. Experiments on WN18RR and FB15K-237 showed improved link prediction over NBFNet and one-shot LLM design. Further experiments examined different design LLMs and extended evolution to NBFNet node-update and CompGCN scoring components. These findings provide a methodological basis for LLM-guided component self-evolution in GNNs and other model architectures. Future work will investigate long-term Skill reuse across tasks and extend evolution to additional component interfaces.

STATEMENT OF AI USE

We used LLM-based agents to design and evolve GNN components in the experiments. LLMs also assisted with language polishing of the manuscript.

REPRODUCIBILITY STATEMENT

We use the publicly available WN18RR and FB15K-237 datasets. Data preparation and experimental settings are described in the appendix. The source code will be made available at https://anonymous.4open.science/r/SkillGNN-7621.

REFERENCES

Ivana Balazevic, Carl Allen, and Timothy Hospedales. TuckER: Tensor factorization for knowledge graph completion. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 5185–5194. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1522. URL https://aclanthology.org/D19-1522/.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*, volume 26, 2013. URL https://papers.neurips.cc/paper_files/paper/2013/hash/1cecc7a77928ca8133fa24680a88d2f9-Abstract.html.

Haibo Chen, Xin Wang, Zeyang Zhang, Haoyang Li, Ling Feng, and Wenwu Zhu. AutoGFM: Automated graph foundation model with adaptive architecture customization. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 9295–9315. PMLR, 2025. URL https://proceedings.mlr.press/v267/chen25bp.html.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. In *Advances in Neural Information Processing Systems*, volume 33, pp. 13260–13271, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/99cad265a1768cc2dd013f0e740300ae-Abstract.html.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2D knowledge graph embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. doi: 10.1609/aaai.v32i1.11573. URL https://ojs.aaai.org/index.php/AAAI/article/view/11573.

Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. Graph neural architecture search. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pp. 1403–1409, 2020. doi: 10.24963/ijcai.2020/195. URL https://www.ijcai.org/Proceedings/2020/195.

Kaiqi Gong, Xiao Song, Songsong Liu, and Yong Li. EXP-GNN: Towards expressive propagation function design for graph neural network-based knowledge graph completion. *Expert Systems with Applications*, 299:129957, 2026. doi: 10.1016/j.eswa.2025.129957. URL https://doi.org/10.1016/j.eswa.2025.129957.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. URL https://arxiv.org/abs/1412.6980.

Fei Liu, Tong Xialiang, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32201–32223. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24bs.html.

Yuxuan Liu, Zhaochen Su, Yuhao Zhang, Jiahe Guo, Zhongwei Xie, Huihao Jing, Lingyun Xie, Qing Zong, Yauwai Yim, Zhixiong Zhang, Haoran Li, and Yangqiu Song. Rethinking self-evolving agent skills: Feedback dynamics over multiple rounds. *arXiv preprint arXiv:2608.02636*, 2026. URL https://arxiv.org/abs/2608.02636.

Zeyao Ma, Jing Zhang, Xiaokang Zhang, Jiaxi Yang, Zongmeng Zhang, Jiajun Zhang, Yuheng Jing, Lei Zhang, Hao Zheng, Wenting Zhao, Junyang Lin, and Binyuan Hui. Scaling agentic verifier for competitive coding. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR, 2026. URL https://arxiv.org/abs/2602.04254.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on Machine Learning*, pp. 809–816. Omnipress, 2011. URL https://icml.cc/2011/papers/438_icmlpaper.pdf.

Yanhui Peng and Jing Zhang. LineaRE: Simple but powerful knowledge graph embedding for link prediction. In *2020 IEEE International Conference on Data Mining (ICDM)*, 2020. doi: 10.1109/ICDM50108.2020.00051. URL https://arxiv.org/abs/2004.10037.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024. doi: 10.1038/s41586-023-06924-6. URL https://www.nature.com/articles/s41586-023-06924-6.

Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You CAN teach an old dog new tricks! on training knowledge graph embeddings. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BkxSmlBFvr.

Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web*, volume 10843 of *Lecture Notes in Computer Science*, pp. 593–607. Springer, 2018. doi: 10.1007/978-3-319-93417-4_38. URL https://link.springer.com/chapter/10.1007/978-3-319-93417-4_38.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://papers.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

KaShun Shum, Binyuan Hui, Jiawei Chen, Lei Zhang, X. W., Jiaxi Yang, Yuzhen Huang, Junyang Lin, and Junxian He. SWE-RM: Execution-free feedback for software engineering agents. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=H9wMe1G76j.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. RotatE: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HkgEQnRqYQ.

Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pp. 57–66. Association for Computational Linguistics, 2015. doi: 10.18653/v1/W15-4007. URL https://aclanthology.org/W15-4007/.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Eric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 2071–2080. PMLR, 2016. URL https://proceedings.mlr.press/v48/trouillon16.html.

Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1911.03082.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *International Conference on Learning Representations*, 2015. URL https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/ICLR2015_updated.pdf.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. SkillOpt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026. doi: 10.48550/arXiv.2605.23904. URL https://arxiv.org/abs/2605.23904.

Yongqi Zhang, Quanming Yao, Wenyuan Dai, and Lei Chen. AutoSF: Searching scoring functions for knowledge graph embedding. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 2020a. URL https://arxiv.org/abs/1904.11682.

Yongqi Zhang, Quanming Yao, and James T. Kwok. Bilinear scoring function search for knowledge graph learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45 (2):1458–1473, 2023. doi: 10.1109/TPAMI.2022.3157321. URL https://doi.org/10.1109/TPAMI.2022.3157321.

Zhanqiu Zhang, Jianyu Cai, Yongdong Zhang, and Jie Wang. Learning hierarchy-aware knowledge graph embeddings for link prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3065–3072, 2020b. doi: 10.1609/aaai.v34i03.5701. URL https://ojs.aaai.org/index.php/AAAI/article/view/5701.

Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural Bellman-Ford networks: A general graph neural network framework for link prediction. In *Advances in Neural Information Processing Systems*, volume 34, pp. 29476–29490, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/f6a673f09493afcd8b129a0bcf1cd5bc-Abstract.html.

A SUBGRAPH CONSTRUCTION

A.1 GRAPH STATISTICS

Table 5 summarizes the full datasets and the fixed subgraphs used for component evolution. The subgraph validation triples are drawn from the official validation split and partitioned into reflect, gate, and an audit reserve that is set aside for future research and left unused in this study.

Table 5: Full-dataset and selected-subgraph statistics. Counts exclude added inverse edges. Subgraph counts are taken from the frozen construction records. A dash denotes an inapplicable split.

	WN18RR		FB15K-237	
Statistic	Full graph	Subgraph	Full graph	Subgraph
Entities	40,943	4,000	14,541	4,102
Relations	11	11	237	237
Training triples	86,835	7,441	272,115	36,000
Validation triples (total)	3,034	1,000	17,535	2,000
reflect	-	495	-	993
gate	-	252	-	504
audit	-	253	-	503
Test triples	3,134	-	20,466	-

A.2 SUBGRAPH SAMPLING

For WN18RR, we sample 1,000 validation queries by relation and endpoint degree in the training graph, favoring shared endpoints. Starting from these endpoints, we add entities from their two-hop training neighborhoods, prioritizing entities with more connections to the endpoints and using higher local degree to break ties. We cap the subgraph at 4,000 entities and retain at least one training edge per entity, yielding 7,441 training triples (Table 5). Validation queries remain separate from training edges, and test data are excluded. The subgraph and its validation splits will be released in the code repository.

For FB15K-237, we compare three candidates from each of the dense, frequency-matched, and structure-aware sampling strategies. Each candidate contains 36,000 unique training facts. Frequency matching preserves required support edges while matching full-graph relation quotas. Structure-aware sampling extends frequency matching by prioritizing neighborhoods with multiple answers for the same relation and applying bounded edge swaps within each relation. Candidates with the same sample ID share validation anchors across sampling strategies.

A.3 STRUCTURAL COMPARISON

We use total variation to quantify distributional changes introduced by sampling. For distributions P and Q defined over the same categories, total variation is

$$\mathrm{TV}(P, Q) = \frac{1}{2} \sum_{k} |P_k - Q_k|. \tag{8}$$

The structural comparison contains eight terms.

