Towards smaller language models via layer looping

Sabri Eyuboglu [1] **Dylan Zinsley** [2] **Jon Saad-Falcon** [1] **Simran Arora** [1] **Atri Rudra** [2] **James Zou** [1] **Chris Ré** [1]

Abstract

Language models store a huge amount of knowledge in their parameters. Typically, these parameters are organized as a deep stack of dense layers. This dominant architecture bears little resemblance to the implementations of optimized data stores (*e.g.* a database management system like PostgreSQL), which begs the question: *are there other architectures that can store and query the same information more efficiently?* In this work, we explore two simple modifications to the standard architecture: *looping* — sharing parameters across layers — and *mixture-of-experts* (MoE). We compare the space complexity of standard and *looped-moe* models on a simple task where the model must memorize a knowledge graph (KG) during training and answer multi-hop queries over it at inference time. We prove that the looped-moe model can store a KG of size T and answer q-hop queries with $\mathcal{O}(T)$ parameters. In contrast, we argue that the size of a standard model must scale with the complexity of the queries q, providing a lower bound of $\Omega(qT)$ parameters for a restricted class of standard models. We confirm this scaling with experiments on synthetic KGs, finding that looped-conditional models can reliably answer four-hop queries over KGs that are $9\times$ larger than parameter-matched standard models can.

1. Introduction

Modern language models store an enormous amount of information in their parameters, from details on rulings from specific US jurisdictions (Guha et al., 2024) to function signatures of arcane software libraries (Nijkamp et al., 2022). Increasing the parameter count reliably improves the breadth of a model's knowledge (Kaplan et al., 2020), so the strongest models have billions parameters organized as a deep stack of dense layers (Touvron et al., 2023; Team et al., 2024; Workshop et al., 2022; Brown et al., 2020). However, this *standard architecture* (defined in Section 2) eschews many basic principles that enable traditional data stores to process queries with limited memory consumption. *Are there alternatives that use fewer parameters to store and query the same information?*

In this work, we study the space complexity with which different architectures can store and query knowledge in their parameters. To facilitate the evaluation, we compare architectures on a simplified task we call *knowledge graph learning*: each model is trained to memorize a knowledge graph (KG) (Vrandečić & Krötzsch, 2014; Suchanek et al., 2007) and answer multi-hop queries over it. In this setting, we can carefully control the amount of information stored in the model and the complexity of those queries.

We illustrate several limitations of the standard model architecture (defined in Section 2) when applied to the knowledge graph learning task. (1) In order to answer compositional queries (*e.g.* multi-hop QA), a standard model must store information redundantly. (2) Standard models access and move every single byte of stored information on every query (Brown et al., 2020). This means answering queries from one domain requires processing data from another (*e.g.* answering questions about the US constitution requires processing information about C++ syntax).

We show that two well-known but non-standard modifications can address these limitations.

1. *Looped layers* (Yang et al., 2023) (also referred to as *universal* (Dehghani et al., 2018), *weight-tied* (Lan et al., 2020; Xue et al., 2022), or *iterative* (Jaegle et al., 2021) models) reuse the same parameters at every layer, enabling them to answer complex queries without maintaining redundant copies of information.
2. *Mixture-of-expert layers* (Fedus et al., 2022; Jiang et al., 2024; Shazeer et al., 2017) activate different subsets of model parameters dependent on the input. This allows us to scale the parameter count with limited effect on FLOP count.

We combine these techniques into a single *looped-moe* model and prove it can answer q-hop queries over a knowledge graph with T triples using only $\mathcal{O}(T)$ parameters

*Equal contribution [1]Department of Computer Science, Stanford University [2]University at Buffalo, The State University of New York. Correspondence to: Sabri Eyuboglu <eyuboglu@stanford.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

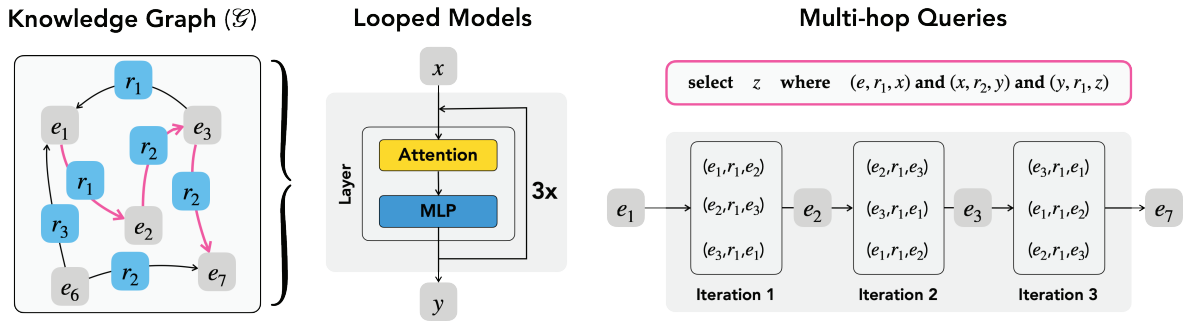

Figure 1. **Overview of knowledge graph learning with looped models.** *(Left)* A toy knowledge graph with entities, e_i, connected to other entities, e_j, via relations r_j. In knowledge graph learning (see Section 4), a language model is trained to memorize a knowledge graph and answer queries over it. *(Center)* Schematic of a looped language model ($B = 1$ and $l = 3$) in which the same layer is repeatedly applied to the input (see Section 3). The triples from the knowledge graph are stored in the parameters of the layer. *(Right)* Looped models can answer multi-hop queries over the knowledge graph without storing information redundantly (see Appendix B).

(ignoring the dependency on vocab size c). In contrast, no upper bound better than $\mathcal{O}(Tq)$ parameters is known for standard models. Thus, looped-moe models effectively break the dependency between the complexity of the query and the requisite model size. In Section 5, we discuss these results in further detail.

In experiments, we measure the size of the largest knowledge graph over which a model can reliably answer queries. As we increase the number of parameters in the model, the rates of growth for both architectures are consistent with the rates predicted by our theoretical analysis in Section 5. Strikingly, we find that looped-moe models can perfectly answer four hop queries (q=4) over knowledge graphs that are up to $9\times$ larger than parameter-matched standard models can.

In summary, our contributions are:

- We formalize the problem of knowledge graph learning to facilitate the evaluation of language model space complexity.
- We provide the first known upper bounds on the space complexity of standard and looped-moe models in the context of knowledge graph learning.
- In experiments, we show that looped-moe models can internalize knowledge graphs $9\times$ larger than standard models can.

See Appendix A for a more detailed discussion of related work.

2. Preliminaries

In this section, we describe the *standard language model architecture*. Note that for the sake of brevity we omit some details including biases and residual connections below. For a full description of architectures, see Appendix B.2.

Language models with the *standard* architecture consist of L layers. Each layer is defined as the composition of a *sequence mixer* (*e.g.* attention (Vaswani et al., 2017)) and a *state mixer* (*e.g.* MLP). The ℓ^{th} layer takes as input $\boldsymbol{x}^\ell \in \mathbb{R}^{N\times d}$ of length N and dimension d, and computes output $\boldsymbol{y}^\ell \in \mathbb{R}^{N\times d}$. The layers are applied sequentially such that $\boldsymbol{y}^\ell = \boldsymbol{x}^{\ell+1}$.

In this work, we assume the sequence mixer is attention, parameterized by a projection matrix $\boldsymbol{W}^{\text{qkv}} \in \mathbb{R}^{d\times 3d}$. Attention first projects the input $\boldsymbol{x}$: $\boldsymbol{q} = \boldsymbol{x}\boldsymbol{W}^{\text{qkv}}[:, \mathbf{0} : \boldsymbol{d}]$, $\boldsymbol{k} = \boldsymbol{x}\boldsymbol{W}^{\text{qkv}}[:, \boldsymbol{d} : \mathbf{2}\boldsymbol{d}]$, $\boldsymbol{v} = \boldsymbol{x}\boldsymbol{W}^{\text{qkv}}[:, \mathbf{2}\boldsymbol{d} : \mathbf{3}\boldsymbol{d}]$. Then, the projected embeddings are aggregated along the sequence dimensions according to:

$$\text{Attn}(\boldsymbol{W}^{\text{qkv}}; \boldsymbol{x}) = \text{softmax}(\frac{1}{\sqrt{d}}\boldsymbol{q}\boldsymbol{k}^\top)\boldsymbol{v} \tag{1}$$

We also assume that the *state mixer* is an MLP variant called a *gated-linear unit* (Shazeer, 2020). This MLP, which includes an elementwise multiplication of the input with itself, is now used in most large language models (*e.g.* LLaMa (Touvron et al., 2023)). It is defined as

$$\text{MLP}(\mathbf{W}^{\text{in}}, \mathbf{W}^{\text{out}}; \boldsymbol{x}) = (\sigma(\boldsymbol{x}\boldsymbol{W}^{\text{in}}[:, : d'])\odot \boldsymbol{x}\mathbf{W}^{\text{in}}[:, d' :])\boldsymbol{W}^{\text{out}} \tag{2}$$

where σ is an element-wise non-linearity (*e.g.* ReLU), $\mathbf{W}^{\text{in}} \in \mathbb{R}^{d\times 2d'}$, and $\mathbf{W}^{\text{out}} \in \mathbb{R}^{d'\times d}$.

A *layer* is simply the composition of these two operations (with residual connections omitted for brevity):

$$\begin{aligned}&\text{Layer}(\boldsymbol{W}^{\text{qkv}}, \boldsymbol{W}^{\text{in}}, \boldsymbol{W}^{\text{out}}; \boldsymbol{x}) = \\ &\text{MLP}(\boldsymbol{W}^{\text{in}}, \boldsymbol{W}^{\text{out}}; \text{Attn}(\boldsymbol{W}^{\text{qkv}}; \boldsymbol{x}))\end{aligned} \tag{3}$$

In the standard architecture, different weights are used at each layer, so at the ℓ^{th} layer we compute $\boldsymbol{x}^{\ell+1} = \text{Layer}(\boldsymbol{Q}^\ell, \boldsymbol{K}^\ell, \boldsymbol{V}^\ell, \boldsymbol{W}^{\text{in},\ell}, \boldsymbol{W}^{\text{out},\ell}; \boldsymbol{x}^\ell)$.

3. Looped-MoE Models

In this section, we describe two familiar modifications to the standard architecture: looped layers and mixture-of-experts. We then combine them into a single *looped-moe* model. [1].

Looped layers In a standard language model, different weights are used at every layer. In a *looped* language model, a single block of B layers is repeatedly applied $\frac{L}{B}$ times. In other words, we only maintain B layers worth of weights and at layer l we use the weights from layer $l \bmod B$.

In the extreme case ($B = 1$), we maintain one set of weights $\boldsymbol{W}^{\text{qkv}}, \boldsymbol{W}^{\text{in}}, \boldsymbol{W}^{\text{out}}$, and reuse them at every layer:

$$\forall \ell \in [L] \qquad \boldsymbol{x}^{\ell+1} = \text{Layer}(\boldsymbol{W}^{\text{qkv}}, \boldsymbol{W}^{\text{in}}, \boldsymbol{W}^{\text{out}}; \boldsymbol{x}^{\ell}) \quad (4)$$

This is sometimes referred to as a *universal model* (Dehghani et al., 2019), *weight-tied* (Lan et al., 2020; Xue et al., 2022), or *iterative* (Jaegle et al., 2021) model in the literature.

