Exploring Alternative Approaches to Language Modeling for Learning from Data and Knowledge

Yuxin Zi, Kaushik Roy, Vignesh Narayanan, Amit Sheth

Artificial Intelligence Institute, University of South Carolina
yzi@email.sc.edu, kaushikr@email.sc.edu, vignar@sc.edu, amit@sc.edu

Abstract

Despite their wide applications to language understanding tasks, large language models (LLMs) still face challenges such as hallucinations - the occasional fabrication of information, and alignment issues - the lack of associations with human-curated world models (e.g., intuitive physics or common-sense knowledge). Additionally, the black-box nature of LLMs makes it highly challenging to train them meaningfully in order to achieve desired behaviors. Specifically, the attempt to adjust LLMs' concept embedding spaces can be highly intractable, which involves analyzing the implicit impact on LLMs' numerous parameters and the resulting inductive biases. This paper proposes a novel architecture that wraps powerful function approximation architectures within an outer, interpretable read-out layer, which can be scrutinized to explicitly observe the effects of concept modeling during training of the LLM. This is in contrast with the gradient-based implicit mechanisms, which solely rely on modifications to the LLM parameters which, therefore, do not lend themselves to scrutiny. Through extensive experiments across both generative and discriminative language modeling tasks, we analyze the abilities of our proposed architecture in comparison to the state-of-the-art LLMs of comparable size. We further provide a qualitative analysis of the interpretable read-out layer, and visualize the concepts captured by this layer. Our findings show the potential of our approach for robust LLM hallucination control and enhanced alignment of LLMs with human expectations.

1 Introduction

Language modeling involves extracting extensive patterns from vast amounts of data and representing these patterns in high-dimensional vector spaces (Zhou et al. 2023). These vector spaces enable us to gauge the similarity or dissimilarity between various concepts by calculating the distances between their embedded vectors. For instance, the embeddings of different fruits, like apples, grapes, and watermelons, will be located within a small distance (e.g., ε) from each other. Consequently, the shared (such as being fruits) or distinct (such as the nutrition value) characteristics of these concepts are implicitly captured by the parameters of the language models that embed them. This implicit representation of concept features by language model parameters, such as fruits providing natural sugars, makes it challenging to align the model's parameter space to achieve desired outcomes (Huang et al. 2023). For instance, a knowledge graph (KG) provides information about the nutritional properties of fruits such as *apples*, *grapes*, and *watermelons* explicitly. Now, let us consider the scenario where we train an LLM to capture the concepts of *apples*, *grapes*, and *watermelons* for a specific task, e.g., designing a diet plan emphasizing high levels of antioxidants. Since *watermelons* do not possess significant levels of antioxidants, the LLM should learn the concept *watermelon* distinctly from fruits with high antioxidants. Instead of relying on implicit representations by the LLM's parameters, we can theoretically, explicitly enforce this categorization within the LLM's concept embedding space using the KG, to separate the concept fruits into fruits with antioxidants vs. fruits with natural sugars, through training on a corpus that contains these distinctions from different contexts (Sarzynska-Wawer et al. 2021). However, in this paper we propose an approach that does not rely on vector space modifications and the resulting parameter-level changes, in order to allow for interpretable modifications. We introduce an outer interpretable read-out layer to enable the explicit observation of modifications to a language model's concept embedding space. This layer is added as a last layer after the LLM's parametric architectural layers. In the subsequent sections, we develop and formalize the proposed approach. We then evaluate our approach on benchmark language modeling and understanding tasks. We show that our method performs comparably with state-of-the-art LLMs of comparable size while allowing a cleaner and more interpretable way to understand the LLM's concept representations.

2 The Interpretable Read-Out Layer

Let $g_\theta(x) : x \to \mathbb{R}^d$ be a function parameterized by θ which embeds a concept x as a d dimensional embedding vector. Let $f_\beta : g_\theta(x) \to y, y \in \mathbb{R}^C$ be a linear function $\beta^T g_\theta(x)$ which maps the embedding for x to a set of C target outcomes (e.g., C target classes for sentiment classification or C possible next concepts - words or tokens, for language generation).

The similarity of concepts in an embedding space depends on the target task. That is, after f_β is trained for a target task, we say two concepts x_i and x_j are similar if $||g_\theta(x_i) -$

$g_\theta(x_j)|| \leq \varepsilon$, where ε is some small number, and $||.||$ is an appropriate distance metric.

We introduce a non-parametric *interpretable read-out layer* over $g_\theta(x)$, denoted by $\Phi(x)$, which characterizes the concept x using $g_\theta(x)$ and other "support" concepts before computing f_β. In the next section, we define such "support" concepts in the interpretable read-out layer, and explain their roles for explicit concept modeling. Specifically, we discuss the core underpinnings of language modeling and subsequently examine how fundamental operations of state-of-the-art language models, i.e., transformers, formalize those underpinnings (Vaswani et al. 2017). Then, we generalize those operations to reveal that adding the interpretable read-out layer is the logical next-step for endowing the model with interpretability.

A Closer Look at Language Modeling, the Transformer Architecture and Defining the Interpretable Read-Out Layer $\Phi(x)$

Although so far we have only discussed about concept embeddings of single words, e.g., *apple*, a language model also embeds *concept-phrases* of arbitrary length, e.g., *a red apple with vitamin C*. The key idea in an LLM is that a set of co-occurring concepts that form a concept phrase act as each other's "support" in the LLM's concept embedding space during the computation of the concept embedding. Intuitively, the co-occurring concepts *apple*, and *vitamin C* "support" the interpretation of the individual tokens (*apple*, *vitamin* and *C*) as belonging together, i.e., being close in the LLM's embedding space. If the concept of *apple* were co-occurring with *Steve Jobs* in the concept phrase, the concept *Steve Jobs* would support a different interpretation of *apple* (i.e., embeddings for *apple*, *vitamin*, and *C* would not be close in this case). The core operation in a transformer, the self-attention operator, leverages this idea of "support" during computing for concept and concept phrase embeddings.