1. **Training-relation frequency.** TV between relation frequencies in the full-graph and subgraph training facts.
2. **Validation-relation frequency.** TV between relation frequencies of eligible full-validation queries and selected queries.
3. **Endpoint degrees.** Mean head/tail degree-bin TV at the same selected query endpoints. Each graph's degree quantiles define its low, medium, and high bins.
4. **Two-hop support.** Mean absolute difference in directed and undirected two-hop support rates for the selected queries.

5. **Reverse-edge support.** Mean absolute difference in same-relation and any-relation reverse-edge support rates.
6. **Answer multiplicity.** Mean head/tail answer-count TV, using bins of 1, 2–4, and at least 5 training-supported answers.
7. **Relation cardinality scale.** Mean relative difference in query-weighted tails-per-head (tph) and heads-per-tail (hpt), using $|a - b| / \max(|a|, |b|)$ for each positive-valued pair.
8. **Relation type.** Mean of the per-relation type-change rate and the training-triple-weighted relation-type distribution TV after sampling. Types are one-to-one, one-to-many, many-to-one, and many-to-many, using the threshold 1.5 on tph and hpt.

Table 6 compares the evaluation subgraph with its full graph. Neighborhood statistics use the same selected queries in both graphs, with path support measured by existence rather than counts. Both subgraphs retain all training relations. WN18RR has a lower mean structural distance and preserves every relation's cardinality type. FB15K-237 matches relation frequencies more closely but shows larger changes in relation cardinality.

Table 6: Structural comparison between subgraphs and full graphs. TV denotes total variation. Mean structural distance averages the first eight discrepancies, with lower values indicating closer agreement.

Statistic	WN18RR	FB15K-237
Training-relation frequency TV	0.194811	0.062217
Validation-relation frequency TV	0.044669	0.026765
Mean degree-bin TV	0.155000	0.092500
Mean two-hop support gap	0.008000	0.000000
Mean reverse-edge support gap	0.000000	0.000000
Mean answer-count-bin TV	0.074000	0.144250
Mean relative tph/hpt gap	0.183447	0.685381
Relation-type discrepancy	0.091318	0.230431
Mean structural distance D_{struct}	0.093906	0.155193
Training-relation coverage	100%	100%
Per-relation type retention	100%	72.15%

A.4 FB15K-237 Subgraph Selection

We evaluate candidate subgraphs containing 36,000 unique training facts and all 237 relations. Their validation endpoints must have training support, and their splits must be disjoint. For each candidate, we train NBFNet for 20 epochs and TransE for at most 5,000 steps or 600 seconds, whichever comes first. Both models select checkpoints on reflect before gate evaluation.

Among candidates whose NBFNet gate MRR exceeds TransE's, we minimize the fixed selection score

$$S = 0.6 D_{\mathrm{struct}} + 0.4 \left| m^{\mathrm{reflect}}_{\mathrm{NBFNet}} - 0.364199 \right|. \tag{9}$$

The second term uses the full-graph NBFNet validation MRR, 0.364199, as its reference. For candidates within 0.001 of the minimum score, we prefer the one with fewer entities.

The selected dense subgraph (Table 7) has an absolute difference of 0.028294 between its NBFNet reflect MRR and the full-validation reference, with 72.1519% per-relation type retention. The gate MRR comparison determines which candidates are ranked by the selection score.

A.5 Evaluation Splits

During evolution, reflect selects subgraph checkpoints and supplies design feedback, while gate determines Skill-update acceptance and component selection. The *audit* reserves of 253 WN18RR and 503 FB15K-237 validation queries remain unused. Full validation selects full-graph checkpoints and supplies the reference in Equation (9). Official test data are excluded from search and used only to evaluate frozen models.

Table 7: Evaluation of FB15K-237 candidate subgraphs. Sample IDs distinguish the three subgraphs generated by each strategy. Each candidate has 36,000 training facts and 237 relations. Model performance is reported as MRR. Lower D and S are better.