Mixture-of-experts A mixture-of-experts (MoE) layer routes each input to a different subset of its parameters (Shazeer et al., 2017). This allows us to increase the number of parameters in a model without increasing the number of FLOPs required for inference. Below we define a simple MoE for the MLP. We use softmax and a single linear layer with weight $\boldsymbol{W}^{\text{switch}} \in \mathbb{R}^{d \times m}$ to route to k (typically $k = 2$) experts. Each token $\boldsymbol{x}[j]$ in the input is routed separately.

$$s[j, i] = \text{softmax}(\boldsymbol{x}[j]\boldsymbol{W}^{\text{switch}})[i] \quad (5)$$

$$\tilde{s}[i] = \begin{cases} s[i] & \text{if } s[i] \text{ in top } k \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

$$\text{MLP}_{\text{MoE}}(\{\boldsymbol{W}_i^{\text{in}}\}_{i=1}^k, \{\boldsymbol{W}_i^{\text{out}}\}_{i=1}^k, \boldsymbol{W}^{\text{switch}}; \boldsymbol{x})[j] = \sum_{i=1}^{m} \text{MLP}(\boldsymbol{W}_i^{\text{in}}, \boldsymbol{W}_i^{\text{out}}; \boldsymbol{x}[j]) \frac{\tilde{s}[j, i]}{\sum_{p=1}^{m} \tilde{s}[j, p]} \quad (7)$$

In a looped-moe model with $B = 1$, we use the same set of MoE weights at each layer.

$$\forall \ell \in [L] \qquad \boldsymbol{x}^{\ell+1} = \text{Layer}(\boldsymbol{W}^{\text{qkv}}, \{\boldsymbol{W}_i^{\text{in}}\}_{i=1}^k, \{\boldsymbol{W}_i^{\text{out}}\}_{i=1}^k, \boldsymbol{W}^{\text{switch}}; \boldsymbol{x}^{\ell}) \quad (8)$$

In our experiments, we use $B = 1$.

4. Knowledge Graph Learning

In this section, we formally define the problem of *knowledge graph learning*, the task of memorizing a knowledge graph and answering queries over it.

[1]Similar models also based on these modifications are also referred to as sparse universal transformers (Tan et al., 2023) and MoE universal transformers (Csordás et al., 2024) in the literature

Knowledge Graphs A knowledge graph $\mathcal{G}$ is a directed, labeled graph made up of a set of triples (e_1, r_1, e_2) that connect two *entities* e_i, e_j with a *relation* r_k. Each triple encodes a fact. For example, if e_1 represented `France` and r_1 the relation `capital` then e_2 would be the entity `Paris`. We denote the number of distinct entities with E, relations with R, and triples with T. [2]

Queries A *query* is a request for one or more entities which satisfy a predicate.[3] The predicate uses a combination *literal* entities and relations with query *variables*. For example, the query below requests all entities x for which there is a triplet (e_1, r_1, x):

$$\textbf{select} \quad x \quad \textbf{where} \quad \underbrace{(e_1, r_1, x)}_{\textbf{predicate}} \quad (9)$$

Continuing with the example above, this query is effectively asking: *what is the capital of France?*

In our theoretical analysis, we focus on *path queries* over *matching graphs*. A graph $\mathcal{G}$ is *matching* if relations are one-to-one. That is, for all entities e_i and relations r, there exists exactly one entity e_j for which a triple (e_i, r, e_j) exists and exactly one entity e_k for which a triple (e_k, r, e_j) exists. A *path query* requires composing one or more triples together. It consists of a starting entity e and a sequence relations r_1, r_2, r_3. The output is found by following the relations from start entity. We denote the length of the path with q. The path query below is of length $q = 2$.

$$\textbf{select} \quad y \quad \textbf{where} \quad \underbrace{(e, r_1, x) \textbf{ and } (x, r_2, y)}_{\textbf{predicate}} \quad (10)$$

For example, a length-two path query might ask: *what nationality is Chelsea's current manager?*

Encoding The input to the model is a sequence tokens representing the query. For example, the input to the model for the query in Equation (10) is shown below where each item in brackets is one token.

$$\textit{Input:} \texttt{[}e_1\texttt{]} \ \texttt{[}r_1\texttt{]} \ \texttt{[y]} \ \texttt{[and]} \ \texttt{[y]} \ \texttt{[}r_2\texttt{]} \ \texttt{[x]} \quad (11)$$

The input token sequence is *encoded* as a matrix $\boldsymbol{u} \in \mathbb{R}^{N \times d}$ where N is the length of the sequence and d is the hidden dimension of model. In our theoretical analysis, we assume each token $\boldsymbol{u}[i]$ in the vocabulary is represented by a unique binary code $\boldsymbol{u}[i] \in \{-1, 1\}^d$. Representing the full binary

[2]In practice, knowledge graphs (*e.g.* WikiData) are typically stored in a format specified by the resource description framework (RDF) (Klyne & Carroll, 2004)

[3]There are a few different DSLs, like SPARQL (Harris et al., 2013), that allow you to make queries over an RDF graph.

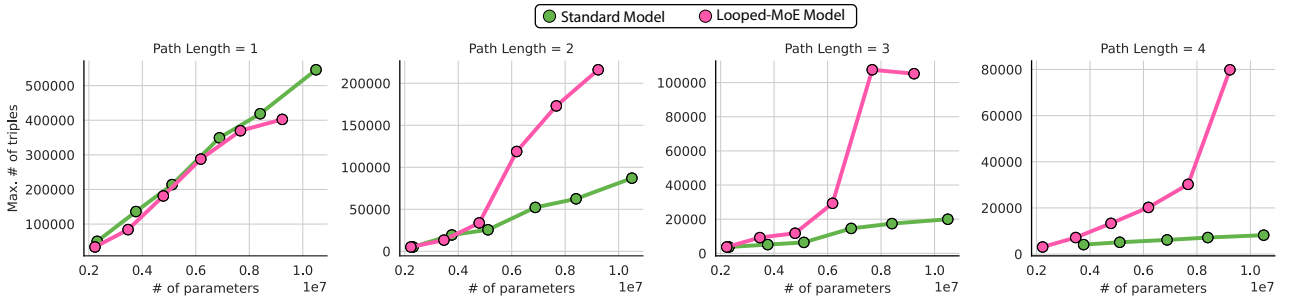

Figure 2. **Looped-MoE models exhibit improved parameter-efficiency for multi-hop path queries.** The x-axis shows the number of parameters in the model and the y-axis shows the size of the largest knowledge graph (measured in number of triples, see Section 4) over which the model can answer k-hop queries with accuracy $> 97\%$. Each plot shows results for a different number of hops k. Each hue represents a different model architecture. The figure illustrates how a standard model's capacity scales inversely with the length of the queries while the capacity of looped conditional models is affected less by the query length.

code for each token requires $d \geq \log_2 c$. If $d > \log_2 c$, the rest of embedding is padded with zeros. In our experiments, token embeddings are initialized randomly and learned.

The model transforms $\boldsymbol{u}$ into a matrix $\boldsymbol{y} \in \mathbb{R}^{N \times d}$. The output matrix can be *decoded* into a sequence of N tokens (*e.g.* via a nearest neighbor lookup). This sequence is then compared with a ground truth label sequence of the same length as the input. The labels corresponding to the input in Equation (22) are:

$$\textit{Labels:} \texttt{[-]} \quad \texttt{[-]} \quad \texttt{[-]} \quad \texttt{[-]} \quad \texttt{[-]} \quad \texttt{[-]} \quad [e_4] \tag{12}$$

When computing loss and accuracy, the labels `[-]` are ignored.

Training We assume that the model is trained via gradient descent on a sample of queries over a knowledge graph $\mathcal{G}$. We assume that every triple in the graph appears at least once in the training set.

Evaluation The model is then *evaluated* on queries over the **same** knowledge graph $\mathcal{G}$. The queries might be different, but the underlying triples are the same as those seen during training. So, we are testing whether or not the model has stored the triples in its parameters in such a way that they can be applied to answer the queries.

5. Bounds on Model Size for Knowledge Graph Learning

In this section, we theoretically analyze the space and time complexity that standard and looped-conditional models require to answer path queries over a matching graph.

Recall that T is the number of triples in the KG, q is length of the multi-hop query, and c is the vocab size. We begin with upper bounds for standard models.

Theorem 5.1. *For a matching graph $\mathcal{G}(E, R, T)$, a standard model with $\mathcal{O}(qT \log c + \log^2 c)$ parameters and $\mathcal{O}(q^2 T \log c + q \log^2 c)$ FLOPs can answer any path query $\mathcal{Q}$ with q hops over $\mathcal{G}$.*

For this upper bound, we show how to construct a model with $\mathcal{O}(q)$ layers and $\mathcal{O}(T)$ parameters per layer that can answer any query with q hops over a graph $\mathcal{G}$ with T triples. In this construction, we simply keep a separate copy of each triple at every layer. Interestingly, the amount of redundancy in this solution scales with the complexity of the queries q.

The natural question is whether this redundancy can be avoided. Proving a general lower bound for all standard models is challenging. Instead, we consider a restricted class of solutions based on constant-depth "shortcuts" like those studied by Liu et al. (2023). In this class of solutions, instead of storing each triple separately, we explicitly store $\tilde{q} - hop$ paths, enabling models with less than q layers. Intuitively, these strategies are unlikely to provide parameter reductions because there are exponentially-many ($R^{\tilde{q}}$) paths, leading to a blow-up in required layer width.

Theorem 5.2. *To answer all path queries $\mathcal{Q}$ with q hops over a matching graph $\mathcal{G}(E, R, T)$ with $R \geq e$, a standard model with $L \leq q$ layers requires $\Omega(qER)$ parameters.*

We conjecture that this redundancy is unavoidable for any standard model and reasonable values of R and q. Proving more general lower bounds for standard models is an important area for future work.

Next, we provide upper bounds for looped-moe models.

Theorem 5.3. *For a matching graph $\mathcal{G}(E, R, T)$, a looped model with $\mathcal{O}(T \log c + \log^2 c)$ parameters and $\mathcal{O}(q^2 \sqrt{T} \log c + q \log^2 c)$ FLOPs can answer any path query $\mathcal{Q}$ with q hops over $\mathcal{G}$.*

Here we show how to construct a looped-moe model consisting of a single MoE block with a total of $\mathcal{O}(T)$ parameters. In this construction, the triples are split among $\sqrt{T}$ experts

and at each iteration of the block, the router selects the expert containing the next triple in the path. Critically, this model requires only a single copy of each triple, avoiding the redundancy of the standard model's construction.

6. Knowledge Graph Learning Experiments

In this section, we assess whether the theoretical rates described above are consistent with the rates we observe in practice.

6.1. Synthetic Knowledge Graphs

First, we run experiments on synthetic KGs that measure the maximum number of triples T over which a model can answer queries with high accuracy.

Setup We compare two model architectures, a standard model and a looped-conditional model (see Section 2), at six model sizes (with model dimensions [64, 96, 128, 160, 192, 224]). To estimate, the maximum size knowledge graph each model can store, we use the following procedure:

1. **Generate** a synthetic matching knowledge graph $\mathcal{G}$ with $E = 128$ entities, $R = 8$ relations, and $T = 1{,}024$ triples.
2. **Train** the model on a sample of $128 \times R \times E$ training queries over $\mathcal{G}$.
3. **Evaluate** the model on a different (potentially overlapping) sample of $R \times E$ test queries over the same graph $\mathcal{G}$. If the accuracy of the model is above 97%, we increase the size of the knowledge graph by adding R new relations and continue training the model (move to step 2). Otherwise, we record the current size of the knowledge graph and exit.

The input and label tokens for the queries are generated as described in Section 4. For training, we use cross entropy loss and the Adam optimizer. We sweep over four log-spaced learning rates in the range $[10^{-4}, 10^{-2.5}]$ and report the maximum knowledge graph size achieved.

Figure 2 can be reproduced or extended to new architectures using the scripts provided at `https://github.com/<blinded-url>`.