Let T be the set of all possible concepts in a language. Let $X = [x_1, x_2, \ldots, x_N]$ denote a concept phrase, which is an ordered list of concepts such that each $x_n \in X$ is an element in the set T. Thus the self-attention for the concept x_i, denoted by $\mathbf{SA_{x_i}}$ is computed as $\sum_j \sigma(\frac{q_i^T k_j}{\sqrt{d}})v_j = \sum_j \left(\frac{exp(\frac{q_i^T k_j}{\sqrt{d}})}{\sum_j exp(\frac{q_i^T k_j}{\sqrt{d}})}\right)v_j$. The q_i, k_j, and v_j are vectors computed using $\mathbf{W_q} g_\theta(x_i)$, $\mathbf{W_k} g_\theta(x_j)$, and $\mathbf{W_v} g_\theta(x_j)$, respectively ($g_\theta$ is as defined in Section 2). The matrices $\mathbf{W_q}$, $\mathbf{W_k}$, and $\mathbf{W_v}$ are square matrices of dimension $d \times d$. Notice that the self-attention computation for x_i depends on j other co-occurring "support" concepts. We now generalize this idea to obtain the interpretable read-out layer.

$$\begin{aligned}\mathbf{SA_{x_i}} &= \sum_j \sigma(\frac{q_i^T k_j}{\sqrt{d}})v_j = \sum_j \left(\frac{exp(\frac{q_i^T k_j}{\sqrt{d}})}{\sum_j exp(\frac{q_i^T k_j}{\sqrt{d}})}\right)v_j \\ &= \sum_j \left(\frac{exp(\frac{||q_i||^2 - ||q_i - k_j||^2 + ||k_j||^2}{2\sqrt{d}})}{\sum_j exp(\frac{||q_i||^2 - ||q_i - k_j||^2 + ||k_j||^2}{2\sqrt{d}})}\right)v_j\end{aligned} \tag{1}$$

Then, we substitute $q_i = -q_i' = -\mathbf{W_q} g_\theta(x_i)$, $k_j = -k_j' = -\mathbf{W_k} g_\theta(x_j)$, and $v_j = \mathbf{W_v} g_\theta(x_j)$. We also observe that $exp(\frac{-||l-m||^2}{\sqrt{d}})$ is the Gaussian kernel with bandwidth $\sqrt{d}$, denoted by the inner product $\phi(l)^T\phi(m)$, where $\phi(.)$ is the infinite-dimensional map (Gaussian kernel is the infinite-dimensional inner product). Thus, we rewrite Eq (1) as:

$$\begin{aligned}\mathbf{SA_{x_i}} &= \sum_j \left(\frac{\phi(0)^T\phi(\mathbf{W_q} g_\theta(x_i))\phi(\mathbf{W_k} g_\theta(x_j))^T\phi(\mathbf{W_q} g_\theta(x_i))\phi(0)^T\phi(\mathbf{W_k} g_\theta(x_j))}{\sum_j \phi(0)^T, \phi(\mathbf{W_q} g_\theta(x_i))\phi(\mathbf{W_k} g_\theta(x_j))^T\phi(\mathbf{W_q} g_\theta(x_i))\phi(0)^T\phi(\mathbf{W_k} g_\theta(x_j))}\right)\mathbf{W_v} g_\theta(x_j) \\ &= \sum_j \left(\frac{C||\phi(\mathbf{W_q} g_\theta(x_i))^T\phi(\mathbf{W_k} g_\theta(x_j))||^2}{C\sum_j ||\phi(\mathbf{W_q} g_\theta(x_i))^T\phi(\mathbf{W_k} g_\theta(x_j))||^2}\right)\mathbf{W_v} g_\theta(x_j), \quad C = \phi(0)^T\phi(0)\end{aligned} \tag{2}$$

$$= \sum_j \left(\frac{\Phi(\mathbf{W_q} g_\theta(x_i))^T\Phi(\mathbf{W_k} g_\theta(x_j))}{\sum_j \Phi(\mathbf{W_q} g_\theta(x_i))^T\Phi(\mathbf{W_k} g_\theta(x_j))}\right)\mathbf{W_v} g_\theta(x_j) \quad \Phi(.)^T\Phi(.) = ||\phi(.)^T\phi(.)||^2 \tag{3}$$

We replace $||\phi(.)^T\phi(.)||^2$ with a new inner product (kernel) $\Phi(.)^T\Phi(.)$ as the product of two kernels is still a kernel. In this way, we have introduced a read-out layer over g_θ and the notion of "support" concepts x_j for characterizing the concept x_i. In the next section, we will explain how we enable the explicit observation of language modeling outcomes by incorporating the interpretable read-out layer $\Phi(.)$ and the "support" concepts. It is important to note that although earlier methods have demonstrated the feasibility of constructing a kernel to instantiate the self-attention operation, they have yet to explicitly derive the specific form containing the Gaussian kernel as in this work (Tsai et al. 2019; Chowdhury et al. 2021). Furthermore, our objective is not to offer an alternative kernel re-formulation, but to achieve a formulation through which we can explicitly interpret the LLM's concept embeddings using "support" concepts, as we will detail in the following sections.