Strategy	Sample	D	NBFNet reflect	NBFNet gate	TransE gate	S
Dense (locked)	1	0.15519	0.335905	0.339238	0.292328	**0.104433**
Dense	2	0.15491	0.318788	0.330116	0.309020	0.111113
Dense	3	0.15298	0.327348	0.334476	0.311158	0.106531
Frequency	1	0.17959	0.309186	0.320152	0.282923	0.129759
Frequency	2	0.17771	0.308936	0.312846	0.274437	0.128732
Frequency	3	0.17666	0.308626	0.315780	0.288496	0.128228
Structure	1	0.15516	0.305506	0.329652	0.273900	0.116572
Structure	2	0.15598	0.306614	0.316130	0.278011	0.116622
Structure	3	0.15609	0.311621	0.308475	0.293364	0.114684

B Experimental Protocol

B.1 NBFNet Configuration

We use the author-maintained NBFNet-PyG implementation and its dataset-specific settings, with single-process batch sizes of 16 for WN18RR and 64 for FB15K-237. Run records retain the source version, process count, precision, training budget, and evaluation schedule.

B.2 Shared Learning Objective

We keep the learning objective used in the NBFNet implementation (Zhu et al., 2021) fixed when changing components, including self-adversarial negative weighting (Sun et al., 2019). For a positive score s^+ and K strict-negative scores s_j^-, the loss is

$$\begin{aligned} \mathcal{L} &= \frac{1}{2}\left[-\log\sigma(s^+) - \sum_{j=1}^{K} w_j \log\sigma(-s_j^-)\right], \\ w_j &= \mathrm{stopgrad}\left(\frac{\exp(s_j^-/T)}{\sum_{\ell=1}^{K}\exp(s_\ell^-/T)}\right). \end{aligned} \tag{10}$$

Here T is the dataset-specific temperature. The weights for negative examples are detached from gradient computation, and the positive and negative terms receive equal total weight. The strict-negative sampler is unchanged across components.

B.3 SkillOpt Configuration

For NBFNet scoring, each 100-step WN18RR run consists of ten SkillOpt epochs, each with twenty design tasks processed in pairs. These evolution steps are distinct from the twenty training epochs used to evaluate each valid GNN component. Slow and meta updates take place in addition to the regular steps.

Each regular update allows three operations chosen from adding, deleting, or replacing Skill content. Design tasks cover symmetry, asymmetry, inversion, directionality, composition, transitivity, hierarchy, multi-answer behavior, and overall performance. The design and optimization LLMs remain fixed within each run.

C Link Prediction Evaluation

Let r_q denote the target rank for each of N head or tail queries. We compute

$$\mathrm{MRR} = \frac{1}{N}\sum_q \frac{1}{r_q}, \qquad \mathrm{MR} = \frac{1}{N}\sum_q r_q, \qquad \mathrm{H@}\,k = \frac{1}{N}\sum_q \mathbf{1}[r_q \le k]. \tag{11}$$

where $\mathbf{1}[\cdot]$ denotes the indicator function, which equals 1 if the condition holds and 0 otherwise.

For each query, we rank all entities in the corresponding graph with pessimistic tie handling. Head and tail queries receive equal weight. Filtering removes other known correct entities while retaining the target.

For reflect evaluation, filtering uses facts from the subgraph training and reflect splits. For gate evaluation, filtering also includes gate facts. Full-validation filtering uses the full training and validation splits, whereas test filtering uses all official splits. These rules apply to both original and evolved components. We retain per-query ranks and full-precision metrics. Lower MR and higher MRR/Hits indicate better performance. Table 8 reports the MR results.

Table 8: Literature-reported and SkillGNN's mean rank. Results for R-GCN and NBFNet are from Zhu et al. (2021), while CompGCN results are from Vashishth et al. (2020).

Method	WN18RR	FB15K-237
R-GCN	2719	221
CompGCN	3533	197
NBFNet	636	114
SkillGNN	598	114

D Exact Expression of the Locked WN18RR Component

Let $\mathrm{n}(z) = z/\max(\|z\|_2, \epsilon)$, with the implementation's fixed numerical-stability constant ϵ. Given query representation q and final candidate state x, the program first computes

$$\begin{aligned} &c = W_c q + b_c, \quad \eta = \mathrm{softplus}(W_\eta q + b_\eta) + 10^{-6}, \quad b = W_b q + b_b, \\ &u = q \odot c, \qquad\qquad v = q \odot \eta + b. \end{aligned} \tag{12}$$