Results In these experiments, we measure how the size of the knowledge graph that the model can store scales with the number of parameters. We find that looped-moe models can answer four hop queries (q=4) over knowledge graphs that are up to 9x larger than parameter-matched standard models can. As illustrated in Figure 2, the gap between looped-moe and standard models grows as the number of hops in the queries increases. When only evaluating on 1-hop queries, there is no difference between the architectures.

The rates of growth in the size of the knowledge graph are consistent with the rates predicted by our theoretical analysis in Section 5.

Notice how the standard model's rate of improvement (slope of green lines in Figure 2) drops from approximately 0.061 triples per parameter at $q = 1$ to 0.0002 triples per parameter at $q = 3$ (a $30.5\times$ decrease). This drop in the slope as we increase q is consistent with the $\mathcal{O}(Tq)$ bound on model size provided by Theorem 5.1 and with the conjecture that there exists a matching lower bound of $\Omega(Tq)$.

In contrast, the looped-moe model's rate of improvement only drops from 0.053 to 0.015 triples per parameter (a $3.5\times$ decrease). This is closer to the ideal rate of $\mathcal{O}(T)$ predicted by Theorem 5.3, but there is still a modest dependency on q. Closing this gap between theory and practice is an important challenge going forward.

7. Conclusion

In this work, we show that *looping*, sharing the same parameters across layers, enables small models to memorize and answer queries over larger knowledge graphs. Our experiments are limited to a simple setting where a model is trained and evaluated directly on the triples in a knowledge graph. Future work should evaluate the benefits of looped models when trained on large, unstructured text and code corpora (Gao et al., 2020; Kocetkov et al., 2022).

References

Adolphs, L., Dhuliawala, S., and Hofmann, T. How to query language models? *arXiv preprint arXiv:2108.01928*, 2021.

AlKhamissi, B., Li, M., Celikyilmaz, A., Diab, M., and Ghazvininejad, M. A review on language models as knowledge bases. *arXiv preprint arXiv:2204.06031*, 2022.

Arora, S., Eyuboglu, S., Timalsina, A., Johnson, I., Poli, M., Zou, J., Rudra, A., and Ré, C. Zoology: Measuring and improving recall in efficient language models. *International Conference on Learning Representations*, 2023.

Bai, S., Kolter, J. Z., and Koltun, V. Deep equilibrium models. *Advances in neural information processing systems*, 32, 2019.

Bevilacqua, M., Ottaviano, G., Lewis, P., Yih, S., Riedel, S., and Petroni, F. Autoregressive search engines: Generating

substrings as document identifiers. *Advances in Neural Information Processing Systems*, 35:31668–31683, 2022.

Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., van den Driessche, G., Lespiau, J.-B., Damoc, B., Clark, A., de Las Casas, D., Guy, A., Menick, J., Ring, R., Hennigan, T., Huang, S., Maggiore, L., Jones, C., Cassirer, A., Brock, A., Paganini, M., Irving, G., Vinyals, O., Osindero, S., Simonyan, K., Rae, J. W., Elsen, E., and Sifre, L. Improving language models by retrieving from trillions of tokens, 2022.

Bosselut, A., Rashkin, H., Sap, M., Malaviya, C., Celikyilmaz, A., and Choi, Y. Comet: Commonsense transformers for automatic knowledge graph construction. *arXiv preprint arXiv:1906.05317*, 2019.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Csordás, R., van Steenkiste, S., and Schmidhuber, J. Are neural nets modular? inspecting functional modularity through differentiable weight masks. *arXiv preprint arXiv:2010.02066*, 2020.

Csordás, R., Piękos, P., and Irie, K. Switchhead: Accelerating transformers with mixture-of-experts attention. *arXiv preprint arXiv:2312.07987*, 2023.

Csordás, R., Irie, K., Schmidhuber, J., Potts, C., and Manning, C. D. Moeut: Mixture-of-experts universal transformers. *arXiv preprint arXiv:2405.16039*, 2024.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, Ł. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and ukasz Kaiser. Universal transformers, 2019.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Giannou, A., Rajput, S., Sohn, J.-y., Lee, K., Lee, J. D., and Papailiopoulos, D. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pp. 11398–11442. PMLR, 2023.

Guha, N., Nyarko, J., Ho, D., Ré, C., Chilton, A., Chohlas-Wood, A., Peters, A., Waldon, B., Rockmore, D., Zambrano, D., et al. Legalbench: A collaboratively built benchmark for measuring legal reasoning in large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Guu, K., Lee, K., Tung, Z., Pasupat, P., and Chang, M.-W. Realm: Retrieval-augmented language model pretraining, 2020.

Harris, S., Seaborne, A., and Prud'hommeaux, E. Sparql 1.1 query language. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/, 2013. W3C Recommendation.

Jaegle, A., Gimeno, F., Brock, A., Vinyals, O., Zisserman, A., and Carreira, J. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pp. 4651–4664. PMLR, 2021.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., de las Casas, D., Hanna, E. B., Bressand, F., Lengyel, G., Bour, G., Lample, G., Lavaud, L. R., Saulnier, L., Lachaux, M.-A., Stock, P., Subramanian, S., Yang, S., Antoniak, S., Scao, T. L., Gervet, T., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. Mixtral of experts, 2024.

Kandpal, N., Deng, H., Roberts, A., Wallace, E., and Raffel, C. Large language models struggle to learn long-tail knowledge, 2023.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models, 2020.

Khandelwal, U., Fan, A., Jurafsky, D., Zettlemoyer, L., and Lewis, M. Nearest neighbor machine translation, 2021.

Klyne, G. and Carroll, J. J. Resource description framework (rdf): Concepts and abstract syntax. 2004. URL http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

Kocetkov, D., Li, R., Allal, L. B., Li, J., Mou, C., Ferrandis, C. M., Jernite, Y., Mitchell, M., Hughes, S., Wolf, T., et al. The stack: 3 tb of permissively licensed source code. *arXiv preprint arXiv:2211.15533*, 2022.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. Albert: A lite bert for self-supervised learning of language representations, 2020.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., tau Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

Liu, B., Ash, J. T., Goel, S., Krishnamurthy, A., and Zhang, C. Transformers learn shortcuts to automata, 2023.

Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., Savarese, S., and Xiong, C. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*, 2022.

Nye, M., Andreassen, A. J., Gur-Ari, G., Michalewski, H., Austin, J., Bieber, D., Dohan, D., Lewkowycz, A., Bosma, M., Luan, D., et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

Pan, S., Luo, L., Wang, Y., Chen, C., Wang, J., and Wu, X. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–20, 2024. doi: 10.1109/TKDE.2024.3352100.

Petroni, F., Piktus, A., Fan, A., Lewis, P., Yazdani, M., De Cao, N., Thorne, J., Jernite, Y., Karpukhin, V., Maillard, J., et al. Kilt: a benchmark for knowledge intensive language tasks. *arXiv preprint arXiv:2009.02252*, 2020.

Press, O., Smith, N. A., and Lewis, M. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.

Press, O., Zhang, M., Min, S., Schmidt, L., Smith, N. A., and Lewis, M. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*, 2022.

Qin, G. and Eisner, J. Learning how to ask: Querying lms with mixtures of soft prompts. *arXiv preprint arXiv:2104.06599*, 2021.

Shazeer, N. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017.

Shen, Y., Zhang, Z., Cao, T., Tan, S., Chen, Z., and Gan, C. Moduleformer: Modularity emerges from mixture-of-experts, 2023.

Suchanek, F. M., Kasneci, G., and Weikum, G. Yago: a core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pp. 697706, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595936547. doi: 10.1145/1242572.1242667. URL https://doi.org/10.1145/1242572.1242667.

Tan, S., Shen, Y., Chen, Z., Courville, A., and Gan, C. Sparse universal transformer. *arXiv preprint arXiv:2310.07096*, 2023.

Taylor, W. L. cloze procedure: A new tool for measuring readability. *Journalism quarterly*, 30(4):415–433, 1953.

Team, G., Mesnard, T., Hardin, C., Dadashi, R., Bhupatiraju, S., Pathak, S., Sifre, L., Rivière, M., Kale, M. S., Love, J., et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Vrandečić, D. and Krötzsch, M. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):7885, sep 2014. ISSN 0001-0782. doi: 10.1145/2629489. URL https://doi.org/10.1145/2629489.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Workshop, B., Scao, T. L., Fan, A., Akiki, C., Pavlick, E., Ilić, S., Hesslow, D., Castagné, R., Luccioni, A. S., Yvon, F., et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

Wu, K., Wu, E., Cassasola, A., Zhang, A., Wei, K., Nguyen, T., Riantawan, S., Riantawan, P. S., Ho, D. E., and Zou, J. How well do llms cite relevant medical references? an evaluation framework and analyses. *arXiv preprint arXiv:2402.02008*, 2024.

Xue, F., Shi, Z., Wei, F., Lou, Y., Liu, Y., and You, Y. Go wider instead of deeper. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 8779–8787, 2022.

Yang, L., Lee, K., Nowak, R., and Papailiopoulos, D. Looped transformers are better at learning learning algorithms. *arXiv preprint arXiv:2311.12424*, 2023.

Yuksekgonul, M., Chandrasekaran, V., Jones, E., Gunasekar, S., Naik, R., Palangi, H., Kamar, E., and Nushi, B. Attention satisfies: A constraint-satisfaction lens on factual errors of language models, 2024.

Zhang, X., Bosselut, A., Yasunaga, M., Ren, H., Liang, P., Manning, C. D., and Leskovec, J. Greaselm: Graph reasoning enhanced language models for question answering. *arXiv preprint arXiv:2201.08860*, 2022a.

Zhang, X., Shen, Y., Huang, Z., Zhou, J., Rong, W., and Xiong, Z. Mixture of attention heads: Selecting attention heads per token. *arXiv preprint arXiv:2210.05144*, 2022b.

Zhang, Y., Backurs, A., Bubeck, S., Eldan, R., Gunasekar, S., and Wagner, T. Unveiling transformers with lego: a synthetic reasoning task. *arXiv preprint arXiv:2206.04301*, 2022c.

A. Extended Related Work

Mixture of experts Mixture-of-expert layers have received increased attention recently (Fedus et al., 2022; Jiang et al., 2024; Shazeer et al., 2017). An MoE model is a neural network that uses only a subset of its parameters to process each input. In principle, MoE models can store factual information at a lower computational cost than traditional models. However, realizing these efficiency gains in practice is challenging (Shazeer et al., 2017). Because MoE models have a huge number of parameters, they are often IO-bound during inference, requiring low-level implementations that carefully manage memory. Research into making MoE models more efficient is ongoing (Fedus et al., 2022; Jiang et al., 2024).

Mixture of experts is typically applied to the MLP and not the sequence mixer (*e.g.* attention). However, recently, several works have studied the application MoEs to the attention mechanism (Zhang et al., 2022b; Shen et al., 2023; Tan et al., 2023; Csordás et al., 2023). These works differ in which part of the attention mechanism they make conditional/sparse. For example, in SwitchHead (Csordás et al., 2023) only the value and output projections are made conditional, while in Mixture of Attention Heads and Sparse Universal Transformers (Zhang et al., 2022b; Tan et al., 2023) the query and the output projections are made conditional.

Looping layers. In the context of large language models, weight-tying was first studied in works on universal transformers (Dehghani et al., 2018), deep equilibrium models (Bai et al., 2019), and Albert (Lan et al., 2020), perceiver (Jaegle et al., 2021).