Language Modeling by Leveraging the Interpretable Read-Out Layer

Support Concepts Recall the target task of designing a diet plan rich in antioxidants from Section 1, using just the

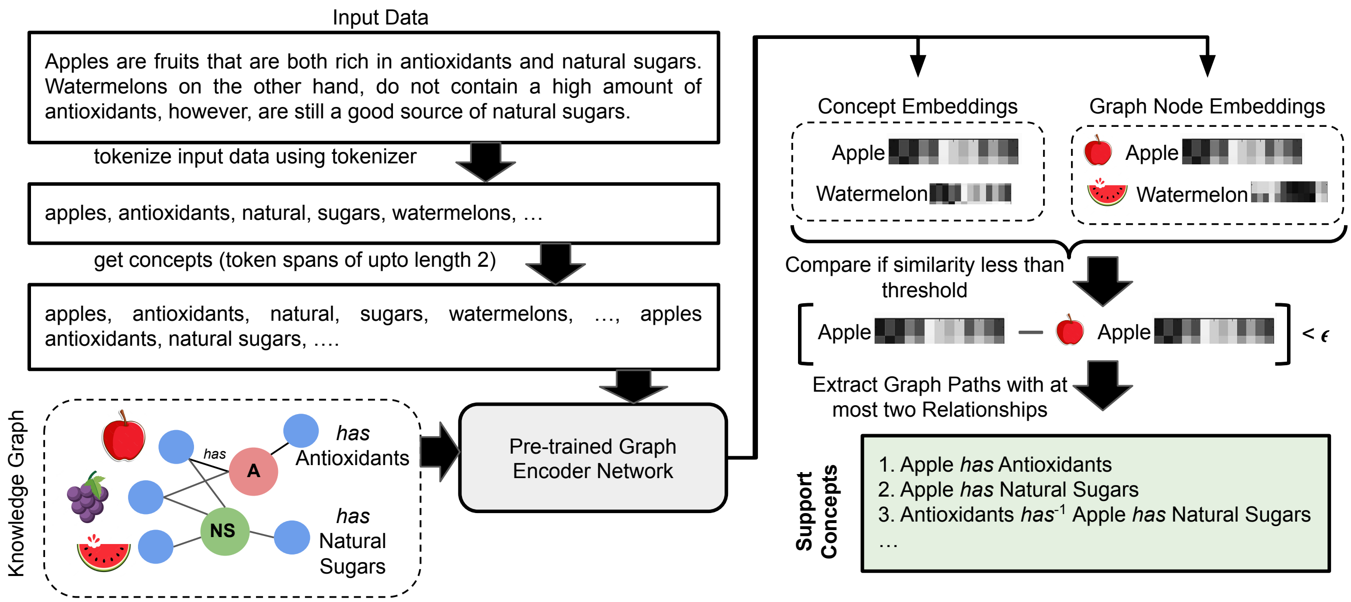

Figure 1: Process for deriving supportive concepts from data and an external KG. Initially, the data undergoes tokenization. Subsequently, token spans of length K are identified as potential concept candidates for retrieving graph paths from the KG (where K is set to 2 in the illustration). Graph paths are extracted by evaluating vector distances between concepts and graph nodes. These graph paths then serve as the resulting supportive concepts.

fruits - *apples*, *grapes*, and *watermelons*. We can obtain "support" concepts as paths from external KGs. For example, the KGs can have the triples *Apple is_a Fruit*, and *Apple has Antioxidants*, which can be reformulated as the path *Antioxidants has*$^{-1}$ *Apple is_a Fruit* using inverse relationships. Similarly, for *grape*, we have the path *Antioxidants has*$^{-1}$ *Grape is_a Fruit*. Thus, we can characterize the concepts *Apple* and *Grape* as being *fruits rich in antioxidants*, an appropriate description of a fruit category for the target task. Thus in this example, x_i in Eq (3) is the concept *apple* and the x_j are the KG paths, *Antioxidants has*$^{-1}$ *Apple is_a Fruit*, and *Antioxidants has*$^{-1}$ *Grape is_a Fruit*. Figure 1 depicts this process.

Layering the Interpretable Read-Out Layer over Different Parametric Function Approximation Architectures for Language Modeling In traditional parametric language modeling, the self-attention operations such as described in Eq (1) are layered on top of one another (e.g., 12 layers in BERT (Devlin et al. 2018)). In our approach, we instead lay the read-out layer $\Phi(.)$ over the parametric architecture g_θ (e.g., could be a 12-layer transformer architecture). This allows us to leverage the high-capacity function approximation capabilities of complex parametric architectures in g_θ, while still retaining the explicit interpretation of concept descriptions using "support" concepts, as explained in the previous section (Section 2). Also, we can now experiment with different parametric architectures for g_θ to achieve optimal language modeling performance. Figure 2 illustrates the idea of layering the interpretable read-out layer on top of parametric architectures and how the layer's output can be interpreted.

3 Definitions

In following sections, we introduce definitions required for explaining our method and experiments, then elaborate on the choices for the parametric function approximation g_θ and the interpretable read-out layer $\Phi(.)$.

Data Structure Definitions

Knowledge Graphs We formally define a KG and its paths (paths in the next section), as we use KG paths as "support" concepts in our experiments. A KG is denoted by $KG(\mathbf{V}, \mathbf{E}, \mathbf{L})$, where the sets $\mathbf{V}$ and $\mathbf{E}$ are the vertices and edges of the graph, respectively. The set $\mathbf{L}$ consists of relationships represented by the edges $e(v_1, v_2) \in \mathbf{E}$, $v_1, v_2 \in \mathbf{V}$. The relationships represented by the edges $e(v_1, v_2) \in \mathbf{E}$ are given by a set of N boolean-valued functions:

$$\mathbf{L} = \{r_{e(v_1,v_2)} : y_e \in \{0,1\}^N \mid e \in \mathbf{E}\} \tag{4}$$

Here 1 and 0 denotes whether the relationship $r_{e(v_1,v_2)}$ between vertices $v1, v2 \in V$ holds or not. We use $L(e)$ to denote the value y_e associated with the relationship $r_{e(v_1,v_2)}$. We use $subject(e)$ and $object(e)$ to denote the incident vertices $v1, v2$ for the edge $e(v1, v2)$. Figure 1 (bottom-left) shows an example of a KG where the vertices $\mathbf{V}$ is the set $\{apple, watermelon, grape, antioxidants, natural\ \ sugar\}$, the edges $\mathbf{E}$ is the set $\{has(apple, antioxidants), has(apple, natural\ sugar), \ldots\}$, and $\mathbf{L}$ is the set $\{apple\ \ has\ \ antioxidants\ : 1, apple\ has\ natural\ sugar\ : 1, \ldots\}$. Note that the set $\mathbf{L}$ is not completely specified, i.e., only the relationships that hold ($L(e) = 1$) are stored, and the ones that are not in $\mathbf{L}$ are assumed not to hold ($L(e) = 0$).