The base branch f_0 is a two-layer ReLU MLP with input $[x; q]$ and hidden width 64. The mixed branch models interactions with a two-layer GELU MLP f_1 of hidden width 64,

$$\begin{aligned} R_{\mathrm{mix}}(q, x) = {} & 0.40 f_1([x - q;\ x \odot q;\ \mathrm{n}(u);\ x - v]) \\ & + 0.25\langle \mathrm{n}(u), \mathrm{n}(x)\rangle \\ & + 0.20\langle \mathrm{n}(u \odot x), \mathrm{n}(v)\rangle \\ & - 0.15\|\mathrm{n}(v) - \mathrm{n}(x)\|_2. \end{aligned} \tag{13}$$

For the phase branch, independent query mappings control the rotation angles and feature weights,

$$\begin{aligned} \psi = \pi \tanh(W_\psi q + b_\psi), \qquad d = W_d q + b_d, \\ R_{\mathrm{phase}}(q, x) = 0.08\langle \mathrm{Rot}(q, \psi) \odot d, x\rangle. \end{aligned} \tag{14}$$

Here, rotation treats the two halves of an even-dimensional real vector as real and imaginary coordinates and applies coordinate-wise planar rotations. Two zero-initialized scalar gates combine these branches in Equation (6). The selected program fixes the coefficients, while graph-model training learns the query mappings and MLP parameters along with the gates.

E Triple Classification Protocol

Classification reuses fixed triples labeled positive (1) or negative (-1). WN18RR contains 6,068 validation and 6,268 test examples, balanced by label and covering all eleven relations. Before evaluation, we map entity and relation IDs to those used by the frozen model. This mapping preserves every supplied example without resampling or filtering.

We score each triple on the original training graph using its raw logit, without inverse-query averaging. For relation r, let the validation-score range be $[a_r, b_r]$. Candidate thresholds are $a_r + i \times 0.001$ for integers i from zero through $\lfloor (b_r - a_r)/0.001 \rfloor$. We choose the threshold with the highest validation accuracy and use the lower value to break ties. Triples with scores at or above the selected threshold are classified as positive.

Table 9: Recorded LLM usage during evolution. Tokens are in millions (M). Calls include component design and Skill optimization.

Dataset	Component	Design LLM	Steps	Calls	Input (M)	Output (M)
WN18RR	NBFNet scoring component	GPT-5.6 Sol	100	536	18.415	0.541
	NBFNet scoring component	DeepSeek-V4.1-Flash	100	640	18.646	5.757
	NBFNet scoring component	DeepSeek-V4-Pro-0813	100	470	14.193	3.498
	NBFNet node-update component	GPT-5.6 Sol	100	425	6.282	0.116
	CompGCN scoring component	GPT-5.6 Sol	100	418	6.629	0.233
FB15K-237	NBFNet scoring component	GPT-5.6 Sol	40	111	3.907	0.109

A test triple is evaluated only when its relation has a validation threshold. Coverage is the fraction of supplied test examples that meet this condition. All WN18RR test relations have a threshold. NBFNet and SkillGNN share this procedure without training an additional classifier.

F Implementation Details

F.1 CompGCN Implementation

CompGCN uses the authors' corr+ConvE implementation with one graph-convolution layer. Initial entity embeddings have dimension 100, while graph outputs and queries use dimension 200. We train in batches of 128 using Adam at learning rate 0.001 with label smoothing 0.1. Subgraph evaluations use 100 epochs, while the full-graph budget permits up to 500 epochs.

To keep the authors' source code unchanged, a shared adapter handles legacy FFT and function interfaces. We check circular correlation, forward outputs, gradients, and checkpoint reloads through this adapter. Subgraph runs use only training facts and the reflect/gate splits.

F.2 Resource Usage

The LLM usage in Table 9 covers five 100-step WN18RR runs and the first 40 evolution steps on FB15K-237. We aggregate available LLM usage records, counting each call once. Missing usage is not estimated, so incomplete totals remain lower bounds. For NBFNet scorer evolution with GPT-5.6 Sol, recorded GPU time totals 19.75 hours over 100 steps on WN18RR and 58.29 hours over the first 40 steps on FB15K-237. These totals sum the durations of subgraph jobs, counting cached evaluations once. They measure cumulative GPU time rather than end-to-end search time.