Several works have studied looped models on synthetic tasks. More recently, Giannou et al. (2023) showed that looped models can implement simple programs. In a follow-on work, they show that looped transformers are better at in-context learning (Yang et al., 2023). Related to our study, Csordás et al. (2020) study weight-tying in the context of modularity. The Learning Equality and Group Operations (LEGO) synthetic task (Zhang et al., 2022c) is equivalent to the path queries over a matching graph studied in our work (see Section 4). (Zhang et al., 2022c) show that weight-tied models (specifically Albert (Lan et al., 2020)) demonstrate improved generalization for longer path lengths. However, they do not study the space complexity.

Language models as knowledge bases. A number of works have explored techniques for improving the access of knowledge through fine-tuning or prompting (AlKhamissi et al., 2022). For single-hop queries, cloze-style prompting is the default strategy (Adolphs et al., 2021; Taylor, 1953), optionally augmented with a mixture of prompts (Qin & Eisner, 2021). For more complicated, multi-step queries, “scratchpads" emerged as a way to perfrom iterated computation (Nye et al., 2021). (Wei et al., 2022; Press et al., 2022) built on this work with “chain-of-thought" reasoning.

A number of papers propose datasets or methodology for evaluating the factuality of language models (Petroni et al., 2020). (Kandpal et al., 2023) show the limitations of language models in encoding long-tail knowledge. Most related to our work is (Yuksekgonul et al., 2024) who show that large language models struggle to answer constraint satisfaction problems.

Several works highlight a *compositionality gap* in language models. For example, Press *et al.* show that language models often fail to answer multi-hop questions even when they can answer the constituent single-hop questions. Strikingly, they find that this gap *increases* with model and dataset scale (Press et al., 2022).

Augmenting generation with external knowledge sources. One approach to addressing the information storage and access problem is to augment generations with external knowledge bases. In this approach, connects large language models to external databases, allowing the model to dynamically retrieve and integrate relevant information as part of the generation process. Recently, retrieval-augmented generation (RAG) approaches, in which language models retrieve from large corpora of *unstructured* text, has received significant interest from both industry and academic communities (Borgeaud et al., 2022; Guu et al., 2020; Khandelwal et al., 2021; Lewis et al., 2021; Bevilacqua et al., 2022). Related to RAG, there is also a long line of work on leveraging *structured* knowledge graphs (KG) in language model predictions (Zhang et al., 2022a; Bosselut et al., 2019; Pan et al., 2024).

In principle, RAG and KG based approaches enable AI systems to cheaply access huge amounts of factual information. However, in practice, the strongest RAG models still store huge amounts of factual information stored in their parameters and can “hallucinate" just like non-RAG models (Wu et al., 2024). Thus, we view RAG as a complementary to work on parametric knowledge, not a replacement.

B. Theoretical Analysis

B.1. Summary of results

In Appendix B.2 (**Model Architectures**), we define the standard architecture and looped-conditional architectures. In Appendix B.4 (**Primitives**), we define a set of basic primitives we will use in later proofs. We'll prove the space and time complexity standard and looped-moe models require to implement the primitives. In Appendix B.5 (**Path Queries**), we provide bounds on the space and time complexity required for each architecture to solve path queries on matching graphs.

In the following, all vectors are assumed to be column vectors.

B.2. Model Architectures

Standard architecture Language models with the *standard* architecture consist of L layers. Each layer is defined as the composition of a *sequence mixer* (*e.g.* attention (Vaswani et al., 2017)) and a *state mixer* (*e.g.* MLP). The ℓ^{th} layer takes as input $\boldsymbol{x}^\ell \in \mathbb{R}^{N\times d}$ of length N and dimension d, and computes output $\boldsymbol{y}^\ell \in \mathbb{R}^{N\times d}$. The layers are applied sequentially such that $\boldsymbol{y}^\ell = \boldsymbol{x}^{\ell+1}$. Our focus is on the state mixer and assume the attention module is defined as in (Arora et al., 2023).

In this work, we assume the sequence mixer is attention, parameterized by three projection matrices $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{N\times d}$. Attention first projects the input $\boldsymbol{x}$: $\boldsymbol{q} = \boldsymbol{Q}\boldsymbol{x}$, $\boldsymbol{k} = \boldsymbol{K}\boldsymbol{x}$, $\boldsymbol{v} = \boldsymbol{V}\boldsymbol{x}$. Then, the projected embeddings are aggregated along the sequence dimensions according to:

$$\text{Attn}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{B}; \boldsymbol{x}) = \text{softmax}(\frac{1}{\sqrt{d}}\boldsymbol{q}\boldsymbol{k}^\top + \boldsymbol{B})\boldsymbol{v} \tag{13}$$

To simplify the theoretical analysis, instead of positional embeddings, we assume there is a fixed bias matrix $\boldsymbol{B}^{N\times N}$ added to the attention scores before the softmax. This resembles ALiBi, a popular technique commonly used when training language models with length extrapolation (Press et al., 2021). The values of the bias matrix are controlled by a hyperparameter so they do not count towards the parameter count.

In this work, we assume that the *state mixer* is a GLU defined as:

$$\text{GLU}(\boldsymbol{W}^{\text{in}}, \boldsymbol{W}^{\text{gate}}, \boldsymbol{W}^{\text{out}}, \boldsymbol{b}^1, \boldsymbol{b}^2; \boldsymbol{x}) = (\sigma(\boldsymbol{x}\boldsymbol{W}^{\text{in}} + \boldsymbol{b}^1) \odot (\boldsymbol{x}\mathbf{W}^{\text{gate}} + \boldsymbol{b}^2))\boldsymbol{W}^{\text{out}} \tag{14}$$

where σ is an element-wise non-linearity (*e.g.* ReLU), $\mathbf{W}^{\text{in}} \in \mathbb{R}^{d\times d'}, \boldsymbol{W}^{\text{gate}} \in \mathbb{R}^{d\times d'}, \mathbf{W}^{\text{out}} \in \mathbb{R}^{d'\times d}, \boldsymbol{b}^1 \in \mathbb{R}^{1\times d'}$, and $\boldsymbol{b}^2 \in \mathbb{R}^{1\times d'}$.

A *layer* is simply the composition of these two operations (with residual connections):

$$\boldsymbol{y} = \text{Attn}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{B}; \boldsymbol{x}) \tag{15}$$

$$\text{Layer}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{B}, \boldsymbol{W}^{\text{in}}, \boldsymbol{W}^{\text{gate}}, \boldsymbol{W}^{\text{out}}, \boldsymbol{b}^1, \boldsymbol{b}^2; \boldsymbol{x}) = \text{GLU}(\boldsymbol{W}^{\text{in}}, \boldsymbol{W}^{\text{gate}}, \boldsymbol{W}^{\text{out}}, \boldsymbol{b}^1, \boldsymbol{b}^2; \boldsymbol{y} + \boldsymbol{x}) + \boldsymbol{y} + \boldsymbol{x} \tag{16}$$

In the standard architecture, different weights are used at each layer, so at the ℓ^{th} layer we compute $\boldsymbol{y}^\ell = \text{Layer}(\boldsymbol{Q}^\ell, \boldsymbol{K}^\ell, \boldsymbol{V}^\ell, \boldsymbol{B}^\ell, \boldsymbol{W}^{\text{in},\ell}, \boldsymbol{W}^{\text{gate},\ell}, \boldsymbol{W}^{\text{out},\ell}, \boldsymbol{b}^{1,\ell}, \boldsymbol{b}^{2,\ell}; \boldsymbol{x}^\ell)$.

Below we describe the variations on the standard architecture that we study in this work.

Mixture-of-experts layers A mixture-of-experts (MoE) model routes each input to a different subset of its parameters (Shazeer et al., 2017). This allows us to increase the number of parameters in a model without increasing the number of FLOPs required for inference. Below we define a simple MoE for an MLP. We use softmax and a single linear layer with weight $\boldsymbol{W}^{\text{switch}} \in \mathbb{R}^{d\times m}$ to route to k (typically $k = 2$) experts. Each token $\boldsymbol{x}[j]$ in the input is routed separately.

$$s[j, i] = \text{softmax}(\boldsymbol{x}[j]\boldsymbol{W}^{\text{switch}})[i] \tag{17}$$

$$\tilde{s}[i] = \begin{cases} s[i] & \text{if } s[i] \text{ in top } k \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

$$\text{MOE}(\{\boldsymbol{W}_i^{\text{in}}\}_{i=1}^k, \{\boldsymbol{W}_i^{\text{gate}}\}_{i=1}^k, \{\boldsymbol{W}_i^{\text{out}}\}_{i=1}^k, \{\boldsymbol{b}_i^1\}_{i=1}^k, \{\boldsymbol{b}_i^2\}_{i=1}^k, \boldsymbol{W}^{\text{switch}}; \boldsymbol{x})[j] = \sum_{i=1}^m \text{GLU}(\boldsymbol{W}_i^{\text{in}}, \boldsymbol{W}_i^{\text{gate}}\boldsymbol{W}_i^{\text{out}}, \boldsymbol{b}^1, \boldsymbol{b}^2; \boldsymbol{x})[j] \frac{\tilde{s}[j, i]}{\sum_{p=1}^m \tilde{s}[j, p]} \tag{19}$$

Looped layers In a standard language model, different weights are used at every layer. In a *looped* language model, a single block of B layers is repeatedly applied $\frac{L}{B}$ times. In other words, we only maintain B layers worth of weights and at layer l we use the weights from layer $l \bmod B$.

In the extreme case ($B = 1$), we maintain one set of weights $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{B}, \boldsymbol{W}^{\text{in}}, \boldsymbol{W}^{\text{gate}}, \boldsymbol{W}^{\text{out}}, \boldsymbol{b}^1, \boldsymbol{b}^2$, and reuse them at every layer:

$$\forall \ell \in [L] \qquad \boldsymbol{y}^\ell = \text{Layer}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{B}, \boldsymbol{W}^{\text{in}}, \boldsymbol{W}^{\text{gate}}, \boldsymbol{W}^{\text{out}}, \boldsymbol{b}^1, \boldsymbol{b}^2; \boldsymbol{x}^\ell) \tag{20}$$

This is sometimes referred to as a *universal model* (Dehghani et al., 2019), *weight-tied* (Lan et al., 2020; Xue et al., 2022), or *iterative* (Jaegle et al., 2021) model in the literature.

B.3. Problem Formulation

We study the formal problem of memorizing a knowledge graph and answering queries over the information in it. In our theoretical analysis, we use the problem formulation from Section 4 with the following modifications to the encoding scheme.

As described in Section 4, the input to the model is a sequence of tokens representing the query. For example, the input to the model for the query in Equation (10) is shown below where each item in brackets is one token.

$$\textit{Input:} \texttt{[}e_1\texttt{]} \;\; \texttt{[}r_1\texttt{]} \;\; \texttt{[v]} \;\; \texttt{[and]} \;\; \texttt{[v]} \;\; \texttt{[}r_2\texttt{]} \;\; \texttt{[s]} \tag{21}$$

The input token sequence is *encoded* as a matrix $\boldsymbol{u} \in \mathbb{R}^{N \times d}$ where N is the length of the sequence and d is the hidden dimension of model.

Because we only study path queries, in our theoretical analysis, we drop the conjunction token `[and]` from the predicate. Further, we "flatten" each triple so that the input is $\boldsymbol{u} \in \mathbb{R}^{q \times d}$.

$$\boldsymbol{u} \equiv \begin{pmatrix} \boldsymbol{e_1}^\top & \boldsymbol{r_1}^\top & \boldsymbol{v}^\top & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{v}^\top & \boldsymbol{r_2}^\top & \boldsymbol{s}^\top & \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}$$

Both of these modifications could be implemented with attention, but we exclude them to simplify the presentation.