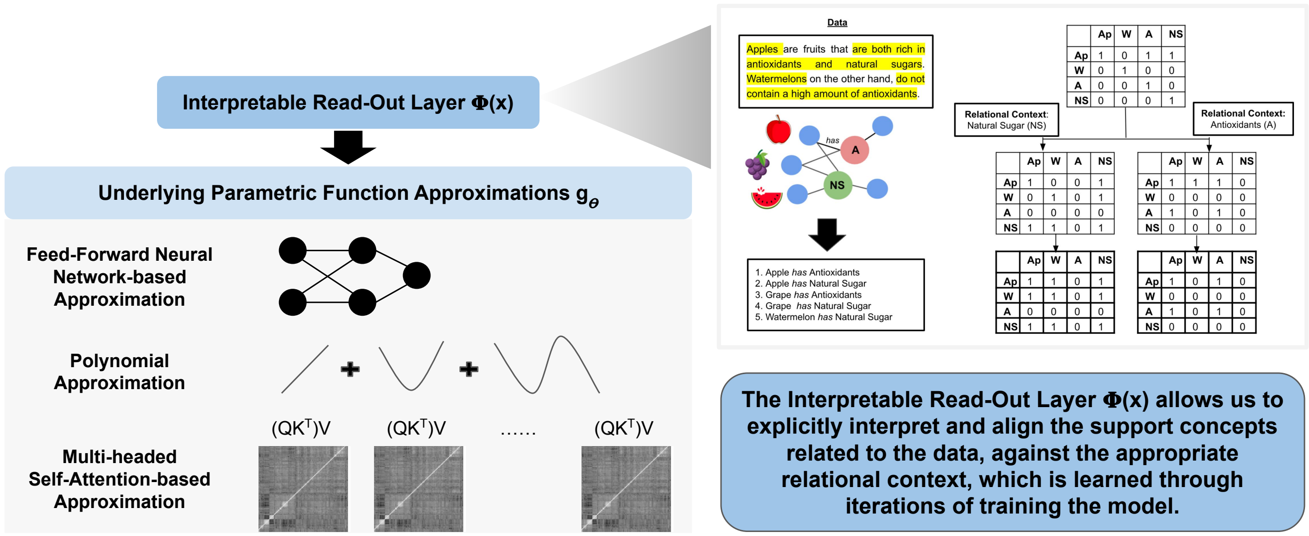

Figure 2: Shows different possible parametric function choices for g_θ under the interpretable read-out layer $\Phi(x)$, all of which have universal approximation properties. The figure also illustrates how the read-out layer can be interpreted as defining the appropriate relational context for the target task of designing a diet plan rich in antioxidants (Section 1), using *apples*, *grapes*, and *watermelons*. The relevant data-specific context is captured in $\Phi(.)$ throughout the training iterations of the model. This is depicted as an adjacency matrix (formed by utilizing thresholded dot product values calculated by Equation (2)).

Knowledge Graph Paths Given a KG denoted by $KG(\mathbf{V}, \mathbf{E}, \mathbf{L})$, a K length path $p_K(v_l, v_m)$ between two vertices $v_l, v_m \in \mathbf{V}$, is a sequence of edges $e1, e2, .., e_K \in \mathbf{E}$, such that $L(e_k) = 1 \wedge subject(e_1) = v_l \wedge object(e_K) = v_m \wedge object(e_k) = subject(e_{k+1}),\ k \in \{1, 2, \ldots, K\}$. We have shown examples of KG paths in Section 2.

Task Definitions

We experiment with generative (text generation) and discriminative (classification) modeling tasks. We describe these tasks and how a KG is used to solve the tasks formally in the following subsections.

Generative Modeling In generative modeling, given a context sequence of H previous concepts $[x_1, \ldots, x_H]$, the task is next-concept prediction, i.e., predicting the concept x_{H+1} from among a predefined vocabulary of C concepts. The forward pass for this computation is written using a modification to f_β (introduced in Section 2) and Eq 3 as:

$$
\begin{aligned}
&f_\beta(x_{H+1} \mid [x_1, \ldots, x_H]) \\
&= \beta \sum_j \left(\frac{(\mathbf{W_q} g_\theta(x_i))^T (\mathbf{W_k} g_\theta(x_j))}{\sum_j (\mathbf{W_q} g_\theta(x_i))^T (\mathbf{W_k} g_\theta(x_j))} \right) \mathbf{W_v} g_\theta(x_j), \\
&x_j \in \{x_1, \ldots, x_H\}
\end{aligned}
\tag{5}
$$

Here the "support" concepts are the set of concepts in context sequence $[x_1, \ldots, x_H]$. Note here that we set $\Phi(.)$ to be the identity function, thus reducing the inner product in Eq (3) to the standard scalar product in Eq (5), as this is what we experiment with. Crucially, this change still retains the explicit interpretation described in Section 2 as the inner-product is still the outermost layer over g_θ. We leave the exploration of different $\Phi(.)$ for future work.