Encoding The input to the model is a sequence tokens representing the query. For example, the input to the model for the query in Equation (10) is shown below where each item in brackets is one token.

$$\textit{Input:} \texttt{[}e_1\texttt{]} \;\; \texttt{[}r_1\texttt{]} \;\; \texttt{[y]} \;\; \texttt{[and]} \;\; \texttt{[y]} \;\; \texttt{[}r_2\texttt{]} \;\; \texttt{[x]} \tag{22}$$

The input token sequence is *encoded* as a matrix $\boldsymbol{u} \in \mathbb{R}^{N \times d}$ where N is the length of the sequence and d is the hidden dimension of model. In our theoretical analysis, we assume each token $\boldsymbol{u}[i]$ in the vocabulary is represented by a unique binary code $\boldsymbol{u}[i] \in \{-1, 1\}^d$. Representing the full binary code for each token requires $d \geq \log_2 c$. If $d > \log_2 c$, the rest of embedding is padded with zeros. In our experiments, token embeddings are initialized randomly and learned.

The model transforms $\boldsymbol{u}$ into a matrix $\boldsymbol{y} \in \mathbb{R}^{N \times d}$. The output matrix can be *decoded* into a sequence of N tokens (*e.g.* via a nearest neighbor lookup). This sequence is then compared with a ground truth label sequence of the same length as the input. The labels corresponding to the input in Equation (22) are:

$$\textit{Labels:} \texttt{[-]} \;\; \texttt{[-]} \;\; \texttt{[-]} \;\; \texttt{[-]} \;\; \texttt{[-]} \;\; \texttt{[-]} \;\; \texttt{[}e_4\texttt{]} \tag{23}$$

When computing loss and accuracy, the labels `[-]` are ignored.

B.4. Primitives

Definition B.1. `KeepNegatives`$(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{B}, \boldsymbol{W}^{\text{in}}, \boldsymbol{W}^{\text{gate}}, \boldsymbol{W}^{\text{out}}, \boldsymbol{b}^1, \boldsymbol{b}^2, l, r; \boldsymbol{x}) \to \boldsymbol{z}$

INPUT: $\boldsymbol{Q} \in \mathbb{R}^{d' \times d'}, \boldsymbol{K} \in \mathbb{R}^{d' \times d'}, \boldsymbol{V} \in \mathbb{R}^{d' \times d'}, \boldsymbol{B} \in \mathbb{R}^{N \times N}, \boldsymbol{W}^{\text{in}} \in \mathbb{R}^{d' \times d'}, \boldsymbol{W}^{\text{gate}} \in \mathbb{R}^{d' \times d'}, \boldsymbol{W}^{\text{out}} \in \mathbb{R}^{d' \times d'}, \boldsymbol{b}^1 \in \mathbb{R}^{1 \times d'}, \boldsymbol{b}^2 \in \mathbb{R}^{1 \times d'}, \boldsymbol{x} \in \mathbb{R}^{N' \times d'}, l \in \mathbb{Z}, r \in \mathbb{Z}$.

OUTPUT: $\boldsymbol{z} \in \mathbb{R}^{N' \times d'}$ such that it is defined as:

$$\begin{aligned}\boldsymbol{y} &= \text{Attn}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{B}; \boldsymbol{x}) + \boldsymbol{x} \\ \boldsymbol{z}[:, i] &= \left(\left(\left(\boldsymbol{x}\boldsymbol{W}^{\text{in}} + \boldsymbol{b}^1\right) \odot \left(\boldsymbol{x}\boldsymbol{W}^{\text{gate}} + \boldsymbol{b}^2\right)\right) \boldsymbol{W}^{\text{out}}\right) + y \text{ if } i < l \text{ or } i \geq r \\ \boldsymbol{z}[:, i] &= GLU(\boldsymbol{W}^{\text{in}}, \boldsymbol{W}^{\text{gate}}, \boldsymbol{W}^{\text{out}}, \boldsymbol{b}^1, \boldsymbol{b}^2) \text{ if } l \leq i < r.\end{aligned}$$

Note that we can set $l = r$ so that all negatives are kept or $l < 0$ and $r > d'$ so that the normal functionality of a GLU is implemented.

Definition B.2. $\boxed{\texttt{NoResidual}(\boldsymbol{W}^{\text{in}}, \boldsymbol{W}^{\text{gate}}, \boldsymbol{W}^{\text{out}}, \boldsymbol{b}^1, \boldsymbol{b}^2; \boldsymbol{x}) \to \boldsymbol{y}}$

INPUT: $\boldsymbol{W}^{\text{in}} \in \mathbb{R}^{d' \times d'}, \boldsymbol{W}^{\text{gate}} \in \mathbb{R}^{d' \times d'}, \boldsymbol{W}^{\text{out}} \in \mathbb{R}^{d' \times d'}, \boldsymbol{b}^1 \in \mathbb{R}^{1 \times d'}, \boldsymbol{b}^2 \in \mathbb{R}^{1 \times d'}, \boldsymbol{x} \in \mathbb{R}^{N' \times d'}$
OUTPUT: $\boldsymbol{y} \in \mathbb{R}^{N' \times d'}$
Allows a layer to apply an arbitrary GLU function without the residual being added. Specifically,

$$\boldsymbol{y} = \text{GLU}(\boldsymbol{W}^{\text{in}}, \boldsymbol{W}^{\text{gate}}, \boldsymbol{W}^{\text{out}}, \boldsymbol{b}^1, \boldsymbol{b}^2; \boldsymbol{x}).$$

Definition B.3. $\boxed{\texttt{InvertColumn}(k; \boldsymbol{x}) \to \boldsymbol{y}}$

INPUT: $\boldsymbol{x} \in \mathbb{R}^{N' \times d}, k \in \mathbb{Z}, \boldsymbol{x}[:, i] \in \{-1, 0, 1\}^{N' \times 1}$.
OUTPUT: $\boldsymbol{y} \in \mathbb{R}^{N' \times d'}$. Where $\boldsymbol{y}$ is defined as:

$$\boldsymbol{y}[i, j] = \begin{cases} \boldsymbol{x}[i, j] \text{ if } j \neq k \\ 1 \text{ if } j = k \text{ and } \boldsymbol{x}[i, j] = 0 \\ 0 \text{ if } j = k \text{ and } |\boldsymbol{x}[i, j]| = 1 \end{cases}$$

Definition B.4. $\boxed{\texttt{lookup}_{\mathcal{G}}(\boldsymbol{x}) \to \boldsymbol{y}}$

- **Input:** a sequence $\boldsymbol{y} \in \mathbb{R}^{N \times d}$ where each row contains the encoding for a head entity $\boldsymbol{h} \in \mathbb{R}^{\log_2 c}$, a relation $\boldsymbol{r} \in \mathbb{R}^{\log_2 c}$, and a variable $\boldsymbol{v} \in \mathbb{R}^{\log_2 c}$.
- **Output:** a sequence $\boldsymbol{z} \in \mathbb{R}^{N \times d}$ where the first three column blocks of the row are identical to the input. If there exists a triple (h, r, t) in $\mathcal{G}$, then the fourth and fifth column blocks of each row contain $\boldsymbol{v}$ and $\boldsymbol{t}$, respectively. If no such triple exists, that row is identical to the corresponding row in the input. Here is an example of a lookup hit:

  $$\boldsymbol{y}[i] \equiv \begin{pmatrix} \boldsymbol{e}^\top & \boldsymbol{r}^\top & \boldsymbol{v}^\top & \mathbf{0} & \mathbf{0} \end{pmatrix} \qquad \boldsymbol{z}[i] \equiv \begin{pmatrix} \boldsymbol{e}^\top & \boldsymbol{r}^\top & \boldsymbol{v}^\top & \boldsymbol{v}^\top & \boldsymbol{t}^\top \end{pmatrix}$$

  If there is no hit for a row, this is how lookup effects the row:

  $$\boldsymbol{y}[i] \equiv \begin{pmatrix} \boldsymbol{v}^\top & \boldsymbol{r}^\top & \boldsymbol{s}^\top & \mathbf{0} & \mathbf{0} \end{pmatrix} \qquad \boldsymbol{z}[i] \equiv \begin{pmatrix} \boldsymbol{v}^\top & \boldsymbol{r}^\top & \boldsymbol{s}^\top & \mathbf{0} & \mathbf{0} \end{pmatrix}$$

Definition B.5. $\boxed{\texttt{substitute}(\boldsymbol{y}) \to \boldsymbol{z}}$

- **Input:** a sequence $\boldsymbol{y} \in \mathbb{R}^{N \times d}$ containing query predicates, as defined in Equation (9) in the first three column blocks of each row(*i.e.* $\boldsymbol{y}[i, : 3\log_2 c] = \big(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}\big)$) and containing solutions(variable-entity pairs) in the fourth and fifth column blocks(*i.e.* $y[i, 3\log_2 c + 1 : 5\log_2 c] = \big(\boldsymbol{v}, \boldsymbol{e}\big)$). These rows need not have values and can hold $\mathbf{0}^{1\times d}$ to indicate no query and no solution.
- **Output:** a sequence $\boldsymbol{z} \in \mathbb{R}^{N \times d}$ containing the same first three column blocks as $\boldsymbol{y}$ except that any row i where the 1st column block matches the 4th column block of j, i has its first column block replaced by the 5th column block of j. The remaining column blocks are zeroed out.

$$\boldsymbol{y} \equiv \begin{pmatrix} & & \vdots & & \\ \boldsymbol{e}^\top & \boldsymbol{r}^\top & \boldsymbol{v}^\top & \boldsymbol{v}^\top & \boldsymbol{t}^\top \\ & & \vdots & & \\ \boldsymbol{v}^\top & \boldsymbol{r}^\top & \boldsymbol{s}^\top & \mathbf{0} & \mathbf{0} \\ & & \vdots & & \end{pmatrix} \qquad \boldsymbol{z} \equiv \begin{pmatrix} & & \vdots & & \\ \boldsymbol{e}^\top & \boldsymbol{r}^\top & \boldsymbol{v}^\top & \mathbf{0} & \mathbf{0} \\ & & \vdots & & \\ \boldsymbol{t}^\top & \boldsymbol{r}^\top & \boldsymbol{s}^\top & \mathbf{0} & \mathbf{0} \\ & & \vdots & & \end{pmatrix}$$

Proposition B.6. *For any $\boldsymbol{x}$ there is a constant-depth block of layers that computes* `KeepNegatives`*.*

Proof. Here are the steps to implement `KeepNegatives` exclusively through Layer.

1. INPUT: $\boldsymbol{x} \in \mathbb{R}^{N\times d}, l, r$.
   OUTPUT: $\boldsymbol{y} \in \mathbb{R}^{N\times d}$ where $\boldsymbol{y}$ is defined by:

$$\boldsymbol{y} = \text{GLU}(\overline{\boldsymbol{W}}^{in}, \overline{\boldsymbol{W}}^{gate}, \overline{\boldsymbol{W}}^{out}, \overline{\boldsymbol{b}}^1, \overline{\boldsymbol{b}}^2; \overline{\boldsymbol{x}}).$$

   Where $\overline{\boldsymbol{W}}^{in} \in \mathbb{R}^{d\times 2d'}, \overline{\boldsymbol{W}}^{gate} \in \mathbb{R}^{d\times 2d'}, \overline{\boldsymbol{W}}^{out} \in \mathbb{R}^{2d'\times d}, \overline{\boldsymbol{b}}^1 \in \mathbb{R}^{1\times 2d'}, \overline{\boldsymbol{b}}^2 \in \mathbb{R}^{1\times 2d'}, \overline{\boldsymbol{x}} \in \mathbb{R}^{N\times 2d}$ are defined as:

$$\boldsymbol{W}^{\text{out}\prime}[i,:] = \mathbf{0}^{1\times d'} \text{ if } l \le i < r$$
$$\boldsymbol{W}^{\text{out}\prime}[i,:] = \boldsymbol{W}^{\text{out}}[i,:] \text{ otherwise}$$

$$\overline{\boldsymbol{W}}^{in} = \left(\boldsymbol{W}^{\text{in}}, -\boldsymbol{W}^{\text{in}}\right), \overline{\boldsymbol{W}}^{gate} = \left(\boldsymbol{W}^{\text{gate}}, \boldsymbol{W}^{\text{gate}}\right), \overline{\boldsymbol{W}}^{out} = \begin{pmatrix} \boldsymbol{W}^{\text{out}} \\ \hline -\boldsymbol{W}^{\text{out}\prime} \end{pmatrix},$$

$$\overline{\boldsymbol{b}}^1 = \left(\boldsymbol{b}^1, -\boldsymbol{b}^1\right), \overline{\boldsymbol{b}}^2 = \left(\boldsymbol{b}^2, \boldsymbol{b}^2\right), \overline{\boldsymbol{x}} = \left(\boldsymbol{x}, \mathbf{0}^{N\times d}\right).$$

   From the output, take the first d columns and that is the final solution. The correctness of this follows from the fact that $u = Relu(u) - Relu(-u)$.