Generative Modeling using Knowledge Graph paths as "support" concepts x_j in Eq (5) For a concept x and KG denoted by $KG(\mathbf{V}, \mathbf{E}, \mathbf{L})$, we define all KG paths of up to length K corresponding to concept x as

$$
\begin{aligned}
&P_K(x) \\
&= \{p_K(v_l, v_m) \mid k \leq K \ \ v_l, v_m \in \mathbf{V}, \\
&||g^{KGE}(x) - g^{KGE}(v_m)|| \leq \varepsilon\}
\end{aligned}
\tag{6}
$$

where $||g^{KGE}(x) - g^{KGE}(v_m)|| \leq \varepsilon$ denotes if embeddings for v_m and concept x are "close enough" using a KG embedding (KGE) model g^{KGE}. Let $\mathbb{I}_{KG}(x, x_j)$ be an indicator function that returns 1 or 0 indicating whether the "support" concept $x_j \in \Phi(x)$ or not, i.e., if the "support" concept is a path in the KG. Thus, to incorporate KG paths, we make a slight modification to Eq (5) as follows:

$$
\begin{aligned}
&f_\beta(x_{H+1} \mid [x_1, \ldots, x_H]) \\
&= \beta \sum_j \left(\frac{(\mathbf{W_q} g'_\theta(x_i))^T (\mathbf{W_k} g'_\theta(x_j))}{\sum_j (\mathbf{W_q} g'_\theta(x_i))^T (\mathbf{W_k} g'_\theta(x_j))} \right) \mathbf{W_v} g'_\theta(x_j) \\
&g'_\theta = \begin{cases} g_\theta & \text{if } \mathbb{I}_{KG}(x_{H+1}, x_j) = 0 \\ g^{KGE} & \text{if } \mathbb{I}_{KG}(x_{H+1}, x_j) = 1, \end{cases} \\
&x_j \in \{x_1, \ldots, x_H\} \cup P_K(x_{H+1})
\end{aligned}
\tag{7}
$$

Here the "support" concepts include both the set of concepts $[x_1, \ldots, x_H]$ and the KG paths in $P_K(x_i)$.

Discriminative Modeling In discriminative modeling, given a dataset of D data points, i.e., concept set and label pairs ($l_d = \{x_1, x_2, \ldots, x_H\}, m_d = c$), where the labels c are from among a predefined set of labels C, the task is to predict the label c using f_β. Let y denote the predicted variable for the class label c. The forward pass computation for the logit corresponding to label $c \in C$ is as follows:

$$\begin{aligned} & f_\beta(y = c \mid \{x_1, \ldots, x_H\}) \\ & = \beta \sum_j \left(\frac{(\mathbf{W_q} g_\theta(x_i))^T (\mathbf{W_k} g_\theta(x_j))}{\sum_j (\mathbf{W_q} g_\theta(x_i))^T (\mathbf{W_k} g_\theta(x_j))} \right) \mathbf{W_v} g_\theta(x_j), \\ & x_j \in \{l_1 \cup l_2 \cup \ldots \cup l_D\} \setminus \{x_1, \ldots, x_H\} \end{aligned} \tag{8}$$

Thus the "support" concepts for a datapoint during discriminative learning are concepts from all the other $D-1$ datapoints in the dataset. Extending the "support" concepts to the discriminative modeling case involves including the union of all the KG paths $P_K(x_j)$ corresponding to the support x_j.

Defining Approximation Architectures for g_θ

As illustrated in Figure 2, we experiment with different approximation architectures, all of which have universal approximation properties. They are feed-forward neural networks, polynomial approximations, and the multi-headed self-attention architecture.

Feed-Forward Neural Network First, we experiment with a Z layer feed-forward neural network described by:

$$\begin{aligned} & e_x = \mathbf{E}x,\ \mathbf{E} \in \mathbb{R}^d \\ & p_x = e_x + \mathbf{P_e}x,\ \mathbf{P_e} \in \mathbb{R}^d \\ & z_x = \max({\mathbf{W_z}}^T p_x, 0),\ \mathbf{W_1} \in \mathbb{R}^{d \times d_1},\ \mathbf{W_{z \setminus \{1,Z\}}} \in \mathbb{R}^{d_z \times d_{z+1}} \\ & \mathbf{W_Z} \in \mathbb{R}^{d_{Z-1} \times d},\ z \in \{1, 2, \ldots, Z\} \end{aligned} \tag{9}$$

The matrices $\mathbf{E}$, $\mathbf{P_e}$, and the $\mathbf{W_z}$ are the trainable weights in the network (the embedding matrix, the position encoding matrix, and the network weights and biases). We layer multiple feed-forward structures as described in Eq (9) (12 layers in our experiments, each layer with its own trainable weights) to obtain deeper approximation architectures. From each lower layer to the upper layer, we extract the last d dimensional column of z_x output at that lower layer. Finally, to obtain the d dimensional vector corresponding to $g_\theta(x)$, we extract the last column of z_x from the final layer.

Polynomial Approximation Next, we experiment with a polynomial approximation where we compute powers of x up to order J. This is described by the equations

$$\begin{aligned} & e_x = \mathbf{E}x,\ \mathbf{E} \in \mathbb{R}^d \\ & p_x = e_x + \mathbf{P_e}x,\ \mathbf{P_e} \in \mathbb{R}^d \\ & z_x = \{p_x^1, p_x^2, \ldots, p_x^J\} \end{aligned} \tag{10}$$

Each of the $\{p_x^1, p_x^2, \ldots p_x^J\}$ is computed dimension-wise (i.e., each of the d dimensions of x is raised to the power j). The matrices $\mathbf{E}$ and $\mathbf{P_e}$ are the trainable weights in the network (the embedding and position encoding matrices). Once again, we layer multiple such structures to obtain deeper approximation architectures. From each lower layer, we take the average of the polynomial powers at that layer to obtain a d dimensional vector to pass to the layer above it. Finally, to obtain the d dimensional vector corresponding to $g_\theta(x)$, we take the average of the polynomial powers at the last layer.

Multiheaded Self-Attention

$$\begin{aligned} & e_x = \mathbf{E}x,\ \mathbf{E} \in \mathbb{R}^d \\ & p_x = e_x + \mathbf{P_e}x,\ \mathbf{P_e} \in \mathbb{R}^d \\ & q_x^a, k_x^a, v_x^a = \{\mathbf{W_q^a}x, \mathbf{W_k^a}x, \mathbf{W_v^a}x \mid a \in \{1, 2, \ldots, A\}\} \\ & z_x = \{\sigma\left(\frac{(q_x^a)^T k_x^a}{\sqrt{d}}\right) v_x^a \mid a \in \{1, 2, \ldots, A\}\} \end{aligned} \tag{11}$$

The matrices $\mathbf{E}$, $\mathbf{P_e}$, and the $\mathbf{W_q^a}$, $\mathbf{W_k^a}$, and $\mathbf{W_v^a}$ are the trainable weights of the network. A denotes the number of attention heads. Again, multiple self-attention blocks are layered, as in the other two cases. From each lower layer, we take the average of the elements of z_x at that layer to obtain a d dimensional vector to pass to the layer above it. The final d dimensional vector corresponding to $g_\theta(x)$ is obtained from the average of the elements in z_x from the last layer.

4 Experiments

In this section, we describe our hyperparameter configurations and experiments for generative and discriminative modeling in Sections 4, 4 and 4, respectively. Due to space concerns, the table and figure captions contain the discussion about all the experiments. ***We provide the GLUE leaderboard result for context for the numbers in the results tables. However, it should be noted that the leaderboard models are up to 10 times larger than the models implemented in this paper. We will compare larger models in future work when the models have finished training.***

Hyperparameter Configurations

For all our experiments, we use a single A100 GPU. For the feed-forward neural network described in Section 3, we set $d = 384$ (chosen by tuning from the set $\{200, 384, 768\}$), and $d_z = 4000$ (chosen by tuning from the set $\{500, 1000, 2000, 4000\}$). For the polynomial approximation described in Section 3, we set $d = 384$ (chosen by tuning from the set $\{200, 384, 768\}$), and $J = 5$ (chosen by tuning from the set $\{2, 3, 4, 5\}$). Finally, for the multi-headed-self-attention-based network described in Section 3, we set $d = 384$ (chosen by tuning from the set $\{200, 384, 768\}$), and $A = 12$ (chosen by tuning from the set $\{4, 8, 12\}$). These form the basic units, and to get deep architectures, we layer them 4, 4, and 6 times for the feed-forward approximation, polynomial approximation, and self-attention-based approximation, respectively (chosen by tuning from the set $\{4, 6, 12\}$). We also include layer normalization between the layers. We use a train-validation split of 80-20 for all our experiments, and all reported results are evaluation loss scores.

Generative Modeling

Context Size, Tokenization, and Batch Size: For all models, the context size is set to 1024 (chosen by tuning from the set $\{256, 512, 1024\}$). Batch size is set to 32 (chosen by tuning from the set $\{8, 16, 32\}$). For tokenization, we use the GPT-2 tokenizer [1], which consists of $50,257$ tokens (C in Section 3). *Embeddings for the g^{KGE} from Eq (7):* We use ConceptNet Numberbatch Embeddings[2] and EWISE WordNet Embeddings[3] for ConceptNet and WordNet, respectively.

Parameter Initializations: All the parameter matrices for all three methods are randomly initialized. Tables 1 and 2 show the results.

Datasets and Knowledge Graphs We experiment with two text generation tasks. One, we train models for text generation in the style of Shakespeare by using the tiny-Shakespeare dataset[4], which consists of $338,025$ tokens. We use a train-evaluation split of 80% and 20%, respectively. Second, we train models for autocomplete (next-word prediction) using the OpenWebText dataset[5], which consists of $\sim$9 Billion $(9,040,017,095)$ tokens. For the first task, we used the KGs WordNet[6] and ConceptNet[7], respectively. The relationships across both include: *Antonym, DistinctFrom, EtymologicallyRelatedTo, LocatedNear, RelatedTo, SimilarTo, Synonym, AtLocation, CapableOf, Causes, CausesDesire, CreatedBy, DefinedAs, DerivedFrom, Desires, Entails, ExternalURL, FormOf, HasA, HasContext, HasFirstSubevent, HasLastSubevent, HasPrerequisite, HasProperty, InstanceOf, IsA, MadeOf, MannerOf, MotivatedByGoal, ObstructedBy, PartOf, ReceivesAction, SenseOf, SymbolOf, and UsedFor* (Speer, Chin, and Havasi 2017).

Results Table 1 shows the results on the tiny-Shakespeare dataset. We see that multi-headed-self-attention architecture takes only 5 minutes and converges to the least evaluation loss. Across the board, including KGs results in worse performance. The polynomial fit converges the fastest in terms of the number of epochs but the slowest in terms of the number of minutes. Finally, we see that the difference in the evaluation losses with and without KG is significantly smaller using the polynomial approximation method.

Table 2 shows the results on the OpenWebText dataset. We see similar trends as in the tiny-Shakespeare dataset. The multi-headed-self-attention architecture takes the least amount of days and achieves the least evaluation loss. Once again, including KGs consistently results in worse performance. The polynomial fit converges the fastest in terms of the number of epochs but the slowest in terms of the number of days. The polynomial approximation also again shows the least difference in the evaluation losses with and without KGs compared to the other two methods. Note that GPT-2 is fine-tuned, and our models are trained from scratch.