□

Corollary B.7. `KeepNegatives`$(f, l, r, \boldsymbol{x})$*. Being that keep negatives can implement Layer, any f that is composed to exclusively Layer can be implemented through* `KeepNegatives` *with any subset of negatives being kept.*

Proposition B.8. *For any $\boldsymbol{x} \in \mathbb{R}^{N\times d}$ there is a constant-depth(2) block of layers that computes* `NoResidual`$(\boldsymbol{W}^{in}, \boldsymbol{W}^{gate}, \boldsymbol{W}^{out}, \boldsymbol{b}^1, \boldsymbol{b}^2; \boldsymbol{x})$.

Proof. Here are the steps to implement `NoResidual` exclusively through Layer.

1. INPUT: $\boldsymbol{x}^{N\times d}$
   OUTPUT: $\boldsymbol{y}_1 \in \mathbb{R}^{N\times d}$ where $\boldsymbol{y}_1$ is defined as $\texttt{KeepNegatives}(f_1, 0, 0, \boldsymbol{x})$ where:

$$f_1 = \text{Layer}(\mathbf{0}^{d\times d}, \mathbf{0}^{d\times d}, \mathbf{0}^{d\times d}, \mathbf{0}^{N\times N}, \overline{\boldsymbol{W}}_1^{in}, \overline{\boldsymbol{W}}_1^{gate}, \overline{\boldsymbol{W}}_1^{out}, \overline{\boldsymbol{b}}_1^1, \overline{\boldsymbol{b}}_1^2; \boldsymbol{x}).$$

   Where $\overline{\boldsymbol{W}}_1^{in} \in \mathbb{R}^{2d\times 2d'}, \overline{\boldsymbol{W}}_1^{gate} \in \mathbb{R}^{2d\times 2d'}, \overline{\boldsymbol{W}}_1^{out} \in \mathbb{R}^{2d'\times 2d}, \overline{\boldsymbol{b}}_1^1 \in \mathbb{R}^{1\times 2d'}, \overline{\boldsymbol{b}}_1^2 \in \mathbb{R}^{1\times 2d'}$ are defined as:

$$\overline{\boldsymbol{W}}_1^{in} = \left(\begin{array}{cc} \mathbf{0}^{d\times d'} & \boldsymbol{W}^{in} \\ \hline \mathbf{0}^{d\times d'} & \mathbf{0}^{d\times d'} \end{array}\right), \overline{\boldsymbol{W}}_1^{gate} = \left(\begin{array}{cc} \mathbf{0}^{d\times d'} & \boldsymbol{W}^{gate} \\ \hline \mathbf{0}^{d\times d'} & \mathbf{0}^{d\times d'} \end{array}\right), \overline{\boldsymbol{W}}_1^{out} = \left(\begin{array}{cc} \mathbf{0}^{d'\times d} & \mathbf{0}^{d'\times d} \\ \hline \mathbf{0}^{d'\times d} & \boldsymbol{W}^{gate} \end{array}\right),$$

$$\overline{\boldsymbol{b}}_1^1 = \begin{pmatrix} \mathbf{0}^{1\times d'} & \boldsymbol{b}^1 \end{pmatrix}, \overline{\boldsymbol{b}}_1^2 = \begin{pmatrix} \mathbf{0}^{1\times d'} & \boldsymbol{b}^2 \end{pmatrix}.$$

   This computes the layer we want to compute and stores it in the second portion of the matrix. So now the left half of the matrix stores the original values as they get added in by the residual, while the right portion stores the computed output of our desired layer without residual.

2. INPUT: $\boldsymbol{y}_1$
   OUTPUT: $\boldsymbol{y} \in \mathbb{R}^{N\times d}$ where $\boldsymbol{y}$ is defined as $\texttt{KeepNegatives}(f_2, 0, 0, \boldsymbol{y}_1)$. Where,

$$f_2 = \text{Layer}(\mathbf{0}^{d\times d}, \mathbf{0}^{d\times d}, \mathbf{0}^{d\times d}, \mathbf{0}^{N\times N}, \mathbf{0}^{2d\times 2d'}, \boldsymbol{I}^{2d\times 2d'}, \overline{\boldsymbol{W}}_2^{out}, \mathbf{1}^{1\times 2d'}, \mathbf{0}^{1\times 2d'}; \boldsymbol{y}_1).$$

   Where $\overline{\boldsymbol{W}}_2^{out}$ is defined as:

$$\overline{\boldsymbol{W}}_2^{out} = \left(\begin{array}{cc} -\mathbf{1}^{d'\times d} & \mathbf{0}^{d'\times d} \\ \hline \mathbf{1}^{d'\times d} & -\mathbf{1}^{d'\times d} \end{array}\right).$$

   This moves that calculated value to the left half of the matrix while negating out the residuals, giving us our desired output in the original portion of the matrix.

□

Proposition B.9. *For any $\boldsymbol{x} \in \mathbb{R}^{N\times d}$ and $1 \le i \le d'$, and the additional restriction that $\boldsymbol{x}[:, i] \in \{-1, 0, 1\}^{N\times 1}$, there is a constant-depth(4) block of layers that computes $\texttt{InvertColumn}(i, \boldsymbol{x})$.*

Proof. Here are the steps to implement `InvertColumn` exclusively through Layer.

1. INPUT: $\boldsymbol{x} \in \mathbb{R}^{N\times d}$
   OUTPUT: $\boldsymbol{y}_1 \in \mathbb{R}^{N\times d}$, where $\boldsymbol{y}_1$ is defined as: $\texttt{KeepNegatives}(f_1, 0, 0, \boldsymbol{x})$. Where

$$f_1 = \texttt{NoResidual}(\boldsymbol{I}^{d\times d'}, \overline{\boldsymbol{W}}_1^{gate}, \boldsymbol{I}^{d'\times d}, \mathbf{1}^{1\times d'}, \overline{\boldsymbol{b}}_1^2; \boldsymbol{x}),$$

   where $\overline{\boldsymbol{W}}_1^{gate} \in \mathbb{R}^{d\times d'}, \overline{\boldsymbol{b}}_1^2 \in \mathbb{R}^{1\times d'}$ are defined as:

$$\boldsymbol{W}_1^{gate} = \left(\begin{array}{ccc} & \mathbf{0}^{(i-1)\times d'} & \\ \hline \mathbf{0}^{1\times(i-1)} & 1 & \mathbf{0}^{1\times(d'-i)} \\ \hline & \mathbf{0}^{(d-i)\times d'} & \end{array}\right), \boldsymbol{b}_1^2 = \begin{pmatrix} \mathbf{0}^{1\times(i-1)} & 1 & \mathbf{0}^{1\times(d'-i)} \end{pmatrix}.$$

   This element wise squares the column we want to invert while keeping all other values the same. The goal of this step is to convert all -1 in the column to be inverted to 1s.

2. INPUT: $\boldsymbol{y}_1$
   OUTPUT: $\boldsymbol{y}$ where $\boldsymbol{y} = \texttt{KeepNegatives}(f_2, 0, 0, \boldsymbol{y}_1)$. Where

$$f_2 = \texttt{NoResidual}(\mathbf{0}^{d\times d'}, \overline{\boldsymbol{W}}_2^{gate}, \boldsymbol{I}^{d'\times d}, \mathbf{1}^{1\times d'}, \overline{\boldsymbol{b}}_1^2; \boldsymbol{y}_1).$$

   Where $\overline{\boldsymbol{W}}_2^{gate}, \overline{\boldsymbol{b}}_1^2$ are defined as:

$$\overline{\boldsymbol{W}}_2^{gate} = \left(\begin{array}{ccc} \boldsymbol{I}^{(i-1)\times(i-1)} & \mathbf{0}^{(i-1)\times(d'-i+1)} & \\ \hline \mathbf{0}^{1\times(i-1)} & -1 & \mathbf{0}^{1\times(d'-i)} \\ \hline \mathbf{0}^{(d-i)\times i} & & \boldsymbol{I}^{(d-i)\times(d'-1)} \end{array}\right), \overline{\boldsymbol{b}}_2^2 = \left(\mathbf{0}^{1\times(i-1)}, 1, \mathbf{0}^{d'-i}\right).$$

   This negates all values in our target column and adds 1, successfully flipping the bit values while maintaining the other values in the matrix.

□

Proposition B.10. *A constant-depth(4) block of layers using $\mathcal{O}(T\log c + \log^2 c)$ parameters and $\mathcal{O}(qT\log c + q\log^2 c)$ FLOPs can implement* `Lookup` *for all T triples from a matching graph $\mathcal{G}(E, R, T)$.*

Proof. Here are the steps to implement `Lookup`.

1. INPUT: $\boldsymbol{x} \in \mathbb{R}^{N\times d}$
   OUTPUT: $\boldsymbol{y}_1 \in \mathbb{R}^{N\times d}$ where $\boldsymbol{y}_1 = \texttt{KeepNegatives}(f_1, 4\log_2 c, 5\log_2 c, \boldsymbol{x})$ where:

$$f_1 = \texttt{NoResidual}(\boldsymbol{W}_1^{in}, \mathbf{0}^{d\times d'}, \boldsymbol{W}_1^{out}, \boldsymbol{b}^1, \mathbf{0}^{1\times d'}; \boldsymbol{x}),$$

   where $\boldsymbol{W}_1^{in} \in \mathbb{R}^{d\times d'}, \boldsymbol{W}_1^{out} \in \mathbb{R}^{d'\times d}$ are defined as:

$$\boldsymbol{W}_1^{in} = \left(\begin{array}{c} \boldsymbol{I}^{\log_2 c\times\log_2 c}, \mathbf{0}^{\log_2 c\times\log_2 c}, \mathbf{0}^{\log_2 c\times\log_2 c}, \mathbf{0}^{\log_2 c\times\log_2 c}, \boldsymbol{e}_1, \ldots, \boldsymbol{e}_T, \mathbf{0}\rightarrow \\ \hline \mathbf{0}^{\log_2 c\times\log_2 c}, \boldsymbol{I}^{\log_2 c\times\log_2 c}, \mathbf{0}^{\log_2 c\times\log_2 c}, \mathbf{0}^{\log_2 c\times\log_2 c}, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_T, \mathbf{0}\rightarrow \\ \hline \mathbf{0}^{\log_2 c\times\log_2 c}, \mathbf{0}^{\log_2 c\times\log_2 c}, \boldsymbol{I}^{\log_2 c\times\log_2 c}, \boldsymbol{I}^{\log_2 c\times\log_2 c}, \mathbf{0}\rightarrow \\ \hline \leftarrow \mathbf{0} \rightarrow \end{array}\right),$$

$$\boldsymbol{b}^1 = \left(\mathbf{0}^{1\times 4\log_2 c}, (\mathbf{1}-\boldsymbol{d})\rightarrow\right), \boldsymbol{W}_1^{out} = \left(\begin{array}{c} \boldsymbol{I}^{4\log_2 c\times 4\log_2 c}, \mathbf{0}\rightarrow \\ \hline \mathbf{0}^{1\times 4\log_2 c}, \boldsymbol{t}_1, 1, \mathbf{0}\rightarrow \\ \hline \vdots \\ \hline \mathbf{0}^{1\times 4\log_2 c}, \boldsymbol{t}_T, 1, \mathbf{0}\rightarrow \end{array}\right).$$

   This step maintains the first three column blocks, repeats the third column block into the fourth block, puts the tail of a relation into the fifth column block if column blocks one and two are a valid head-relation pair, and in the next column puts a 0/1 flag where 1 means there was a match and 0 meaning no match. At this point, a row where we have a valid match looks exactly as desired, although a row that did not have a match should not have any non-zero values in the 4th column block.