Model	w/o KG	/w KG	#Ep	Mins
Feed-forward Network	3.2	7.1	300	10
Polynomial Approximation	2.5	**4.2**	**100**	22
Multi-head Attention	**1.55**	6.8	200	5
GPT-2-Large-Fine-Tuned	2.2	6.7	250	**4**

Table 1: Results on the tiny-Shakespeare dataset

Model	w/o KG	/w KG	#Ep	Days
Feed-forward Network	5.2	10.3	300	45
Polynomial Approximation	3.5	**6.2**	**200**	60
Multi-head Attention	**2.8**	9.5	300	32
GPT-2-Large-Fine-Tuned	3.2	8.7	250	**2**

Table 2: Results on the OpenWebText dataset

Discriminative Modeling

We experiment with General Language Understanding Evaluation (GLUE) benchmark Tasks - STS (Semantic Textual Similarity Benchmark), MNLI (Multi-genre Natural Language Inference), QNLI (Question Answering Natural Language Inference), WNLI (Winograd Natural Language Inference), RTE (Recognizing Textual Entailment), and QQP (Quora Question Pairs) (Wang et al. 2018). Table 3 shows the results.

Model	STS	QQP	QNLI	WNLI	MNLI	RTE
($\text{GLUE}_{\text{LEADER}}$)	93.5	**90.9**	**96.7**	97.9	**92.5**	**93.6**
OLM_{nn} w/o KG	85.71	86.11	89.9	89.5	75.3	85.1
OLM_{nn} /w KG	89.2	90.2	90.5	90.2	82.1	90.3
OLM_{pf} w/o KG	90.89	86.41	92.3	90.11	88.53	90.4
OLM_{pf} /w KG	**93.55**	90.51	95.56	**98.7**	92.08	92.3
OLM_{mha} w/o KG	88.7	86.2	90.3	91.2	86.3	87.3
OLM_{mha} /w KG	90.5	88.8	93.6	97.9	90.8	90.56

Table 3: Results on the GLUE Benchmark tasks. Here, OLM denotes our language model with the neural network (nn), polynomial-fit (pf), and multiheaded (mha) self-attention architecture, respectively. We find that across the board, adding "support" concepts from the KGs improves scores significantly. Interestingly also, the polynomial fit performs the best among the choices for g_θ.

Interpretability

We perform interpretability analysis for a few examples from the test set, specifically for the task of sentence sim-

[1]https://huggingface.co/docs/transformers/model_doc/gpt2
[2]https://github.com/commonsense/conceptnet-numberbatch
[3]https://github.com/malllabiisc/EWISE
[4]https://raw.githubusercontent.com/karpathy/char-rnn/master/data/tinyshakespeare/input.txt
[5]https://skylion007.github.io/OpenWebTextCorpus/
[6]https://wordnet.princeton.edu/
[7]https://conceptnet.io/

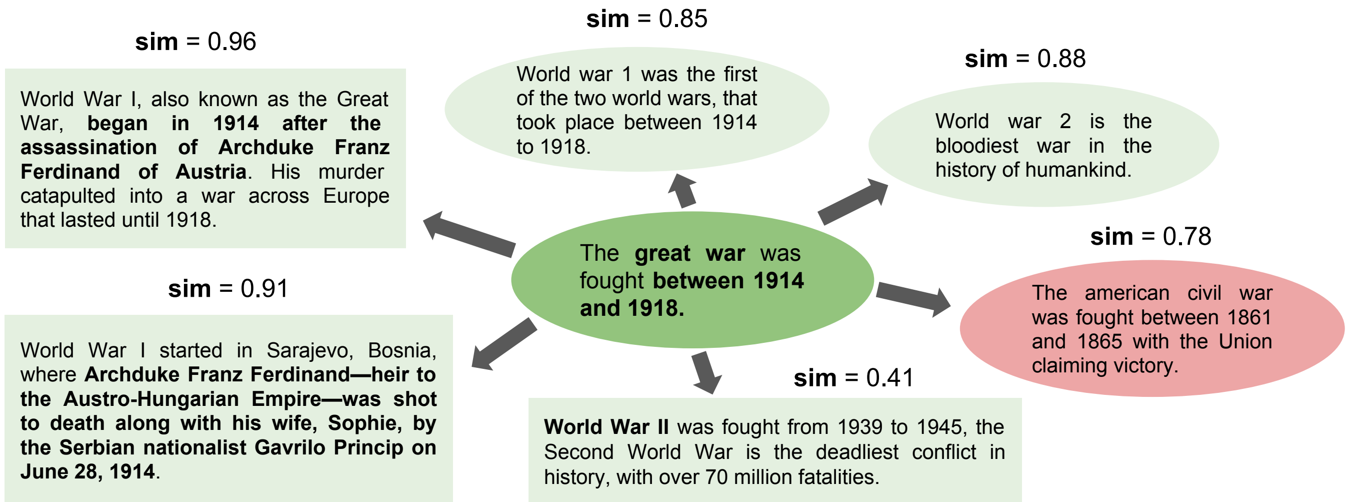

Figure 3: Interpretability results on a sentence similarity example. Since the read-out layer includes explicit inner products, we visualize and highlight in bold, the inner products from different models for a test example comparing sentence similarities against an anchor text (center). The squares green boxes represent the OLM_{pf} model, the green oval boxes represent the OLM_{mha} models, and the pink oval boxes represent the OLM_{nn} model. A significant advantage to using language modeling using our method is that we can directly use it to visualize the quality of the outputs. We find here also that a lot of the highlights correspond to relationships from ConceptNet (e.g., austria, 1914, austro-hungarian, great war, etc.), showing explicitly that KGs benefit performance in the discriminative modeling case.

ilarity. Figure 3 shows an example comparing different sentences talking about World War 1, World War 2, and the American Civil War.