2. INPUT: $\boldsymbol{y}_1$
   OUTPUT: $\boldsymbol{y} \in \mathbb{R}^{N\times d}$ where $\boldsymbol{y} = \texttt{KeepNegatives}(f_2, 0, 0, \boldsymbol{y}_1)$ and f_2 is defined as:

$$f_2 = \texttt{NoResidual}(\boldsymbol{W}_2^{in}, \mathbf{0}^{d\times d'}, \boldsymbol{I}^{d'\times d}, \mathbf{0}^{1\times d'}, \mathbf{0}^{1\times d'}; \boldsymbol{y}_1).$$

where $\boldsymbol{W}_2^{in} \in \mathbb{R}^{d \times d'}$ is defined as:

$$\boldsymbol{W}_2^{in} = \left(\begin{array}{c} \leftarrow \mathbf{0} \rightarrow \\ \hline \mathbf{0}^{1\times 3d'}, \mathbf{1}^{1\times d'}, \mathbf{0} \rightarrow \\ \hline \leftarrow \mathbf{0} \rightarrow \end{array}\right) \boldsymbol{b}^1 = \left(\mathbf{1}^{1\times 3\log_2 c}, \mathbf{0}^{1\times \log_2 c}, \mathbf{1} \rightarrow\right).$$

This step zeros out the 4'th column block for the rows that didn't have a match, producing our final output.

□

Proposition B.11. *A constant-depth block of MoE layers can implement* `Lookup` *for all T triples from a matching graph $\mathcal{G}(E, R, T)$ using $\mathcal{O}(T \log c)$ parameters and $\mathcal{O}(\sqrt{T} \log c)$ FLOPs.*

Proof. We split the graph $\mathcal{G}$ into a set of $\sqrt{T}$ distinct subgraphs $\{\mathcal{G}_i(E, R, \sqrt{T})\}_{i=1}^{\sqrt{T}}$. Each subgraph $\mathcal{G}_i$ is assigned a unique binary key $\boldsymbol{k}_i \in \{-1, 1\}^{\log \sqrt{T}}$. Each triple $(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t})$ is then assigned to a subgraph by matching the first $\log \sqrt{T}$ dimensions of $\boldsymbol{h}$ against the keys $\{\boldsymbol{k}_i\}_{i=1}^{\sqrt{T}}$.

By Proposition B.10, we can perform $\texttt{Lookup}_{\mathcal{G}_i}$ over any graph $\mathcal{G}_i(E, R, \sqrt{T})$ with a constant-depth block of layers using $\mathcal{O}(\sqrt{T} \log c + \log^2 c)$ parameters and FLOPs.

Note that changing the graph from $\mathcal{G}_i$ to $\mathcal{G}_j$ only affects the weights of the GLU (the attention part of the layer is unaffected). This means we can create a single constant-depth block of MoE layers with $\sqrt{T}$ experts that contains all of the weights necessary to compute $\texttt{Lookup}_{\mathcal{G}_i}$ for all $\sqrt{T}$ subgraphs $\mathcal{G}_i$. If the MoE routing distribution is $s[i] = 1$ fix, this block of MoE layers exactly performs $\texttt{Lookup}_{\mathcal{G}_i}$ for any i using $\mathcal{O}(T \log c + \log^2 c)$ parameters and $\mathcal{O}(\sqrt{T} \log c + \log^2 c)$ FLOPs.

For any arbitrary lookup query $(\boldsymbol{h}, \boldsymbol{r})$ assigned to $\mathcal{G}_i$, we can achieve the desired routing distribution by using $k = 1$ and setting the routing matrix $\boldsymbol{W}^{\text{switch}} \in \mathbb{R}^{d \times \sqrt{T}}$ to be

$$\boldsymbol{W}^{\text{switch}} = \begin{pmatrix} \uparrow & \uparrow & & \uparrow \\ \boldsymbol{k}_1 & \boldsymbol{k}_2 & \ldots & \boldsymbol{k}_{\sqrt{T}} \\ \downarrow & \downarrow & & \downarrow \end{pmatrix}.$$

□

Proposition B.12. *A constant-depth(9) block of layers with $\mathcal{O}(\log^2 c)$ parameters and $\mathcal{O}(q \log^2 c)$ FLOPs can perform* `substitute`.

Proof. Here are the steps to implement `substitute`.

1. INPUT: $\boldsymbol{x} \in \mathbb{R}^{N \times d}$
   OUTPUT: $\boldsymbol{y}_1 \in \mathbb{R}^{N \times d}$ where $\boldsymbol{y}_1 = \texttt{KeepNegatives}(f_1, 0, 0, \boldsymbol{x})$ where:

$$f_1 = \text{Layer}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{B}, \mathbf{0}^{d\times d'}, \mathbf{0}^{d\times d'}, \mathbf{0}^{d'\times d}, \mathbf{0}^{1\times d'}, \mathbf{0}^{1\times d'}; \boldsymbol{x}).$$

   For the attention portion of the layer, $\boldsymbol{Q} \in \mathbb{R}^{d\times d}, \boldsymbol{K} \in \mathbb{R}^{d\times d}, \boldsymbol{V} \in \mathbb{R}^{d\times d}, \boldsymbol{B} \in \mathbb{R}^{N\times N}$ are defined as:

$$\boldsymbol{V} = \left(\begin{array}{c} \boldsymbol{I}^{\log_2 c\times \log_2 c}, \mathbf{0}^{\log_2 c\times (d-\log_2 c)} \\ \hline \mathbf{0}^{(d-\log_2 c)\times d} \end{array}\right), \boldsymbol{K} = \left(\begin{array}{c} \mathbf{0}^{3\log_2 c\times d} \\ \hline \boldsymbol{I}^{\log_2 c\times \log_2 c}, \mathbf{0}^{\log_2 c\times d} \\ \hline \mathbf{0}^{(d-4\log_2 c)\times d} \end{array}\right), \boldsymbol{V} = \left(\begin{array}{c} \mathbf{0}^{4\log_2 c\times d} \\ \hline \boldsymbol{I}^{\log_2 c\times \log_2 c}, \mathbf{0}^{\log_2 c\times (d-\log_2 c)} \\ \hline \mathbf{0}^{(d-1)\times d} \end{array}\right),$$

$$\boldsymbol{B} = (\boldsymbol{d} - \mathbf{1})^{N\times N}.$$

   The attention layer uses the values in the first column block as queries, the 4'th column block as keys, and the 5'th column block as values. The GLU implements identity as it lets the residual pass through without any modification.

2. INPUT: $\boldsymbol{y}_1$
   OUTPUT: $\boldsymbol{y}_2 \in \mathbb{R}^{N \times d}$ Where $\boldsymbol{y}_2 = \texttt{KeepNegatives}(f_2, 0, 0, \boldsymbol{y}_1)$ where

$$f_2 = \texttt{NoResidual}(\boldsymbol{0}^{d \times d'}, \boldsymbol{W}_2^{gate}, \boldsymbol{I}^{d' \times d}, \boldsymbol{1}^{1 \times d'}, \boldsymbol{0}^{1 \times d'}; \boldsymbol{x}).$$

   Where

$$\boldsymbol{W}_1^{gate} = \begin{pmatrix} \boldsymbol{I}^{3\log_2 c \times 3\log_2 c} & \boldsymbol{0}^{3\log_2 c \times (d - 3\log_2 c)} \\ \boldsymbol{0}^{2\log_2 c \times d} & \\ \boldsymbol{0}^{\log_2 c \times 3\log_2 c} & \boldsymbol{I}_1^{\log_2 c \times \log_2 c} \quad \boldsymbol{e}_1^{(\log_2 c \times (d - 4\log_2 c))^\top} \\ \boldsymbol{0}^{(N - 6\log_2 c) \times d} & \end{pmatrix}.$$

   The GLU keeps the first 3 column blocks the same, overwrites the 4'th with the 5'th and takes a single bit of the 4'th and stores it in the first position of the 5'th. This bit is either -1 or 1 if there was a matching key due to our embedding format. We use this as a flag to in the later steps.

3. INPUT: $\boldsymbol{y}_2$
   OUTPUT: $\boldsymbol{y}_3 \in \mathbb{R}^{N \times d}$ Where $\boldsymbol{y}_3 = \texttt{KeepNegatives}(f_3, 0, 0, \boldsymbol{y}_2)$ where

$$f_3 = \texttt{InvertColumn}(4\log_2 c + 1, \boldsymbol{y}_2).$$

   This step inverts the flag mentioned in the previous step.

4. INPUT: $\boldsymbol{y}_3$
   OUTPUT: $\boldsymbol{y} \in \mathbb{R}^{N \times d}$ where $\boldsymbol{y} = \texttt{KeepNegatives}(f_4, 0, 0, \boldsymbol{y}_3)$ where:

$$f_4 = \texttt{NoResidual}(\boldsymbol{W}_4^{in}, \boldsymbol{I}^{d \times d}, \boldsymbol{W}_4^{out}, \boldsymbol{0}^{1 \times d'}, \boldsymbol{1}^{1 \times d'}; \boldsymbol{y}_3).$$

   Where $\boldsymbol{W}_4^{gate}, \boldsymbol{W}_4^{out}$ are defined as follows:

$$\boldsymbol{W}_4^{in} = \begin{pmatrix} \boldsymbol{0}^{4\log_2 c \times d'} \\ \boldsymbol{1}^{1 \times \log_2 c} \quad \boldsymbol{0}^{1 \times (d' - \log_2 c)} \\ \boldsymbol{0}^{(N - 4\log_2 c - 1) \times d'} \end{pmatrix},$$

$$\boldsymbol{W}_4^{out} = \begin{pmatrix} \boldsymbol{I}^{\log_2 c \times \log_2 c} & \boldsymbol{0}^{\log_2 c \times (d - \log_2 c)} \\ \boldsymbol{0}^{\log_2 c \times \log_2 c} & \boldsymbol{I}^{\log_2 c \times \log_2 c} \quad \boldsymbol{0}^{\log_2 c \times (d - 2\log_2 c)} \\ \boldsymbol{0}^{\log_2 c \times \log_2 c} & \boldsymbol{0}^{\log_2 c \times \log_2 c} \quad \boldsymbol{I}^{\log_2 c \times \log_2 c} \quad \boldsymbol{0}^{\log_2 c \times (d - 3\log_2 c)} \\ \boldsymbol{I}^{\log_2 c \times \log_2 c} & \boldsymbol{0}^{\log_2 c \times (d - \log_2 c)} \\ \boldsymbol{0}^{(d' - 4\log_2 c) \times d} & \end{pmatrix}.$$

   This masks the first column based on the flag, and moves and values that were matched into this column, producing the final output.

□

B.5. Upper Bounds for Path Queries

First, we upper bound the number of parameters and FLOPs required by a looped model.