5 Future Work

Additional Forms of Support Concepts

In Section 2, we explored the application of KGs in relation to support concepts. Moving forward, we will incorporate support concepts from Instructing Tuning datasets as described in the subsequent two paragraphs.

Support Concepts from Instruction Tuning Datasets Consider the (`Prompt`, `Instruction`) pair, (`Prompt`: *Given information <data>, give me a food item rich in antioxidants containing apples*, `Instruction`: *Here is a food item rich in antioxidants containing apples - A fruit salad with the fruits apples and grapes*), where an example of *<data>* is shown in Figure 2. x_i is the concept *apple*, x_j can be the "support" concepts in `Instruction` that affects the distribution of LLM's generated output given x_i.

A Note on Instruction-based Fine-Tuning of LLMs Instruction-based fine-tuning of LLMs using Reinforcement Learning with Human Feedback (RLHF) updates a policy function that uses proximal policy gradient-based methods to modify the LLM's output distribution (Ouyang et al. 2022). In the case of RLHF, the policy function for modifying the LLM can be seen as an interpretable read-out layer over the LLM. More formally, let $\pi : (x_i, x_j) \rightarrow [0,1]$ denote a distribution over different "support" concepts x_j's in the `Instruction` given concept x_i in the `Prompt`. The quantity $\left(\frac{\Phi(\mathbf{W_q} g_\theta(x_i))^T \Phi(\mathbf{W_k} g_\theta(x_j))}{\sum_j \Phi(\mathbf{W_q} g_\theta(x_i))^T \Phi(\mathbf{W_k} g_\theta(x_j))}\right)$ in Eq (2) also defines such a distribution. Thus RLHF achieves the objective of reducing the divergence between distributions π and $\left(\frac{\Phi(\mathbf{W_q} g_\theta(x_i))^T \Phi(\mathbf{W_k} g_\theta(x_j))}{\sum_j \Phi(\mathbf{W_q} g_\theta(x_i))^T \Phi(\mathbf{W_k} g_\theta(x_j))}\right)$, for eaxample, reducing the KL-divergence in RLHF using proximal policy gradient-based methods.

Efficient Inner Product Implementations

Since we derive an explicit inner product or kernel formulation in the interpretable read-out layer, we can exploit high-quality sub-quadratic approximations to kernels to significantly speed up learning and inference (Poli et al. 2023; Hallgren 2021).

6 Conclusion

In this paper, we explore alternative perspectives to language modeling and show that it is just as effective as parametric approaches to language modeling. Moreover, we show that it adds the benefit of easy visualization and interpretation. Thus, the methods described in this paper have the potential for implementing mitigation strategies with respect to observed negative effects in language models, such as hallucination and alignment issues.

Acknowledgements

This research is partially supported by NSF Award 2335967 "EAGER: Knowledge-guided neurosymbolic AI with guardrails for safe virtual health assistants" (Sheth et al. 2021, 2022; Sheth, Roy, and Gaur 2023).

References

Chowdhury, S. P.; Solomou, A.; Dubey, A.; and Sachan, M. 2021. On learning the transformer kernel. *arXiv preprint arXiv:2110.08323*.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Hallgren, F. 2021. Kernel PCA with the Nystr\" om method. *arXiv preprint arXiv:2109.05578*.

Huang, S.; Dong, L.; Wang, W.; Hao, Y.; Singhal, S.; Ma, S.; Lv, T.; Cui, L.; Mohammed, O. K.; Liu, Q.; et al. 2023. Language is not all you need: Aligning perception with language models. *arXiv preprint arXiv:2302.14045*.

Ouyang, L.; Wu, J.; Jiang, X.; Almeida, D.; Wainwright, C.; Mishkin, P.; Zhang, C.; Agarwal, S.; Slama, K.; Ray, A.; et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744.

Poli, M.; Massaroli, S.; Nguyen, E.; Fu, D. Y.; Dao, T.; Baccus, S.; Bengio, Y.; Ermon, S.; and Ré, C. 2023. Hyena Hierarchy: Towards Larger Convolutional Language Models. *arXiv preprint arXiv:2302.10866*.

Sarzynska-Wawer, J.; Wawer, A.; Pawlak, A.; Szymanowska, J.; Stefaniak, I.; Jarkiewicz, M.; and Okruszek, L. 2021. Detecting formal thought disorder by deep contextualized word representations. *Psychiatry Research*, 304: 114135.

Sheth, A.; Gaur, M.; Roy, K.; and Faldu, K. 2021. Knowledge-intensive language understanding for explainable ai. *IEEE Internet Computing*, 25(5): 19–24.

Sheth, A.; Gaur, M.; Roy, K.; Venkataraman, R.; and Khandelwal, V. 2022. Process knowledge-infused ai: Toward user-level explainability, interpretability, and safety. *IEEE Internet Computing*, 26(5): 76–84.

Sheth, A.; Roy, K.; and Gaur, M. 2023. Neurosymbolic Artificial Intelligence (Why, What, and How). *IEEE Intelligent Systems*, 38(3): 56–62.

Speer, R.; Chin, J.; and Havasi, C. 2017. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31.

Tsai, Y.-H. H.; Bai, S.; Yamada, M.; Morency, L.-P.; and Salakhutdinov, R. 2019. Transformer dissection: a unified understanding of transformer's attention via the lens of kernel. *arXiv preprint arXiv:1908.11775*.

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

Wang, A.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; and Bowman, S. R. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Zhou, C.; Li, Q.; Li, C.; Yu, J.; Liu, Y.; Wang, G.; Zhang, K.; Ji, C.; Yan, Q.; He, L.; et al. 2023. A comprehensive survey on pretrained foundation models: A history from bert to chatgpt. *arXiv preprint arXiv:2302.09419*.