Proposition B.13. *For a matching graph $\mathcal{G}(E, R, T)$, a looped model with $\mathcal{O}(T \log c + \log^2 c)$ parameters and $\mathcal{O}(q^2 T \log c + q \log^2 c)$ FLOPs can answer any path query $\mathcal{Q}$ with q hops over $\mathcal{G}$.*

Proof. Our high-level strategy is to construct a constant-depth stack of Layer that composes the `lookup` and `substitute` and loop it q times.

The looped-moe model is parameterized by a single constant-depth stack of Layer as defined in equation 16. By Proposition B.10, with a constant-depth block of layers, we can store and lookup all T distinct triples in the matching graph $\mathcal{G}$ using $\mathcal{O}(T \log c)$ parameters and $\mathcal{O}(T \log c)$ FLOPs. After looking up the value of a variable, we can replace all instances of that variable with its value using the `substitute` primitive. By Proposition B.12, with a constant-depth block of layers, we can perform substitution using $\mathcal{O}(\log^2 c)$ parameters and $\mathcal{O}(\log^2 c)$ FLOPs. We then loop the stack q times, producing the desired output. □

Theorem B.14. *For a matching graph $\mathcal{G}(E, R, T)$, a looped-moe model with $\mathcal{O}(T \log c + \log^2 c)$ parameters and $\mathcal{O}(q^2 \sqrt{T} \log c + q \log^2 c)$ FLOPs can answer any path query $\mathcal{Q}$ with q hops over $\mathcal{G}$.*

Proof. We use the same strategy as in Proposition B.13. However, by Proposition B.11 the `lookup` can now be implemented by an MoE using $\mathcal{O}(q\sqrt{T} \log c)$ FLOPs. □

Next, we provide upper bounds for the standard model.

Theorem B.15. *For a matching graph $\mathcal{G}(E, R, T)$, a standard model with $\mathcal{O}(qT \log c + \log^2 c)$ parameters and $\mathcal{O}(q^2 T \log c + q \log^2 c)$ FLOPs can answer any path query $\mathcal{Q}$ with q hops over $\mathcal{G}$.*

Proof. We use the same high-level strategy as in Proposition B.13. However, because standard models do not implement looping, we create a $\mathcal{O}(q)$-depth stack that simply repeats the stack q times.

□

B.6. Lower Bound for Path Queries

Finally, under some assumptions on model expressity, we provide a lower bound on the number of parameters that a standard model requires to perform path queries

Relation product models. We'll begin by describing the restricted class of models we consider, which we refer to as *Relation Product Models*. This is a restricted model class that includes a number of the most obvious solutions to the problem of storing and answering path queries. Importantly, the class of relation product models includes the models that we constructed above to prove the upper bounds in Theorem B.15 and Theorem B.14 as a special case. As a recap, the solutions above handle q-hop queries using q layers, each storing every relation separately. Inspired by Liu et al. (2023), we also want to consider "shortcut" solutions that use depth less than the number of hops ($L < q$). In this class of solutions, instead of storing each triple separately, we explicitly store the product of multiple relations in a single layer. In the extreme case, we can store all R^q possible products of q relations in a single layer.

Intuitively, these constant-depth solutions are unlikely to provide parameter reductions because there are exponentially-many ($R^{\tilde{q}}$) paths, leading to a blow-up in required layer width. In what follows, we provide formal guarantees that there is no way to do better than our upper bound of $O(qT \log c)$ parameters using a relation product model and standard architecture.

Now we will define relation product models more formally. We can represent any matching graph $\mathcal{G}(E, R, T)$ with R permutation matrices $\{\boldsymbol{P}_i\}_{i=1}^R$, where each $\boldsymbol{P}_i \in \{0, 1\}^{E \times E}$ is a matrix with 1-hot rows. The answer to any q-hop path query over a matching graph is given by simply multiplying a one-hot encoding of the start entity with q transition matrices. For example, if the starting entity i is represented by a one-hot vector $\boldsymbol{e}_i \in \{0, 1\}^E$, then the answer to the following two-hop path query is given by $\boldsymbol{e}^\top \boldsymbol{P}_3 \boldsymbol{P}_6$.

$$\textbf{select} \quad y \quad \textbf{where} \quad (e, r_3, x) \textbf{ and } (x, r_6, y)$$

The set of all such products — capturing all possible q-hop paths for all $q > 1$ — are the monomials over the set of permutation matrices.

We consider the class of models where each layer can be expressed as multiplication with a polynomial over the transition matrices of the knowledge graph. We'll refer to this class of models as a *Relation Product Model*.

Definition B.16 (Relation Product Model)**.** A model where each layer can only transform input entities $\boldsymbol{e}_i$ according to the following expression:

$$\boldsymbol{e}_i^\top \sum_{i=1}^{m} g_i(\boldsymbol{x}) \tilde{\boldsymbol{P}}_i \tag{24}$$

where each $\tilde{\boldsymbol{P}}_i = \prod_{j=1}^{R} \boldsymbol{P}_j^{y_{ij}}$ is a monomial over the transition matrices, $y_{ij} \in \mathbb{N}^0$ are non-negative integer powers, $\boldsymbol{x}$ is the full input, and $g_i : \mathbb{R}^E \to \mathbb{R}$ is an arbitrary function that produces coefficients of the polynomial dependent on the input $\boldsymbol{x}$.

A full L-layer relation product model can then only transform the starting entity $\boldsymbol{e}$ according to the following expression:

$$\mathbf{e}_i^\top \prod_{\ell=1}^{L} \sum_{i=1}^{m_\ell} g_{\ell i}(\boldsymbol{x}) \tilde{\boldsymbol{P}}_{\ell i} \tag{25}$$

Note that this definition of a relation product model is *agnostic* to the architecture and encoding scheme used. It simply restricts the kinds of transformations a layer can perform on entity. How that transformation is implemented would depend on the specific architecture and encoding scheme used.

Assumptions and Lower Bound. In our proof of the lower bound below, we rely on one assumption about the number of parameters that the standard architecture requires to implement a layer of a relation product model.

Assumption B.17. To implement the entity transformation $\boldsymbol{e}_i^\top \sum_{i=1}^{m} g_i(\boldsymbol{x}) \tilde{\boldsymbol{P}}_i$ over a matching graph $\mathcal{G}(E, R, T)$, requires $\Omega(mE)$ parameters.

Why is this a reasonable assumption? In Lemma B.18, we show that implementing m *arbitrary* permutations requires $\Omega(mE)$ parameters. In a relation product model, the permutations are not arbitrary, but products of k base permutationns. It's possible that there is a scheme that allows us to leverage the structure in the m products of k permutations with fewer than $\Omega(mE)$ parameters. However, because a single layer of the standard model includes a constant number of matrix multiplications, we conjecture that it is not possible to achieve a better than constant factor reduction in parameters.

Lemma B.18. *To implement the entity transformation $\boldsymbol{e_i}^\top \sum_{i=1}^{m} g_i(\boldsymbol{x}) \boldsymbol{P}_i$ over a matching graph $\mathcal{G}(E, R, T)$, we require $\Omega(mE)$ parameters.*

Each layer in a vacuum, that is without other layers, needs to be able to, given any set of m permutations, have its values set so that they can all be represented.

Proof. First, we show that it takes mE parameters to represent m distinct permutations. With E entities, there are $E!$ different choices of permutations, thus there are $\binom{E!}{m}$ choices of m distinct tuple choices. We take the logarithm of this as B bits can represent 2^B choices. When m is $\text{poly}(E)$,

$$\log \binom{E!}{m} = \Omega\left(mE \log E\right)$$

via Stirlings approximation. Further, with the assumption that each parameter is $\log E$ bits, the layer requires $\Omega(mE)$ number of parameters. □

Stirling's approximation for binomial. Stirlings approximation is:

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \tag{26}$$

Stirling's approximation for $\log(n!)$:

$$\begin{aligned}\log(n!) &= \Omega\left(\log\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right)\right)\\ &= \Omega\left(\log\left(\sqrt{2\pi n}\right) + \log\left(\left(\frac{n}{e}\right)^n\right)\right)\\ &= \Omega\left(\frac{1}{2}\log\left(2\pi n\right) + n\log\left(\frac{n}{e}\right)\right)\\ &= \Omega\left(\log n + n\left(\log n - \log e\right)\right)\\ &= \Omega\left(n\log n\right)\end{aligned}$$

Known bound for all $1 \leq a \leq b$:

$$\binom{a}{b} \geq \left(\frac{b}{a}\right)^a$$

Stirling's approximation for $\log\binom{E!}{m}$:

$$\binom{E!}{m} \geq \left(\frac{E!}{m}\right)^m.$$

Applying log to both sides we get

$$\log\binom{E!}{m} \geq \log\left(\left(\frac{E!}{m}\right)^m\right).$$

With log rules,

$$\log\binom{E!}{m} \geq m\log\left(\frac{E!}{m}\right).$$

With log rules,

$$\log\binom{E!}{m} \geq m\left(\log\left(E!\right) - \log\left(m\right)\right).$$

Since $E! > m$,

$$\log\binom{E!}{m} = \Omega\left(m\log\left(E!\right)\right).$$

Via Stirling's approximation,

$$\log\binom{E!}{m} = \Omega\left(mE\log E\right).$$

Finally, we provide a lower bound on the number of parameters required by a relation product model to answer all path queries over a matching graph.

Theorem B.19. *To answer all path queries $\mathcal{Q}$ with q hops over a matching graph $\mathcal{G}(E, R, T)$ with $R \geq e$, a relation product model using the standard architecture and $L \leq q$ layers requires $\Omega(qER)$ parameters.*

Proof. We use a counting argument to lower bound the number of parameters required by a standard model.

The number of monomial terms in the polynomial representing the model $\prod_{\ell=1}^{L}\sum_{i=1}^{m_\ell} g_{\ell i}(\boldsymbol{x})\tilde{\boldsymbol{P}}_{\ell i}$ (see Equation (25)) is at most $\prod_{\ell=1}^{L} m_\ell$. There are R^q possible path queries of length q over $\mathcal{G}$, each corresponding to a distinct monomial. Thus, in order for the model to answer all possible path queries, we require that

$$R^q \leq \prod_{\ell=1}^{L} m_\ell. \tag{27}$$

Let us assume for the sake of contradiction that $n < \beta \cdot qER$ for all $\beta > 0$. By Lemma B.18, we can bound the total number of parameters in the model n in terms of the number of monomials per layer m_ℓ

$$\alpha \cdot E\sum_{\ell=1}^{L} m_\ell \leq n < \beta \cdot qER,$$

where $\alpha > 0$ are constants. This is the same as

$$\sum_{\ell=1}^{L} m_\ell < qR.$$

By the inequality of arithmetic and geometric means, we have

$$\prod_{\ell=1}^{L} m_\ell \leq \left(\frac{1}{L} \sum_{\ell=1}^{L} m_\ell \right)^L < \left(\frac{qR}{L} \right)^L. \tag{28}$$

Further, assuming that $R \geq e$ and using the fact that $\frac{q}{L} - 1$ is non-negative when $q \geq L$, we have by Taylor's theorem that

$$R^{\frac{q}{L}-1} \geq e^{\frac{q}{L}-1} \geq 1 + \left(\frac{q}{L} - 1 \right) \geq \frac{q}{L}.$$

We can condense this inequality and rearrange to show:

$$R^{\frac{q}{L}-1} \geq \frac{q}{L},$$

or equivalently

$$R^{\frac{q}{L}} \geq \frac{qR}{L},$$

which is the same as

$$R^q \geq \left(\frac{qR}{L} \right)^L.$$

Combining the above inequalities with Equation (28) yields

$$\prod_{\ell=1}^{L} m_\ell < R^q,$$

which contradicts Equation (27).

□